A Formal Justification of a Design Rule for Avoiding Post-completion Errors

Paul Curzon and Ann Blandford

Technical Report:IDC-TR-2003-005
December 2003

Interaction Design Centre

Interaction Design Centre
School of Computing Science
Middlesex University
Trent Park Campus
Bramley Road
N14 4YZ

For further details of this technical report series contact:
Paul Curzon (p.curzon@mdx.ac.uk)

A Formal Justification of a Design Rule for Avoiding Post-completion Errors

Paul Curzon[1] and Ann Blandford[2]

[1]Middlesex University, Interaction Design Centre, Bramley Road, London N14 4YZ
[2]University College London Interaction Centre, 26 Bedford Way, London WC1H 0AP
p.curzon@mdx.ac.uk, a.blandford@ucl.ac.uk

Abstract. Interactive systems combine a human operator with a computer component. Either may be a source of error. The verification processes used must ensure not only the correctness of the computer component, but also that it minimizes the risk of human error. Human-centred design aims to do this by designing systems in a way that make allowance for human frailty. One approach to such design is to adhere to design rules. Design rules proposed, however, are often ad hoc. We examine how a formal cognitive model, encapsulating results from the cognitive sciences, can be used to justify such design rules in a way that integrates their use with existing formal hardware verification techniques. This approach provides a lightweight use of formal proof that is also integrated with fully formal verification approaches. We consider here the verification of a design rule intended to prevent a commonly occurring class of human error know as the post-completion error. All proof has been machine-checked using the HOL proof system.

Keywords

Formal cognitive architecture, user error, design rules, formal verification.

1 Introduction

Interactive computer systems are systems that combine a human operator with a computer system. Such a system needs to be both correct and usable. With the increasing ubiquity of interactive computer systems, usability becomes increasingly important. Minor usability problems can scale to having major economic and social consequences. Usability has many aspects. We concentrate on one aspect: user error. Humans are naturally prone to error. Though such error is not predictable in the way the behaviour of a faulty computer may be, much human error is systematic and as such can be modelled and reasoned about.

Design approaches to prevent usability problems often tend to be ad hoc: following lists of design rules, sometimes apparently contradictory, that are based on the experience of HCI experts. Furthermore the considerations of usability experts are often far removed from those of computer system verification approaches, where the emphasis is on correctness of the system against a functional

specification. In this paper we consider how the two worlds of formal computer system verification and human-centred usability verification can be integrated. We propose a way in which usability design rules can be both formalised and derived from formalised principles of cognition within a framework that integrates with traditional verification techniques. Our approach integrates usability verification with computer system verification. It also integrates lightweight use of formal proof outside the design cycle with fully formal, but more time-consuming, verification approaches within the design cycle. We illustrate the approach by considering a well studied and widely occurring class of human error known as the post-completion error. As described more fully below, a post-completion error occurs when a user achieves their main goal but omits 'clean up' actions; examples include making copies on a photocopier but forgetting to retrieve the original, filling the car with fuel but failing to replace the fuel cap, and forgetting to take change with the goods from a vending machine.

We first define simple **principles of cognition**. These are principles that generalise the way humans act in terms of the mental attributes of knowledge, tasks and goals. The principles considered do not cover the full range of human cognition. Rather they focus on particular aspects of cognitive behaviour of interest. They do not describe a particular individual, but generalise across people as a class. They are each backed up by evidence from HCI and/or psychology studies. Those presented are not intended to be complete but to demonstrate the approach. We have developed a **formal model** of these principles in higher-order logic. This description is a generic formal cognitive model. By "generic" we mean that it can be targeted to different tasks and interactive systems. It is thus strictly a simple cognitive architecture [17]. In the remainder of the paper we will refer to the generic model as a *cognitive architecture* and use the term *cognitive model* for a version of it instantiated for a given task and system. The underlying principles of cognition are formalised once in the architecture, rather than having to be re-formalised for each new task or system of interest. Whilst higher-order logic is not essential for this, its use makes the formal specifications simpler and more natural than the use of a first-order logic would. Here we use it to make precise the general principles considered, to allow us to then reason about their consequences with respect to user error and design rules. Combining the principles of cognition into a single architecture rather than formalising them separately allows reasoning about their interaction.

The principles, and more formally the cognitive architecture, specify **cognitively plausible behaviour** (see [7]). That is, they specify possible traces of user actions that can be justified in terms of the specific principles. Of course users might also act outside this behaviour, about which situations the model says nothing. Its predictive power is bounded by the situations where people act according to the principles specified. That does not preclude useful results from being obtained, provided their scope is remembered. The architecture allows us to investigate what happens if a person does act in such plausible ways. The behaviour defined is neither "correct" nor "incorrect". It could be either depending on the environment and task in question. It is, rather, "likely" behaviour.

After describing the architecture, we next consider post-completion errors. They are a class of error that people make systematically, if not predictably, in a wide range of situations. We also describe a simple and well known usability design rule that, if followed, eliminates this class of error; broadly speaking, the rule states that the machine will prevent the user from achieving their main goal without completing all the key task steps first. We give a formal specification of the design rule and also formally specify what it means for a post-completion error to occur in terms of the cognitive architecture. We prove a theorem that states that if the design rule is followed, then the erroneous behaviour cannot occur *due to the specified cause* as a result of a person behaving according to the principles of cognition as formalised. All theorems have been machine-checked within the HOL interactive proof system [16].

The design rule is initially formalised in user-centric terms. To enable the integration with machine-centric verification, we next reformulate it in machine-centric terms, ultimately proving that a machine-centric version of the design rule implies the absence of post completion errors. Even though the cognitive architecture is capable of making the error, the design rule ensures that the user environments (as provided by the computer part of the system) in which it would emerge do not occur. Other errors are, of course, still possible. The design rule is well known and our contribution is not the rule itself, but its generic formalisation and the generic theorems justifying it from a formalisation of a small set of principles. More generally our contribution is to demonstrate an integrated way that such reasoning about design rules can be achieved. Because the design rule is formalised, we can be precise about its scope of applicability. This raises the possibility in further work of, for example, unpacking the situations where different design rules appear contradictory at first sight.

2 Related Work

There are several approaches to formal reasoning about the usability of interactive systems. One approach is to focus on a formal specification of the user interface; Campos and Harrison [9] review several such techniques. Most commonly it is used with model-checking-based verification; investigations include whether a given event can occur or whether properties hold of all states. In contrast, Bumbulis *et al* [5] verified properties of interfaces based on a guarded command language embedded in the HOL machine-assisted proof system. Back *et al* [1] illustrate how properties can be proved and data refinement performed of a specification of an interactive system. However, techniques that focus on the interface do not directly support reasoning about design problems that lead to users making systematic errors; also, the usability properties checked are necessarily device-specific and have to be reformulated for each system verified, unlike in our work.

An alternative is formal user modelling of the underlying system. It involves writing both a formal specification of the computer system and one of the user, to support reasoning about their conjoint behaviour. Both system and user are

considered as central components of the system and modelled as part of the analysis. Doing so provides a conceptually clean method of bringing usability concerns into the domain of traditional verification in a consistent way. Examples of formal user modelling include the work of Duke *et al* [14], Moher and Dirda [22] and Paterno' and Mezzanotte [23]. Duke *et al* [14] express constraints on the channels and resources within an interactive system; this approach is particularly well suited to reasoning about interaction that, for example, combines the use of speech and gesture. Moher and Dirda [22] use Petri net modelling to reason about users' mental models and their changing expectations over the course of an interaction; this approach supports reasoning about learning to use a new computer system but focuses on changes in user belief states rather than proof of desirable properties. Paterno' and Mezzanotte [23] use LOTOS and ACTL to specify intended user behaviours and hence reason about interactive behaviour. Back *et al* [1] illustrate how properties can be proved and data refinement performed of a specification of an interactive system.

Our work complements these alternative uses of formal user modelling. None of the above focus on reasoning about user errors. Models typically describe how users are intended to behave: they do not address human fallibility. If verification is to detect user errors, a formal specification of the user, unlike one of a computer system, is not a specification of the way a user should be; rather, it is a description of the way they are [7]. Work that does take account of this includes that of Butterworth *et al* [6] and Rushby [26]. Butterworth *et al* used TLA to reason about reachability conditions within an interaction, while Rushby formalised plausible mental models of the system, looking for discrepancies between these and actual system behaviour. However, like interface-oriented approaches, each model is individually hand-crafted for each new device in this work.

An approach to interactive system verification that focuses directly on errors is exemplified by Fields [15]. He models erroneous actions explicitly, analysing the consequences of each possible action. He thus models the effect of errors rather than their underlying causes. A problem of this approach is the lack of discrimination about which errors are the most important to consider. It does not discriminate random errors from systematic errors which are likely to reoccur and so be most problematic. It also implicitly assumes there is a "correct" plan, from which deviations are errors.

The ongoing safeHCI project at the University of Queensland [21] has broadly similar aims and approach to our overall project, combining the areas of cognitive psychology, human-computer interaction and system safety engineering. The precise focus and details differ, however. SafeHCI has had a focus on hazard analysis and system-specific modelling, whereas our work has an emphasis on machine-assisted verification and generic cognitive models.

In a different tradition, approaches that are based on a cognitive architecture (e.g. the work of Kieras, Wood and Meyer [20]; Gray [18] and Ritter and Young [24]) model the underlying cognitive causes of errors. However, the modelling exemplified by these approaches is too detailed to be amenable to formal proof. Our previous work [11, 12] also followed this approach but at a coarser

level of detail, making formal reasoning using machine-checked proof tractable. In this approach general mechanisms of cognition are modelled and so need be specified only once, independent of any given interactive system [14]. This fact means a single generic model of user behaviour can be written. Furthermore, by explicitly doing the verification at the level of underlying cause, on failed verification, a much greater understanding of the problem is obtained. Rather than just knowing the manifestation of the error – the actions that lead to the problem – the failed proof provides understanding of the underlying causes.

In this paper, we use the cognitive architecture as the basis of formal machine-checked reasoning about interactive systems design in general, rather than to reason directly about specific systems. Our approach is similar to that of [2] in that we are working from a (albeit different and more formal) model of user behaviour to high level guidance. There the emphasis is on a semi-formal basis underpinning the craft skill in spotting when a design has usability problems. In contrast, we are concerned here with guidance for a designer rather than for a usability analyst. We focus on the verification of general purpose design rules rather than the interactive systems themselves.

Providing precision to ensure different people have the same understanding of a concept has been suggested as the major benefit of formal models in interaction design [4]. One approach would therefore be to just formalise the design rules (see [4], [25]). The benefit is the increased understanding of the design rule resulting from attempting to formalise it. In our approach, we not only formalise design rules, we also prove theorems justifying them based on underlying principles about cognition embodied in a formal cognitive architecture. In this way the design rules are formally demonstrated to be effective, up to the assumptions of the principles of cognition. This builds on our previous work where informal argument only was used to justify the effectiveness of design rules [13]. A contribution of this paper is to show how this can be formalised in a way that integrates with other forms of verification.

3 Formalising Cognitively Plausible Behaviour

Our cognitive architecture was developed by formally modelling principles of cognitively plausible behaviour. We do not model erroneous actions explicitly (as is done for example in [15]). Instead, erroneous actions emerge from an abstract description of cognitively plausible behaviour. The behaviour described in the architecture could correspond to correct or incorrect actions being taken, depending on the circumstances. The focus of the description is in terms of internal goals and knowledge of a user. This contrasts with a description of a user's actions as, say, a finite state machine that makes no mention of such cognitive attributes. The principles considered are: non-determinism; goal-based termination; task-based termination; reactive behaviour; communication goals; mental triggers; no-option-based termination; and relevance. These will be described in more detail below. The list is not intended to be exhaustive, but to cover a variety of classes of cognitive principles, based on the motor system, simple

knowledge based cognition, goal-based cognition, *etc.* Also, some of the principles are formalised in a simple way, our intention at this stage being to test the approach, rather than modelling the full richness of the principles. In future work we will increase the richness of the descriptions. In subsequent sections we refer to cognitively plausible behaviour when strictly meaning the subset of cognitively plausible behaviour embedded in the current version of our formalisation.

The cognitive architecture is specified by a relation `USER` that combines formalisations of cognitively plausible behaviour. It takes as arguments the various pieces of information such as the user's goal, actions that the user may take, *etc.* referred to in the description below. Here we give an overview of the cognitive architecture's formalisation in terms of the principles of cognition used; the formalisation is given in more detail in [12]. It is formalised in higher-order logic. We use standard mathematical notation in this paper.

The cognitive architecture is based on a series of non-deterministic temporally guarded action rules. Each describes an action that a user *could* rationally make. The rules are grouped corresponding to a user performing actions for specific cognitively related reasons. Each such group then has a single generic description. Each rule has a guard-action form. They state that if the guard holds at some point then the `NEXT` action taken by the user is that given. By *next* in this context we mean the first action of interest taken by the user after the current point in time. The rules each have the form: `guard t` $\wedge$ `NEXT flag actions action t`, stating that a `guard` is true at time `t` and the `NEXT` action performed from the list of actions relevant to the interaction (given by the list `actions`) is `action`. The action is identified by its position in the list of all actions. The `flag` argument is a specification artifact used to distinguish time between the instances where a rule is firing and so where no actions relevant to the interaction occur.

Non-determinism: In any situation, any one of several behaviours that are plausible might be taken. The separate behaviours are specified as rules. Each rule is formalised in the user model non-deterministically. That is, it is one of a series of options, any of which could be taken. The architecture does not assert that a rule will be followed, just that it may be followed. Below, we outline several such rules. They form the core of the cognitive architecture. By combining them, the architecture asserts that the behaviour of any rule whose guards are true at a point in time is cognitively plausible at that time. It cannot be assumed that any specific rule will be the one that the person will follow if several are cognitively plausible.

Goal-based termination behaviour: Cognitive psychology studies have shown that users intermittently, but persistently, terminate interactions as soon as their goal has been achieved [8]. It is formalised as a guarded rule as described above. We must supply a relation to the cognitive architecture that indicates over time whether the goal is achieved or not. This is referred to as `goalachieved`, in the formal definitions. We also use `finished`, to indicate whether the user considers the interaction to be over. With a vending machine, for example, this may correspond to the person walking away, starting a new interaction (perhaps by hitting a reset button), *etc.* Both `goalachieved` and `finished` are signals

that, given a time, return a boolean value indicating whether the goal is achieved or the interaction terminated (respectively) at that time. If the goal is achieved at a time then the next action of the cognitive architecture is to terminate the interaction: `goalachieved ustate t` $\wedge$ `NEXT flag actions finished t`.

As an example, consider instantiating the `goalachieved` argument of the user model (and other definitions and theorems developed below) for an interaction to do with obtaining chocolate from a vending machine. `UserHasChoc ustate` might be provided as the goalachieved argument to the user model. Here `UserHasChoc` is a state accessor function and `ustate` is a relation representing the user state over time. It would be specified as a tuple of signals over time indicating when each user action occurs and tracking the changes in possessions of the user. More details of instantiating the user model with this example can be found in [11]. Note that `goalachieved` is a higher-order function and can as such represent an arbitrarily complex condition. It might, for example, be that the user has a particular object as above, that the count of some series of objects is greater than some number or a combination of such atomic conditions. In specifying the cognitive architecture we just state that it is a boolean function whose value may vary over time. This makes use of the higher-order nature of the specification language.

Task-based termination behaviour: For the purposes of analysis, the architecture specifies that a user will terminate an interaction when their whole task is achieved. In achieving a goal, subsidiary tasks are often generated. For the user to complete the task associated with their goal they must also complete all subsidiary tasks. Examples of such tasks with respect to a vending machine, for example, include taking back a credit card or taking change. One way to specify these tasks would be to explicitly describe each such task. Instead we use the more general concept of an interaction invariant [12]. The underlying reason for these tasks being performed is that in interacting with the system some part of the state must be temporarily perturbed in order to achieve the desired task. Before the interaction is completed such perturbations must be undone. For example, to pay at a vending machine using a credit card requires the card being inserted and later returned. A condition on the state that holds at the start of the interaction – that the user has the card – must be restored by the end. We specify the need to perform these completion tasks indirectly by supplying the interaction invariant as a higher-order argument to the cognitive architecture. The interaction invariant is an invariant in a similar sense to a loop invariant in program verification. It is an invariant at the level of abstraction of whole interactions. Full task completion involves not only completing the user's goal, but also restoring the invariant by completing all the subsidiary tasks generated in the process.

For example, the invariant for use of a simple vending machine might be that the value of the user's possessions have been restored to their original value, having exchanged coins for chocolate of the same value. If `ustate` is the user state, then a relation of the form `USER_INVARIANT ustate` might be used as the argument where `USER_INVARIANT` is defined to be true at times when the value

of the user's possessions as given in the user state are the same as at the initial time. For example for a vending machine the possessions might be different forms of coins and chocolate. The user invariant is true at times when their combined value is restored. More details of this example can be found in [11].

We assume that on completing the task in this sense of goal achieved and invariant restored, the interaction will be considered terminated by the user, irrespective of any other possible actions apart from actions already mentally triggered (discussed below). This is modelled using an if construct rather than disjunction to give it priority. If both the goal has been achieved and the invariant restored then the user will terminate the interaction, irrespective of what other non-deterministic rules may potentially be active. Otherwise one of the non-deterministic rules will be fired.

`IF (invariant t)` $\wedge$ `(goalachieved t)`
`THEN NEXT flag actions finished t ELSE` *non-deterministic rules*

Reactive behaviour: A user may react to a stimulus or message from a device, doing the action suggested by the stimulus. For example, if a flashing light comes on next to the coin slot of a vending machine, a user might, if the light is noticed, react by inserting coins if it appears to help the user achieve their goal. Reactive behaviour is specified as a general class of behaviour: in a given interaction there may be many different stimuli to react to. Rather than specify this class of behaviour for each, we define the behaviour generically. `REACT` gives the rule defining what it means to react to a given stimulus.

`REACT flag actions stimulus action t =`
stimulus t $\wedge$ NEXT flag actions action t

If at time `t`, the specified `stimulus` is active, the NEXT action taken by the user out of the possible actions, `actions`, at an unspecified later time, may be `action`. As there may be a range of signals designed to be reactive, the user model is supplied with a list of stimulus-action pairs: [(s1, a1); ... (sn, an)]. A list recursive relation, given a list of such pairs, extracts the components and asserts the above rule about them. They are combined using disjunction in the recursive definition, so are non-deterministic choices, and this definition is combined with the other non-deterministic rules.

Communication goal behaviour: A user enters an interaction with knowledge of task dependent sub-goals that must be discharged. Given the opportunity, they may attempt to discharge any such *communication goals* [3]. The precise nature of the action associated with the communication goal may not be known in advance. A communication goal specification is a pre-determined partial plan that has arisen from knowledge of the task in hand independent of the environment in which that task will be accomplished. It is not a fully specified plan, in that no order of the corresponding actions may be specified. For example, when purchasing a ticket, in some way the destination and ticket type must be specified as well as payment made. The way that these must be done and their order may not be known in advance. However, a person enters an

interaction with the aim of purchasing a ticket primed for these communication goals to be addressed. If the person sees an apparent opportunity to discharge a communication goal they *may* do so. Once they have done so they will not expect to need to do so again. No fixed order is assumed over how communication goals will be discharged if their discharge is apparently possible. Communication goals are a reason why people do not just follow instructions.

Communication goals are modelled as guard-action pairs as for reactive signals. The guard describes the situation under which the discharge of the communication goal appears possible. It will include a label signal indicating that the input exists and that it corresponds to the desired action. As for reactive behaviour, a list of (guard, action) pairs is supplied to correspond to each communication goal processed by a recursive definition and included as a disjunct with the non-deterministic rules. This determines when a communication goal may be discharged. However, unlike the reactive signal list that does not change through an interaction, communication goals are discharged. This corresponds to them disappearing from the user's mental list of intentions. We model this by removing them from the communication goal list when done.

Mental triggers: A user commits to taking an action in a way that cannot be revoked after a certain point. Once a signal has been sent from the brain to the motor system to take an action, the signal cannot be stopped even if the person becomes aware that it is wrong before the action is taken. Rather than associate an external stimulus directly with an external action using the disjunctive rules, we associate them with mental "actions" that trigger the process of taking the actual action. Thus the actions in each of the rules described so far will not be externally visible actions, but internal mental actions. A further category of trigger rules is introduced that links the mental decision to the actual action. If one of the mental actions is taken on a cycle then the next action will be the externally visible action it triggers. The cognitive architecture is supplied with a guard-action pair list linking mental triggers with external actions. Mental triggers are given a higher priority than the non-deterministic rules. If a trigger is fired then it will be the next action taken.

No-option-based termination behaviour: A user may terminate an interaction when there is no apparent action they can take that would help complete the task. For example, if on a touch screen ticket machine, the user wishes to buy a weekly season ticket, but the options presented include nothing about season tickets, then the person might give up, assuming their goal is not achievable. The model includes a final default non-deterministic rule that models this case.

Relevance: A user will only take an action if there is something to suggest it corresponds to the desired effect. We do not currently model this explicitly: however, it is implicit in most of the rules. For example, communication goals and the termination rules are by definition only fired when relevant. In particular, the "label" signals referred to above are intended to address aspects of relevance.

Probes: The features of the cognitive architecture discussed above concern aspects of cognition. An extension of the architecture for this paper over that

of our previous work [11] involves the addition of "probes". Probes are an extra set of signals that do not alter the cognitive behaviour of the architecture, but instead make internal aspects of its action over time visible. This allows specifications to be written in terms of hidden internal cognitive behaviour, rather than just externally visible behaviour. This is important for this work, as our aim here is to formally reason about whether design rules address underlying cognitive causes of errors not just their physical manifestation. The form of probe we consider here records for each time instance whether a particular rule fires at that instance. We require here a single probe that fires when the goal-based termination rule described above fires. We formalise this using a function, `Goalcompletion` that extracts the goal completion probe from the collection of probes passed as an argument to the cognitive architecture. To make the probe record goal completion rule events, we add a clause specifying the probe is true to the rule concerning goal completion given above:

(Goalcompletion probes t) $\wedge$ goalachieved t $\wedge$ NEXT flag actions finished t

Each other rule in the architecture has a clause added asserting the probe is false at the time it fires. For example the `REACT` rule becomes:

`(Goalcompletion probes t = F)` $\wedge$ `stimulus t` $\wedge$ `NEXT flag as action t`

A similar clause is also added to the part of the architecture that describes the behaviour when no rule is firing (where no actions relevant to the interaction occur).

4 Verifying a Post-completion Error Design Rule

Erroneous actions are the immediate, obvious cause of failure attributed to human error, as it was a particular action (or inaction) that caused the problem: users pressing a button at the wrong time, for example. However, to understand the problem, and so minimize re-occurrence, approaches that consider the immediate causes alone are insufficient. It is important to consider *why* the person took that action. The ultimate causes can have many sources. Here we consider situations where the ultimate causes of an error are that limitations of human cognition have not been addressed in the design. An example might be that the person pressed the button at that moment because their knowledge of the task suggested it would be sensible. Hollnagel [19] distinguishes between human error `phenotypes` (classes of erroneous actions) and `genotypes` (the underlying psychological cause). He identifies a range of simple phenotypes such as repetition of an action, reversing the order of actions, omission of actions, etc. These are single deviations from required behaviour.

In practical designs it is generally infeasible to make all erroneous actions impossible. Fields [15] uses model-checking to identify errors by introducing phenotypes explicitly into task specifications. A problem with this approach is that it gives many false negatives: few tasks are possible if such errors are arbitrarily

made. The verifier must determine which are real problems. A definition of what is cognitively plausible is one way to make this judgment. A more appropriate aim is therefore to ensure that *cognitively plausible* erroneous actions are not made. To ensure this, it is necessary to consider the genotypes of the possible erroneous actions. We previously showed how a wide range of error phenotypes emerge from our model of cognitively plausible behaviour [13]. We do not claim to cover *all* cognitively plausible phenotypical actions. There are other ways each could occur for reasons we do not consider. However, not all errors that result from the model were explicitly considered when the principles were defined. The scope of the model in terms of erroneous actions is wider than those it was originally expected to encompass.

4.1 Formalising Post-completion Error Occurrence

In this paper, to demonstrate the feasibility of formally reasoning about design rules based on cognitively plausible behaviour, we consider one particular error genotype: the class of errors known as post-completion errors. As outlined above, a post-completion error occurs if the user terminates the interaction for a particular reason – that they have achieved their goal – despite other requirements on them. The same effect (i.e. phenotype) can occur for other reasons (such as malicious behaviour). However that would be considered a different class of error (genotype). Other design rules to prevent it might be required.

In our cognitive architecture this behaviour is modelled by the goal termination rule firing. Our probe signal `Goalcompletion` is used to record the fact that that particular rule has fired or not at any given time. Note that this rule can fire when the goal is achieved but does not have to. Note also that it firing is necessary but not sufficient for the cognitive architecture to make a post completion error. In some situations it is perfectly correct for the rule to fire. In particular if the interaction invariant has been re-established at the point when it fires then an error has not occurred. Thus whilst the error occurring is a direct consequence of the existence of this rule in the model, the rule is not directly modelling erroneous actions, just cognitively plausible behaviour that can lead to an erroneous action in some situations.

We first formalise what it means for a post-completion error to occur. Definition `PCE_OCCURS` specifies that a post-completion error occurs if there exists a time, `t`, less than the end time of the interaction `te`, such that the `Goalcompletion` probe is true at that time but the invariant has not been re-established.

PCE_OCCURS probes invariant te =
  ($\exists$t. t $\leq$ te $\wedge$ Goalcompletion probes t $\wedge$ $\neg$(invariant t))

This takes two higher order arguments, representing the collection of probes indicating which rules fire and the relation indicating when the interaction invariant is established. A final argument indicates the end time of interest. It bounds the interaction under consideration corresponding for example to the

point when the user has left and the machine has reset. The start time of the interaction is assumed to be time zero.

4.2 Formalising the Design Rule

We next formalise a well-known user-centric design rule intended to prevent designs giving a user the opportunity to make a post-completion error. It is based on the observation that the error occurs because it is possible for the goal to be achieved before the task as a whole has been completed. If the interaction design is altered so that all user actions have been achieved before the goal then a post-completion error will not be possible. In particular any tidying up actions associated with restoring the interaction invariant must be either done by the user before the goal can possibly be achieved, or done automatically by the system. This is the design approach taken by the designers of British cash machines where, unlike in the original versions, cards are always returned before cash is dispensed. This prevents the post-completion error where the person takes the cash (achieving their goal) but departs without the card (a tidying task).

We first specify that task completion at a time, `t`, is not simply goal completion but that the invariant is re-established.

TASK_DONE goalachieved invariant t = (goalachieved t $\wedge$ invariant t)

The higher-order variables `goalachieved` and `invariant` are the same as appear in the cognitive architecture.

The formal version of the design rule then states that for all times less than the end time, `te` it is not the case that both the goal is achieved at that time and the task is not done.

PCE_DR goalachieved invariant te =
($\forall$t. t $\leq$ te $\supset$ $\neg$(goalachieved t $\wedge$ $\neg$(TASK_DONE goalachieved invariant t)))

Thus when following this design approach, the designer must ensure that at all times prior to the end of the interaction it is not the case that the goal is achieved when the task as a whole is incomplete. Of course, there are other ways of avoiding the problem corresponding to a range of different design rules that we do not consider here.

The design rule was formulated in the above way to match a natural way to think about the design rule informally according to the above observation. It is trivial to prove within HOL that this is equivalent to a simpler form more convenient for reasoning, in terms of implication.

4.3 Justifying the Design Rule

We now prove a theorem that justifies the correctness of this design rule (up to assumptions in the cognitive architecture). If the design rule works, at least for users obeying the principles of cognition, then the cognitive architecture's

behaviour when interacting with a machine satisfying the design rule should never lead to a post-completion error occurring. We have proved using HOL the following theorem stating this:

$\vdash$ USER ... goalachieved invariant probes ustate mstate $\wedge$
PCE_DR (goalachieved ustate) (invariant ustate) te $\supset$
$\neg$(PCE_OCCURS probes (invariant ustate) te)

We have simplified, for the purposes of presentation the list of arguments to the relation `USER` which is the specification of the cognitive architecture, omitting those arguments that are not directly relevant to the discussion. One way to interpret this theorem is as a traditional correctness specification against a requirement. The requirement (conclusion of the theorem) is that a post-completion error does not occur. The conjunction of the user and design rule is a system implementation. The system is implemented by placing an operator (as specified by the cognitive architecture `USER`) with the machine (as minimally specified by the design rule). This theorem does not assert that real users cannot make the error, just that ones obeying the principles of cognition as modelled will not do so.

The proof of the above theorem is relatively simple. It involves case splits on the goal being achieved and the invariant being established. The only case that does not follow immediately is when the goal is not achieved and the invariant does not hold. However, this is inconsistent with the goal completion rule having fired so still follows fairly easily.

The definitions and theorem proved are generic. They do not specify any particular interaction or even task. A general, task independent design rule has thus been verified.

4.4 Machine-centric rules

The above design rule is in terms of user concerns – an invariant of the form suitable for the cognitive model and a user centric goal. Machine designers are not directly concerned with the user and this design rule is not in a form that is directly of use. The designer cannot manipulate the user directly, only machine events. Thus whilst the above rule and theorem are in a form of convenience to a usability specialist, they are less convenient to a machine designer. If we wish to integrate the two we need a more machine-centric design rule as below.

MACHINE_PCE_DR goalevent minvariant te =
($\forall$t. goalevent t $\supset$ ($\forall$t1. t $\leq$ t1 $\wedge$ t1 $\leq$ te $\supset$ minvariant t1))

This design rule is similar to the user centric version, but differs in several key ways. Firstly, the arguments no longer represent user based predicates. The goal event signal represents a machine event. Furthermore this is potentially an instantaneous event, rather than a predicate that holds from that point on. Similarly, the machine invariant concerns machine events rather than user events.

Thus, for example with a vending machine, the goal as specified in a user-centric way is that the user has chocolate. Once this first becomes true it will continue to hold until the end of the interaction, since for the purposes of analysis we assume that the user does not give up the chocolate again until after the interaction is over. The machine event however, is that the machine fires a signal that releases chocolate. This is a relation on the machine state rather than on the user state: `GiveChoc mstate`. It is also an event that occurs at a single time instance (up to the granularity of the time abstraction modelled). The machine invariant is also similar to the user one but specifying that the value of the *machine's* possessions are the same as at the start of the interaction – it having exchanged chocolate for an equal amount of money. It is also a relation on the machine's state rather than on the user's state.

The ramification of the goal now being an instantaneous event is that we need to assert more than that the invariant holds whenever the goal achieved event holds. The invariant must hold from that point up to the end of the interaction. That is the reason a new universally quantified variable `t1` appears in the definition, constrained between the time the goal event occurs and the end of the interaction.

We prove that this new design rule implies the original, provided assumptions are met about the relationship between the two forms of goal statements and invariants. It is these assumptions that form the basis of the integration between the user and machine-centric worlds.

$$\begin{array}{l}\vdash (\forall t.\ \text{minvariant t} \supset \text{invariant t}) \wedge \\ \quad (\forall t.\ (\text{goalachieved t}) \supset \exists t2.\ t2 \leq t \wedge (\text{goalevent t2})) \supset \\ \qquad \text{MACHINE\_PCE\_DR goalevent minvariant te} \supset \\ \qquad\quad \text{PCE\_DR goalachieved invariant te}\end{array}$$

This asserts that the machine based design rule `MACHINE_PCE_DR` does indeed imply the user centric one `PCE_DR`, under two assumptions. The first assumption is that at all times the machine invariant being true implies that the user invariant is true at that time. The second assumption asserts a connection between the two forms of goal statement. If the user has achieved their goal at some time `t` then there must have existed an earlier time `t2` at which the machine goal event occurred. The user cannot achieve the goal without the machine enabling it.

4.5 Integrating the Theorems

At this point we have proved two theorems. Firstly we have proved that a machine centric statement of a design rule implies a user centric one, and secondly that the user-centric design rule implies that post-completion errors are not made by the cognitive architecture. These two theorems can be combined giving us a theorem that justifies the correctness of the machine-centric design rule with respect to the occurrence of post-completion errors as illustrated in Figure 1. The theorem proved in HOL is:

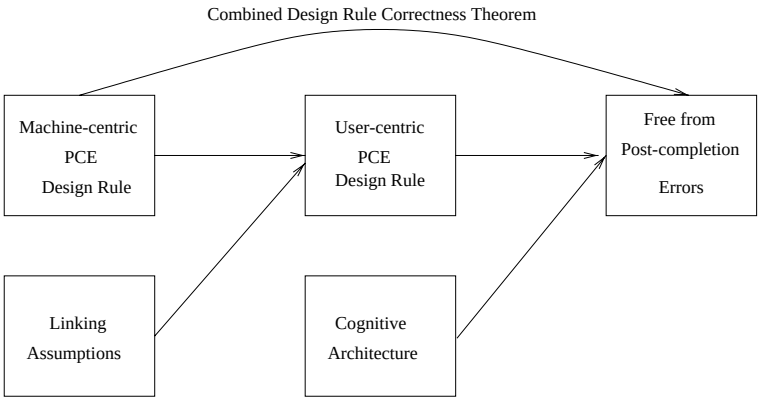

Fig. 1. Verifying the Design Rule in Stages

$$
\begin{array}{l}
\vdash\ (\forall t.\ \text{minvariant}\ t \supset \text{invariant}\ t) \wedge \\
\quad (\forall t.\ (\text{goalachieved}\ t) \supset \exists t2.\ t2 \leq t \wedge (\text{goalevent}\ t2)) \supset \\
\qquad \text{MACHINE\_PCE\_DR goalevent minvariant te} \wedge \\
\qquad \text{USER} \ldots \text{goalachieved invariant probes ustate mstate} \supset \\
\qquad\quad \neg(\text{PCE\_OCCURS probes (invariant ustate) te})
\end{array}
$$

This is a generic correctness theorem, that is independent of the task or any particular machine. It states that under the assumptions that link the machine invariant to the user interaction invariant and the user goal to the machine goal action, the machine specific design rule is "correct". By correct in this context we mean that if any device whose behaviour satisfies the device specification is used as part of an interactive system with a user behaving according to the principles of cognition as formalised, then no post-completion errors will be made. This is despite the fact that the principles of cognition themselves do not exclude the possibility of post-completion errors.

5 Discussion: Integration with full system verification

Our aim has been to verify a usability design rule in a way that integrates with formal hardware verification. The verification of the design rule needs to consider user behaviour. However, hardware designers and verifiers do not want to be concerned with cognitive models. Our aim has been therefore to separate these distinct interests so that they can be dealt with independently, but within a common integrated framework. In this section we discuss in general terms how the verified design rule could be combined with other verification results.

There are several ways the design rule correctness theorem could be used. The most lightweight is to treat the verification of the design rule as a justification of its use in a variety of situations with no further formal reasoning, just informal argument that any particular device design does match the design rule

as specified. Its formal statement then would give a precise statement, including assumptions in the theorem, of what was meant by the design rule. Slightly more formally, the formal statement of the design rule could be instantiated with the details of a particular device. This would give a precise statement about that device. The instantiated design rule correctness theorem then is a specific statement about the absence of user error (assuming cognitively plausible behaviour is followed).

A more heavyweight use of the theorem would be to formally verify that the device specification of interest implies the instantiated design rule. This theorem and its proof would concern only the device specification precisely because of the use of a machine-centric version of the design rule. It would be independent of consideration of the user model or user state. Such a theorem could be proved using machine-assisted proof. However, as it is a simple temporal property of the machine specification it could also be verified using automated model checking technology. Combined with the design rule correctness statement, this result gives a formal result not just that the specification meets the design rule but that in interacting with it a user would not make post-completion errors. If the proof that the device satisfied the design rule was done in HOL, then a HOL theorem would be obtained stating this.

As the verification framework we have used was originally developed for hardware verification, it would then be simple to combine this result with a formal hardware verification result stating that the implementation of the device implied its behavioural specification. This would give a theorem stating that the actual *implementation* of the device as specified would not lead to post-completion errors occurring. The hardware verification is again done independently of the cognitive model and explicit usability concerns. In previous work [10] [27] we demonstrated how such a hardware verification correctness theorem could be similarly chained with a full usability task completion correctness theorem stating that when the cognitive model was placed with the behavioural specification of the device, the combined behaviour of the resulting system was such that the task was guaranteed to be completed. The difference here is that the end usability statement being chained to is about the absence of a class of errors rather than task completion; however, the general approach is the same. We have demonstrated how this could be done both in a pure HOL theorem proving system [10] and in a hybrid system combining HOL with the decision diagram based hardware verification system, MDG [27].

The design rule correctness theorem could thus be combined with a result that a particular device specification meets the design rule. By further combining it with a result that a particular implementation of the device meets the specification we can obtain a theorem that the *implementation* does not result in post-completion errors occurring as is illustrated in Figure 2. This is under the assumption that the user behaves in a cognitively plausible way as specified.

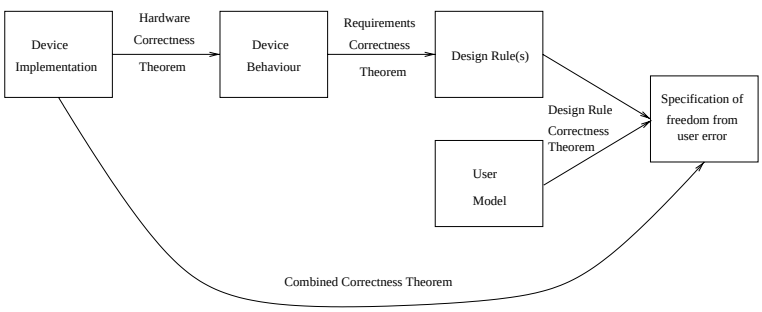

Fig. 2. Combining separate system correctness statements

6 Conclusions

We have shown how a usability design rule can be verified using machine-assisted proof. We started by outlining a set of principles of cognition specifying cognitively plausible behaviour. These principles are based on results from the cognitive science and human-computer interaction literature. From these principles we developed a formal cognitive architecture. This architecture does not directly model erroneous behaviour. However, erroneous behaviour can emerge if placed in an environment (ie with a computer system) that allows it.

We then formally specified a class of errors known as post-completion errors. We also specified two versions of a design rule claimed to prevent post-completion errors from being made. The first is specified in terms of user goals and invariant. The second is in terms of machine events, and so of more direct use to a designer. We proved a theorem that the user centred design rule is sufficient to prevent the cognitive architecture from committing post-completion errors. This theorem is used to derive a theorem that the machine-based formulation is also sufficient. The resulting theorem is a correctness theorem justifying the design rule. It is not an absolute correctness theorem, however. Rather, it says that users behaving according to the principles of cognition will not make post-completion errors when interacting with a device that satisfies the design rule.

The definitions and theorems are all generic and do not commit to any specific task or machine. They are thus a justification of the design rule in general rather than in any specific case. They can be instantiated, if required, to obtain theorems about specific scenarios and then further with specific computer systems.

This work demonstrates an approach that integrates machine-centred verification with user-centred verification. We show how by starting from user centred descriptions of a design rule and its correctness theorem we can ultimately gain a theorem that is concerned with machine events. The final generic theorem obtained has as its conclusion a user-centred statement about a class of human error. The theorem however is about the machine-centred design rule. The formal cognitive architecture forms the core of this theorem.

Furthermore, the higher-order logic framework adopted is that developed for hardware verification. Specifications, whether of implementations, behaviours or design rules, are higher-order logic relations over signals specifying input or output traces. A consequence of this is that the theorems developed integrate directly with hardware verification proofs about the computer component of the system. The theorem developed here, once instantiated for a particular device could be combined with correctness theorems about a specific device to obtain a theorem stating that the machine implementation implied that no post-completion errors could occur. This would require the proof of a linking theorem that the device specification satisfied the machine-centric design rule.

Finally the work presented here builds on the framework of our previous work on fully formal proofs that an interactive system completes the task [11]. A problem with that approach is that with complex systems, even with the simple set of principles of cognition used here, guarantees of task completion may be unobtainable with trade-offs needing to be made. The current approach, in addition to not requiring detailed proof within the design cycle, means that the most important errors for a given application can be focussed on.

Our formal model of user behaviour has very precise semantics that are open to inspection. Of course our reasoning is about what the cognitive architecture might do rather than about any real person. As such, the results should be treated with care. However, errors that the cognitive architecture could make are cognitively plausible and so worth attention.

7 Further Work

We have only considered a single class of error and one simple design rule that prevents it occurring. In doing so we have shown the feasibility of the approach. There are very many other classes of errors. Others that are potential consequences of the principles of cognition are discussed in [13]. There we gave informal arguments that a variety of design rules could prevent them. In future work we will formally model those error classes and design rules, and verify them formally following the approach developed here. This will also allow us to reason about the scope of different design rules especially those that apparently contradict.

In this paper we have focussed on the development of the generic theorems. We have been concerned with the verification of design rules in general, rather than in specific cases. We have argued, however that, since the framework used is that originally developed for hardware verification, integration of instantiated versions of the design rule correctness theorem is straightforward. It requires simple implication chaining of the separate correctness theorems. Major case studies are needed to demonstrate the utility of this approach.

Our model is intended to demonstrate the principles of the approach and covers only a small subset of cognitively plausible behaviour. As we develop it, it will give a more accurate description of what is cognitively plausible. We intend to extend it in a variety of ways. As this is done, more erroneous behaviour

will be possible. We have essentially made predictions about the effects of following design rules. In broad scope these are well known and based on usability experiments. However, one of our arguments is that more detailed predictions can be made about the scope of the design rules, relating them back to concepts such as communication goals. The predictions resulting from the model could be used as the basis for designing further experiments to validate the model and the correctness theorems proved, or further refine it. We have also suggested there are tasks where it might be very difficult or even impossible to produce a design that satisfies all the underlying principles, so that some may need to be sacrificed in particular situations. We intend to explore this issue further.

References

1. R. Back, A. Mikhajlova, and J. von Wright. Modeling component environments and interactive programs using iterative choice. Technical Report 200, Turku Centre for Computer Science, sep 1998.
2. A. Blandford, R. Butterworth, and P. Curzon. Puma footprints: linking theory and craftskill in usability evaluation. In *Proceedings of Interact*, pages 577–584, 2001.
3. A. E. Blandford and R.M. Young. The role of communication goals in interaction. In *Adjunct Proceedings of HCI'98*, pages 14–15, 1998.
4. A.E. Blandford, P.J. Barnard, and M.D. Harrison. Using interaction framework to guide the design of interactive systems. *International Journal of Human Computer Studies*, 43:101–130, 1995.
5. P. Bumbulis, P.S.C. Alencar, D.D. Cowen, and C.J.P. Lucena. Validating properties of component-based graphical user interfaces. In F. Bodart and J. van der Donckt, editors, *Proc. Design, Specification and Verification of Interactive Systems '96*, pages 347–365. Springer, 1996.
6. R. Butterworth, A.E. Blandford, and D. Duke. Using formal models to explore display based usability issues. *Journal of Visual Languages and Computing*, 10:455–479, 1999.
7. R. Butterworth, A.E. Blandford, and D. Duke. Demonstrating the cognitive plausibility of interactive systems. *Formal Aspects of Computing*, 12:237–259, 2000.
8. M. Byrne and S. Bovair. A working memory model of a common procedural error. *Cognitive Science*, 21(1):31–61, 1997.
9. J.C. Campos and M.D. Harrison. Formally verifying interactive systems: a review. In M.D. Harrison and J.C. Torres, editors, *Design, Specification and Verification of Interactive Systems '97*, pages 109–124. Wien : Springer, 1997.
10. P. Curzon and A.E. Blandford. Using a verification system to reason about post-completion errors. Presented at Design, Specification and Verification of Interactive Systems 2000. Available from http://www.cs.mdx.ac.uk/puma/ as working paper WP31.
11. P. Curzon and A.E. Blandford. Detecting multiple classes of user errors. In Reed Little and Laurence Nigay, editors, *Proceedings of the 8th IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'01)*, Lecture Notes in Computer Science. Springer-Verlag, 2001.
12. P. Curzon and A.E. Blandford. A user model for avoiding design induced errors in soft-key interactive systems. In R.J. Bolton and P.B. Jackson, editors,

TPHOLS 2001 Supplementary Proceedings, number ED-INF-RR-0046 in Informatics Research Report, pages 33–48, 2001.

13. P. Curzon and A.E. Blandford. From a formal user model to design rules. In P. Forbrig, B. Urban, J. Vanderdonckt, and Q. Limbourg, editors, *Interactive Systems. Design, Specification and Verification, 9th International Workshop*, volume 2545 of *Lecture Notes in Computer Science*, pages 19–33. Springer, 2002.
14. D.J. Duke, P.J. Barnard, D.A. Duce, and J. May. Syndetic modelling. *Human-Computer Interaction*, 13(4):337–394, 1998.
15. R.E. Fields. Analysis of erroneous actions in the design of critical systems. Technical Report YCST 20001/09, University of York, Department of COmputer Science, 2001. D.Phil Thesis.
16. M.J.C. Gordon and T.F. Melham, editors. *Introduction to HOL: a theorem proving environment for higher order logic*. Cambridge University Press, 1993.
17. W. Gray, R.M. Young, and S. Kirschenbaum. Introduction to this special issue on cognitive architectures and human-computer interaction. *Human-Computer Interaction*, 12:301–309, 1997.
18. W.D. Gray. The nature and processing of errors in interactive behavior. *Cognitive Science*, 24(2):205–248, 2000.
19. E. Hollnagel. *Cognitive Reliability and Error Analysis Method*. Elsevier, 1998.
20. D.E. Kieras, S.D. Wood S.D., and D.E. Meyer. Predictive engineering models based on the EPIC architecture for a multimodal high-performance human-computer interaction task. *ACM Trans. Computer-Human Interaction*, 4(3):230–275, 1997.
21. D. Leadbetter, P. Lindsey, A. Hussey, A. Neal, and M. Humphreys. Towards model based prediction of human error rates in interactive systems. In *Australian Comp. Sci. Communications: Australasian User Interface Conf.*, volume 23(5), pages 42–49, 2001.
22. T.G. Moher and V. Dirda. Revising mental models to accommodate expectation failures in human-computer dialogues. In *Design, Specification and Verification of Interactive Systems '95*, pages 76–92. Wien : Springer, 1995.
23. F. Paterno' and M. Mezzanotte. Formal analysis of user and system interactions in the CERD case study. In *Proceedings of EHCI'95: IFIP Working Conference on Engineering for Human-Computer Interaction*, pages 213–226. Chapman and Hall Publisher, 1995.
24. F.E. Ritter and R.M. Young. Embodied models as simulated users: introduction to this special issue on using cognitive models to improve interface design. *Int. J. Human-Computer Studies*, 55:1–14, 2001.
25. C. R. Roast. Modelling unwarranted commitment in information artifacts. In S. Chatty and P. Dewan, editors, *Engineering for Human-Computer Interaction*, pages 77–90. Kluwer Academic Press, 1998.
26. J. Rushby. Using model checking to help discover mode confusions and other automation suprises. In *3rd Workshop on Human Error, Safety and System Development (HESSD'99)*, 1999.
27. H. Xiong, P. Curzon, S. Tahar, and A. Blandford. Formally linking MDG and HOL based on a verified MDG system. In M. Butler, L. Petre, and K. Sere, editors, *Proc. of the 3rd International Conference on Integrated Formal Methods*, volume 2335 of *Lecture Notes in Computer Science*, pages 205–224, 2002.