Detecting Multiple Classes of User Errors

Paul Curzon and Ann Blandford

Interaction Design Centre, Middlesex University, London, UK
{p.curzon,a.blandford}@mdx.ac.uk

Abstract. Systematic user errors commonly occur in the use of interactive systems. We describe a formal *reusable* user model implemented in higher-order logic that can be used for machine-assisted reasoning about user errors. The core of this model is a series of non-deterministic guarded temporal rules. We consider how this approach allows errors of various specific kinds to be detected by proving a single theorem about a device. We illustrate the approach using a simple case study.

1 Introduction

In this paper, we present an approach to the verification of interactive systems that allows the detection of systematic user errors. The approach extends standard hardware verification techniques based on machine-assisted proof to this new domain. Human error in interactive systems can be just as disastrous as device errors. Whilst it is impossible to eradicate all human error without trivializing system functionality, there are whole classes of persistent user errors whose presence is predictable due to their distinct cognitive cause [11]. Their possibility can be removed completely with appropriate design [8, 2]. If methods for detecting such errors are available system reliability can be improved. It is therefore important that any verification approach used considers a range of errors in a systematic way. Designing devices so that a single class of error is absent is relatively straight forward. However, there are many different reasons for users making mistakes. Design principles used to avoid such errors can conflict, so that without care, in eliminating one class of error, other errors are introduced.

People do not normally behave randomly. Neither do they behave completely logically. However, it is a reasonable, and useful, approximation to say that they behave *rationally*. They enter an interaction with goals and some knowledge of the task they wish to perform. They act in ways that, given their knowledge, seem likely to help them achieve their goals. It is precisely because users are behaving rationally in this way that they make certain kinds of persistent errors with certain device designs. For example, with the early designs of cash machines, users would frequently forget to take back their card. Most current cash machines have been redesigned so that this no longer occurs.

We investigate a method based on a generic user model specified in higher-order logic. This contrasts, for example, with an approach based on formulating properties corresponding to user errors and checking that those properties do not hold of the system. In our approach *rational* user behavior is specified within a *reusable*, generic user model. This model is based on theory from the cognitive sciences and requires validating only once, not for each new device considered. The user model is treated as a component of the system under verification. A single theorem is proved using the HOL proof system [7] that the task under consideration is guaranteed to be completed by this combined system. By taking this approach, rather than considering what could go wrong with each system we concentrate on what the desired outcome of interacting with the system is. The occurrence of systematic user errors is a side effect of the user behaving rationally. Our reasoning is based directly on the underlying behavior that causes the problems to arise. We then check a single positive property (that the task is completed) not a whole range of negative properties (that various situations do not arise). We are concerned here with the detection of user errors. It is not our aim that our user model explain other aspects of behaviour.

We build here on our previous work. In [4] we demonstrated how our approach of a generic user model combined with formal proof could be used to detect the possibility of user errors. We also demonstrated how we could prove their absence. We used a very simple user model, however. It took user goals into account in only a limited way. It could therefore only detect one class of error: the post completion error [2]. This is the situation where once the goal has been achieved completion tasks are forgotten. This work did however show the feasibility of our approach. In [5] we introduced a more accurate generic user model that took user knowledge into account and allowed for a wider range of rational behaviour. We demonstrated how, by proving a different theorem, this more accurate model can be used to detect in isolation a second class of user errors. This class of error is related to communication goals and includes order errors and errors where the device design assumes the user knows they must perform a device specific (as opposed to task specific) action.

In this paper we extend this work by proving a single theorem that detects the presence or absence of both classes of error previously considered together with a third class of user error related to device delay. This ability to simultaneously detect multiple classes of errors is important because design guidelines to avoid such errors can contradict. For example post-completion errors can be eliminated if the order of user actions is carefully dictated by the device. However, devices dictating the order of user actions is precisely what causes the second class of errors we examine.

Formal user modeling is not new. Butterworth *et al* [1] use TLA to describe behavior at an abstract level that does not support re-use of the user model between devices. While it can support reasoning about errors, each model has to be individually hand-crafted. Moher and Dirda [9] use Petri net modeling 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 [10] use LOTOS and ACTL to specify intended user behaviors and hence reason about interactive behavior. Because their user model describes how the user is intended to behave, rather than how users might actually behave, it does not support reasoning about errors. Duke *et al* [6] 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. Our work complements these alternative uses of formal user modelling. It also complements traditional hardware verification approaches where a device implementation is verified to meet a machine-centered specification, or where liveness and safety properties of a device are checked. Such approaches are concerned with the detection of device errors, rather than user errors as here. This is discussed further in [4].

2 The HOL Theorem Prover

The work described here uses the HOL system [7]. It is a general purpose, interactive theorem prover that has been used for a wide variety of applications. A typical proof will proceed by the verifier proving a series of intermediate lemmas, that ultimately can be combined to give the desired theorem. Proofs are written in the meta-language of the theorem prover – Standard ML. Each proof step is a call to an ML function. The proof script is developed by calling such functions interactively. The resulting ML proof script can later be rerun in batch mode to subsequently generate the theorems proved. If modifications are made to the system under verification then much of the proof script is likely to be reusable to verify the new design.

The HOL system provides a wide range of definition and proof tools, such as simplifiers, rewrite engines and decision procedures, as well as lower level tools for performing more precise proof steps. The architecture of the system means that new, trustable proof tools for specific applications can easily be built on top of the core system. Such proof tools are just Standard ML functions that call existing proof functions.

All specifications, goals and theorems in HOL are written in higher-order logic. Higher order logic treats functions as first class objects. Specifications are thus similar to functional programs with logical connectives and quantification. The notation used in this paper is summarized in Table 1.

Our work is based on the specification of a *generic* user model. The use of higher-order logic allows us to do this in an elegant way. As we will see the generic user model is a higher-order relation that takes various functions as arguments. Instantiation of the user model just involves providing concrete arguments to this relation.

3 A Generic User Model and Task Completion Theorem

Our user model is based on a series of non-deterministic (disjunctive) temporally guarded action rules. Each describes an action that a user *could* rationally make. The rules are grouped corresponding to the user performing actions for specific

$a \wedge b$	both a and b are true
$a \vee b$	either a is true or b is true
$a \supset b$	a is true implies b is true
$\forall n.\ P(n)$	for all n, property P is true of n
$\exists n.\ P(n)$	there exists an n for which property P is true of n
f n	the result of applying function f to argument n
$a = b$	a equals b
if a then b else c	if a is true then b is true, otherwise c
$\vdash P$	P is a definition or theorem

Table 1. Higher-order Logic notation

related reasons. Each such group then has a single generic description. Random behavior is explicitly excluded. Thus the model can only be used to detect errors that occur as a result of users acting rationally. If such errors are possible they are liable to occur persistently, if not predictably. Their eradication will thus improve the reliability and usability of the system greatly. The model does not describe what a user *does* do, just what a user *could* do rationally. Our model makes no attempt to describe the likelihood of particular actions: a user error is either possible or not. Since we consider classes of error that can, by appropriate design, be completely eliminated, this strict requirement is in our view appropriate.

Full details of the model are given in [5]. Here, for clarity of explanation we use a semi-formal higher-order logic notation to give an overview.

3.1 Reactive behavior

The first group of non-deterministic rules we consider is that of *reactive* behavior, where a user reacts to a light (for example) coming on, that clearly indicates that a particular action should be taken. For example, if a light comes on next to a button, a user might, if the light is noticed, react by pressing the button. All the rules have the basic form below. If at a time `t` some condition, here `light`, is true then the `NEXT` action taken by the user, at an unspecified later time, out of the list of possible actions may be the given action, here pressing `button`.

```
(light t) ∧ NEXT user_actions button t
```

Note that the relation NEXT does *not* require that the action is taken on the *next cycle*, but rather that it is taken before any other user action. A relation is recursively defined that, given a list of such pairs of inputs and outputs, asserts the above rule about them, combining them using disjunction, so that they are non-deterministic choices. To target the generic user model to a particular device, it is applied to a concrete version of this list containing specific signals:

$[(\mathtt{light}_1,\ \mathtt{button}_1);\ \ldots\ (\mathtt{light}_n,\ \mathtt{button}_n)]$

People do not interact with devices purely in a reactive way, however. They may ignore reactive stimuli for very rational reasons. In subsequent sections we show how the model is extended to take into account some such rational behaviour.

3.2 Communication Goals

People enter an interaction with some knowledge of the task that they wish to perform. In particular, they enter the interaction with communication goals: a task dependent mental list of information the user knows they must communicate to the device. For example, on approaching a vending machine, a person knows that they must communicate their selection to the machine. Similarly, they know they must provide money before they will obtain their selection. While inserting coins is not strictly a communication goal in cognitive science terms, for the purposes of this paper we treat it in the same way. Communication goals are important because a user may take an action as a result not of some stimulus from the machine but as a result of seeing an apparent opportunity to discharge a communication goal. For example, if on approaching a rail ticket machine the first button seen is that with the desired destination on, the person may press it, irrespective of any guidance the machine is giving. No fixed order can be assumed over how communication goals will be discharged if their discharge is apparently possible.

Communication goals can be modeled as guard-action pairs. The guard describes the situation under which the discharge of the communication goal can be attempted. The action is the action that discharges the communication goal. They form the guard and action of a temporally guarded rule. We include an additional guard to this rule, stating that the action will only be attempted if the user's main goal has not yet been achieved. Strictly a similar guard ought to be added to the reactive rules. Currently they describe purely reactive behavior.

```
∼(goalachieved t) ∧ (guard t) ∧ NEXT user_actions action t
```

As for reactive behavior a list of guard-action pairs is provided as an argument to the user model rather than the rules being written directly. The separate rules are combined by disjunction with each of the other non-deterministic rules.

As the user believes they have achieved a communication goal, it is removed from their mental list. This is modeled by a daemon within the user model. It monitors the actions taken by the user on each cycle, removing any from the list used for the subsequent cycle.

3.3 Completion

In achieving a goal subsidiary tasks are often generated. Examples of such tasks include replacing the petrol cap after filling a car with petrol, taking the card back from a cash machine and taking change from a vending machine [2]. 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*. The underlying reason why these tasks must be 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 fill a car with petrol the petrol cap must be removed, and later restored. A condition on the state that holds at the start of the interaction, must be restored by the end. We specify the need to perform these completion tasks indirectly by supplying this interaction invariant as a higher-order argument to the user model.

We assume that a user, on completing the task in this sense, will terminate the interaction, irrespective of any other possible actions. Rather than specifying it as a non-deterministic rule we model it using an if-then-else construct, so that it overrides all other actions. A special user action `finished` indicates that the user has terminated the interaction. If the interaction had been terminated previously then it remains terminated.

```
if   (((invariant t) ∧ (goalachieved t)) ∨ finished (t-1))
then  NEXT user_actions finished t
else  non-deterministic rules
```

Cognitive psychology studies, however, have shown that users also sometimes terminate interactions when only the goal itself has been achieved [2]. This can be modeled as an extra non-deterministic rule.

```
(goalachieved t) ∧ NEXT user_actions finished t
```

The model also allows a user to terminate an interaction when no rational action is available. We assume that the user aborts in this situation. This non-deterministic rule acts as a final default case in the user model. Its guard states that none of the other rules' guards are true.

The user model is described by a relation that combines these separate rules. It takes a series of arguments corresponding to the details relevant to a specific machine: the list of possible user actions, the list of communication goals for the task, the list of reactive stimuli and actions they might prompt, the relevant possessions, the goal of the user and the interaction invariant. By providing these specific details as arguments to the relation, a user model for the specific interaction under investigation is obtained automatically. The important point is that the underlying cognitive model does not have to be provided each time, just lists of relevant actions and so on. The way those actions are acted upon by the model is modeled only once.

3.4 Correctness Theorem

We now consider the theorem to be proved about a device. The usability property we are interested in is that if the user interacts rationally with the machine, based on their goals and knowledge, then they are guaranteed to complete the task for which they started the interaction. As noted earlier, task completion is more than just goal completion. In achieving the goal, other important sub-tasks may result that must then be done in addition to completing the goal. The property that we thus require to hold is that eventually both the goal has been achieved and all other sub-tasks have been completed. The latter can be formalized in terms of the interaction invariant as in the user model. It must be guaranteed that the goal will be achieved and the interaction invariant restored.

The user model and the device specification are both described by relations. The device relation is true of its inputs and output arguments if they describe consistent input-output sequences of the device. Similarly, the user model relation is true if the inputs (observations) and outputs (actions) are consistent sequences that a rational user could perform. The combined system can then be described as the conjunction of the instantiated user model and the specification of the system. The task completion theorem we wish to prove thus has the form:

```
⊢ ∀ state traces .
  initial state ∧
  device specification ∧
  user model
    ⊃ ∃t. (invariant t) ∧
          (goalachieved t)
```

If a theorem of this form can be proved then even if a user is capable of making the rational errors considered, then that potential will not affect the completion of the task: the errors will never manifest themselves.

The theorem is reusable in the same way as the user model. The same information must be provided: notably the users goal and the interaction invariant, together with the device specification and specialized user model. Thus the correctness theorem does not need to be completely reformulated for each verification.

4 User Errors Detected by the User Model

Though the user model is simple, it describes a user who is capable of making a range of persistent but rational errors. The model does not imply that mistakes will always be made, just that the potential is there. The errors are consequences of describing the way results from cognitive science suggest people act in trying to achieve their goals. The errors are detected by attempts to prove the task completion theorem. If a device design is such that users can make errors then it will be impossible to prove that the task can be completed in all situations. It should be noted that we define classes of errors not by their effects but by their cognitive cause. We do not claim that in proving the absence of a particular error that a similar effect might not happen due to some other cause such as a fire alarm ringing in the middle of an interaction.

4.1 Post-completion errors

One kind of common, persistent user error that emerges from the user model is the post-completion error [2]. This is the situation where a user terminates an interaction with outstanding completion tasks remaining. For example, with old cash machines users persistently, though unpredictably, took cash but left their card. Even in laboratory conditions people have been found to make such errors [2]. This behavior emerges as a consequence of there being a rule in the model allowing a user to stop once the goal has been achieved. If a system is to be designed so that such errors do not manifest themselves, then the goal must not be achievable until after the invariant has been restored. If this is so, the rule will only become active in the safe situation when the task is fully completed. Note that such errors could still occur (less frequently) if the system is designed so that the goal is achieved first but that warning messages are printed or beeps sounded to remind the user to do the completion tasks. Such designs do not remove all possibility of the error being made, they just reduce its probability. In our framework such a device is still considered erroneous.

4.2 Communication-goal errors

A further class of errors that can be detected are those based on communication goals. Where there is no task-related, rather than device-related, restriction on the order that communication goals must be discharged, different users may attempt to discharge them in different orders. This will occur even in the face of the device using messages to indicate the order it requires. As with post-completion errors this problem is persistent but occurs unpredictably. It can be avoided if the device does not require a specific order for communication goal actions. In the model this error is a consequence of the communication goal rules activating in any order provided their guard is active, and that if the action is done that communication goal is removed. This means that the user model may be left at a later time with no rational action to take. The abort rule is then activated and the user model terminates the interaction before the task is completed. Similarly if a design assumes device-specific knowledge of a task that is not a communication goal, without giving reactive stimuli, then the user model will abort: a user error occurs.

4.3 Device Delay Errors

The user model can also detect some errors related to device delay. If there is no feedback during delays users often repeat the last action, for example. In the user model if there is no light to react to and the user has no outstanding communication goals then only the abortion rule is active. If such a situation can occur, then the task completion theorem cannot be proved. If there are outstanding communication goals active, the model would force one of those actions to be scheduled. The action could be taken before the device is ready, thus having no effect. The communication goal would be removed from the communication goal list however, so would not necessarily be repeated. At a later point this would lead to only the abortion rule being possible. Again it would not be possible to prove task completion. The device would need to be redesigned so that the delay occurred when the user had no opportunity to discharge outstanding communication goals, and a "wait" message of some form displayed. This would be reactive in the sense that whilst displayed the user would react by doing nothing. Erroneous systems could still escape detection, however. In particular, if the light indicating the previous action remained lit during the computation time, then task completion could be proved though the reasoning would require the user repeating an action. The problem here is that our current user model is not sufficiently accurate. In particular, humans do not react to stimulus unless they believe that it will help them achieve their goal. In future work we will add additional guards to the reactive rules to model *rational reactive* behavior.

5 Case Study

To illustrate the use of the model we consider a vending machine. We use a simple example here to demonstrate the approach. However, any device that could be described in terms of a finite state machine and for which specific user actions and goals can be formulated could in principle be treated in a similar way. This includes not only walk-up-and-use machines such as cash machines but also, for example, safety critical systems such as Air Traffic Control Systems.

The vending machine we consider requires users to supply a pound coin and gives change. The user must insert the coin and make a selection (we simplify this here to the pressing of a single button). Processing time is needed once the money is inserted before change is released (without loss of generality we assume there is just a single cycle of processing). We assume for the sake of simplicity that the chocolate machine always contains chocolate. Despite its simplicity, without careful design, such a machine has the potential for users making communication goal errors - the specific order that the coin is inserted and the selection made is not forced by the task so a user could do them in any order. On the other hand if chocolate is given out before the change, a post-completion error could be made. Delay errors might also occur due to the processing time. Indeed, vending machines with such design problems are widespread. Here we consider a design that overcomes these problems.

Our design accepts coin and selection in any order. Once both have been completed it releases the change. A light flashing by the change slot indicates this. However this only occurs after the delay. A "wait" light indicates to the user that the machine is busy. Note that this processing is scheduled after all communication goals are fulfilled. A sensor on the change slot flap releases chocolate when the change is taken. A finite state machine description of the machine is given in Figure 1. We use a relational version of this finite state machine as the behavioral specification. Our approach is not restricted to finite state machine specifications however.

We must target the generic user model for the device in question. This involves supplying concrete values for each of the model's arguments. We must provide information about the device inputs and outputs, and the user's internal state. This involves defining tuple types with each field corresponding to

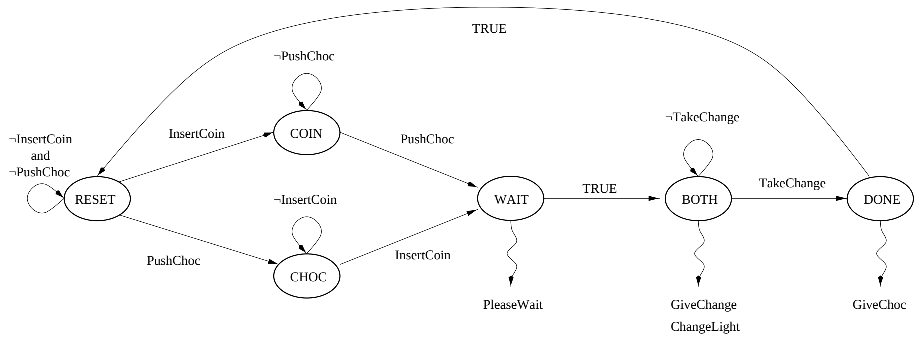

Fig. 1. Finite State Machine Specification of a Chocolate Machine

traces of inputs, outputs, states, etc. Accessor functions to the fields are then used to represent that event.

The first argument that must be supplied to the user model is a list of the actions a user could ever take that affect the interaction. This is used in the rules to specify that all the other actions do not occur when we specify that a particular event happens next. For our example the possible actions are represented by an `InsertCoin` field, a `PushChoc` field corresponding to the user pushing the selection button, a `UserFinished` field indicating the termination of the interaction, a `TakeChange` field and a `Pause` action which means the user is actively waiting:

```
[InsertCoin; PushChoc; UserFinished; TakeChange; Pause]
```

The second piece of information that must be supplied is the initial list of communication goals together with their guards. Here there are two communication goals specified. When a user approaches the machine they know they must insert a coin and that this can only be done if they possess the coin. They also know they must make a selection. There are no task enforced conditions on when this can occur, so its guard is TRUE. It can happen at any time. The communication goal list thus has the form:

```
[(HasCoin, InsertCoin); (TRUE, PushChoc)]
```

We must provide a list of reactive signals. This is a list pairing observations with actions that they prompt the user to make. In our design, there are two such signals: the `ChangeLight` prompts the user to take the change and so trigger the sensor on the change flap; the `PleaseWait` light prompts the user to wait (i.e intentionally do nothing).

```
[(ChangeLight, TakeChange); (PleaseWait, Pause)]
```

We must also indicate the possessions of the user. A relation `POSSESSIONS` in the generic user model gathers this information into an appropriate form. We supply it with the details of the user having chocolate and coins, the machine giving chocolate, the user inserting a coin and counts of the number of coins and chocolate bars possessed by the user.

Finally we must specify the goal of this interaction and the interaction invariant. Both are also used to create the concrete task completion theorem. The goal is to obtain chocolate. Its achievement is given by the field of the user state `UserHasChoc`. For vending machines the interaction invariant, `VALUE_INVARIANT`, can be based on the value of the user's possessions. After interacting with a vending machine the value of the user's possessions should be at least as great as it was at the start (time 1). The value of a users possessions is calculated from possession count and value fields using a relation `VALUE`.

```
VALUE_INVARIANT possessions state t =
    (VALUE possessions state t >= VALUE possessions state 1)
```

This relation will not hold throughout the interaction. When the coin is inserted the value will drop and will only return to its initial value once both chocolate and change are taken. It is only an invariant in the sense that it must be restored by the end of the interaction.

5.1 Proving the Task Completion Theorem

The task completion theorem we proved about this device has the form:

```
⊢∀state COINVAL CHANGEVAL CHOCVAL.
  (COINVAL = CHANGEVAL + CHOCVAL) ∧
  (DeviceState state 1 = RESET) ∧
  (UserHasCoin state 1) ∧
  ∼(UserHasChoc state 1) ∧
  MACHINE_USER state ∧
  MACHINE_SPEC state ⊃
    ∃t. (UserHasChoc state t) ∧
        (VALUE_INVARIANT
               (POSSESSIONS possessions CHOCVAL COINVAL CHANGEVAL)
               state t)
```

Here `MACHINE_SPEC` is the behavioral specification of the vending machine. The relation `MACHINE_USER` is the instantiated user model: notice that its only argument is the state. All the other details required such as communication goals have been provided in the instantiation. The theorem also contains assumptions about the initial device state and that the user has a coin but no chocolate at time 1.

The arguments to the predicate `POSSESSIONS` give signals representing the details of the counts and values of the user's possessions. An advantage of our approach is that proofs can be generic. The correctness theorem we proved is generic with respect to the value of coins, change and chocolate. They are represented in the `POSSESSIONS` predicate by variables `COINVAL`, `CHANGEVAL` and `CHOCVAL` rather than by fixed integers. The correctness theorem contains an assumption that gives the restriction that the values concerned must satisfy:

```
COINVAL = CHANGEVAL + CHOCVAL
```

The correctness theorem holds for any triple of values that satisfy this relation.

We have proved the task completion theorem using symbolic simulation by proof within the HOL theorem prover [7]. Our verification is fully machine-checked. An induction principle concerning the stability of a signal is used repeatedly to step the simulation over periods of inactivity between a rule activating and the action actually happening.

For example, the theorem below states that if the machine is in the `CHOC` state at some time `t1` greater than 0, then eventually state WAIT is entered. It requires that at time `t1` the user has a coin but no chocolate yet, that the interaction was not terminated on the previous cycle and that inserting a coin is still an undischarged communication goal. Once the new state is entered (at some time `t2`) the user will still not have terminated the interaction, the communication goal will have been discharged, the count of the user coins will have been decremented but the counts of chocolate and change will be unchanged.

```
⊢ 0 < t1 ∧
  (DeviceState state t1 = CHOC) ∧
  (UserHasCoin state t1) ∧
  ~(UserHasChoc state t1) ∧
  ~(UserFinished state (t1-1)) ∧
  (UserCommgoals state t1 = [(InsertCoin, UserHasCoin)]) ∧
  MACHINE_USER state ∧
  MACHINE_SPEC state
  ⊃
    (∃t2.
      t1 <= t2 ∧
      (DeviceState state t2 = WAIT) ∧
      ~(UserHasChoc state t2) ∧
      ~(UserFinished state (t2-1)) ∧
      (UserCommgoals state t2 = []) ∧
      (CountChoc state t2 = CountChoc state t1) ∧
      (CountCoin state t2 = CountCoin state t1 - 1) ∧
      (CountChange state t2 = CountChange state t1))
```

A similar theorem is proved about each finite state machine state. The final theorem is proved by combining these separate state theorems.

These theorems are currently proved semi-automatically. A series of lemmas must be proved for each. They are formulated by hand, but then proved automatically by a set of specially written proof procedures. As the lemmas are of a very standard form it should be straightforward to automate their formulation. Furthermore, as design changes are made, many of the lemmas proved are reusable.

6 Conclusions and Further Work

We have presented a verification approach that allows multiple classes of user errors to be detected or verified absent from designs by proving a single task completion theorem. The approach is based on the use of a generic user model. Rather than specify erroneous properties directly, rational behavior is specified. This means that errors that are the result of that rational behavior are detected. This is less restrictive than verifying that nothing bad can possibly happen, whatever the user's goals. An advantage of the approach over specifying properties directly is that the informal and potentially error-prone reasoning implicitly required to generate appropriate properties is formalized. To do this reasoning formally would need a formal user model. By using a generic user model directly, the cognitive basis of the errors is specified and validated once rather than for each new design or task. It also does not need to be revalidated when errors are found and the design subsequently modified.

To illustrate our approach, we described a small case study concerned with the design of vending machines. We considered the verification of a design free of the classes of user errors covered. If we attempted to verify the correctness of a faulty design, the correctness theorem would not be provable. For example, suppose the design released the chocolate first. We would not be able to prove from the user model that the change was taken. Instead, we would only be able to prove that either the change was taken or that the user finished. This is because for this design the completion rule becomes active before the task has been completed. The rule's guard is that the goal has been completed and this is achieved as soon as the user takes the chocolate. Thus rather than proving a step theorem such as that given, we would only be able to prove a conclusion that one of two situations arise, only one of which leads to full task completion. A case study discussing in more detail the attempted verification of a faulty design using our approach is given in [4].

The methodology shows promise for use on more complex examples. We intend to carry out more case studies to test its utility. In particular we are currently working on an Air Traffic Control case study. The main difficulty in verifying more complex systems is the time taken to develop the proof. This problem will be eased as we automate the proofs. We used interactive proof in the HOL system to prove the correctness theorem presented here. We intend to continue to develop tactics to increase the automation in doing this. Currently tactics have been written that automate the proofs of the main lemmas. Further work will automate the formulation of the lemmas and their combination. The lemmas and proofs are very formulaic so much of this task is likely to be straightforward. The use of a common user model makes such automation more tractable.

The current user model by no means covers all aspects of rational user behaviour. We will build on it improving its accuracy. In doing so we will increase the number of classes of error detectable. For example, the rules concerning reactive behaviour need to be made rational so that users only react when it appears to help them achieve their goals. There is also a delay between a person committing to an action and actually taking that action. This can be modelled by linking each external action with an additional internal "commit" action. This will allow user errors resulting from such delays to be detected.

We do not claim our methodology can prevent all user errors. However, by providing a mechanism for detecting a series of classes of systematic errors, the usability of systems in the sense of absence of user errors is improved.

Acknowledgements This work is funded by EPSRC grant GR/M45221. Matt Jones made useful comments about an early version of this paper.

References

1. R. Butterworth, A. Blandford, and D. Duke. Using formal models to explore display based usability issues. *J. of Visual Languages and Computing*, 10:455–479, 1999.
2. M. Byrne and S. Bovair. A working memory model of a common procedural error. *Cognitive Science*, 21(1):31–61, 1997.
3. F. Corella, Z. Zhou, X. Song, M. Langevin and E. Cerny. Multiway Decision Graphs for Automated Hardware Verification. *Formal Methods in System Design*, 10(1):7–46, 1997.
4. Paul Curzon and Ann 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/.
5. Paul Curzon and Ann Blandford. Reasoning about order errors in interaction. Supplementary Proceedings of the International Conference on Theorem Proving in Higher-order Logics, August 2000. Available from http://www.cs.mdx.ac.uk/puma/.
6. D.J. Duke, P.J. Barnard, D.A. Duce, and J. May. Syndetic modelling. *Human-Computer Interaction*, 13(4):337–394, 1998.
7. M.J.C. Gordon and T.F. Melham. Introduction to HOL: a theorem proving environment for higher order logic Cambridge University Press, 1993.
8. W-O Lee. The effects of skills development and feedback on action slips. In Monk, Diaper, and Harrison, editors, *People and Computers VII.* CUP, 1992.
9. 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*, pp 76–92. Wien : Springer, 1995.
10. F. Paterno' and M. Mezzanotte. Formal analysis of user and system interactions in the CERD case study. In *Proc. of EHCI'95: IFIP Working Conference on Engineering for Human-Computer Interaction*, pp 213–226. Chapman and Hall, 1995.
11. J. Reason. *Human Error.* Cambridge University Press, 1990.