Misleading Behaviour in Interactive Systems

Jeremy Gow
UCL Interaction Centre
University College London
31–32 Alfred Place
London, UK
`j.gow@ucl.ac.uk`

Harold Thimbleby
UCL Interaction Centre
University College London
31–32 Alfred Place
London, UK
`h.thimbleby@ucl.ac.uk`

Paul Cairns
UCL Interaction Centre
University College London
31–32 Alfred Place
London, UK
`p.cairns@ucl.ac.uk`

Abstract

We introduce the idea of *partial behaviours* in user interfaces. A partial behaviour can beguile users, and may be a cause of interaction problems — however, it is possible to identify and remove them early in the design process, making them a useful concept for interaction design. A characterisation of partial behaviours is presented, in terms of a matrix algebra model of interactive systems. We use the model to show some real interfaces have undesirable and apparently unnecessary partial behaviours, and we discuss how choices made when modelling affect our notion of partiality. We also briefly describe a design tool that provides automatic support for partial behaviour analysis.

Keywords

User interface design, modes, formal methods, matrix algebra.

1. Introduction

Partial behaviours can mislead users, whether they arise from bad design or from legitimate user interface modes. A simple example of a partial behaviour appears in how the delete key in Microsoft Word deletes the previous keystroke: almost always, ***key***$\boxed{\text{DEL}}$ results in nothing (i.e., the delete removes the effects of typing the last key), but there are a few contexts where this useful and obvious behavioural rule fails. For example, if automatic capitalisation of the word starting a sentence is enabled, typing "`.␣word.`$\boxed{\text{DEL}}$" (i.e., dot, space, word, dot, delete) will result in "`.␣Word`" rather than "`.␣word`" — the difference is the word is still capitalised, so $\boxed{\text{DEL}}$ only deleted some of what the previous keystroke did. Thus the rule "$\boxed{\text{DEL}}$ deletes the last character" is only partially correct, because sometimes $\boxed{\text{DEL}}$ does things differently.

The HCI interest in partial behaviour is that a user might learn a *partial* behaviour but think it is a *general* behaviour (particularly if they have never encountered a counter-example of their inferred rule). In general, it could lead to user frustration and error. In a safety critical context, this may be disastrous. Elsewhere, we have called some aspects of this problem an *interaction trap* [1], inconsistency. but the specific contributions of this paper are to show that:

- Partial behaviour is a useful and usable concept for user interface design which lends itself to early design questions, before empirical methodologies are applicable.
- We can formalise partial behaviour constructively and very generally, and in a way which allows automatic support for the user interface analyst.
- We can characterise the ways in which changing the underlying model affects analysis of partial behaviours.

2. Partiality

We can informally define partial behaviour of a user interface as one where a given user action has a consistent outcome most, but not all, of the time. In other words, there is some minor inconsistency in the outcome.

This definition is closely related to the familiar HCI concept of *modes*, where an action has a consistent outcome in some contexts, with other outcomes possible in different modes. In our introductory example, we can say that Microsoft Word has (at least) two modes for $\boxed{\text{DEL}}$: one simply deletes characters (and all effects of those characters), the other deletes characters but not all their side-effects. We claim that because the second mode is relatively small compared to the first, it is more constructive to view the situation as a single mode defined by a partial behaviour ("*key*$\boxed{\text{DEL}}$ **usually** deletes *key*") with an occasional deviation from the norm. It makes sense, therefore, to define a partial behaviour with respect to a particular interface mode.

Partial behaviours are relevant to HCI because a user is more likely to overlook these relatively uncommon deviations from 'normal' behaviour. It is common for users to mistakenly generalise the behaviour of one interface mode to another, but in the case of partial behaviours they are unlikely to gain the experience needed to correct the mistake.

The idea suggests a simple methodology for critiquing interface designs: identify the main modes of the interface, along with their respective partial behaviours: can and should the interface be redesigned to remove any of these partial behaviours? As we show below, this methodology can lead to analyses that provide useful design feedback, and that are simple enough to allow automatic support.

Note that identifying a partial behaviour does not mean it should be made a general behaviour — there could be any number of reasons why some minor inconsistency is needed in order to satisfy other design constraints. The interface designer will need to make choices. For instance, in our introductory example empirical evidence of user behaviour may suggest a huge advantage for not 'undoing' capitalisation by hitting $\boxed{\text{DEL}}$, even though this makes the rule for $\boxed{\text{DEL}}$ a partial claim. The important point is that the issue is raised and identified specifically, so it can be considered by the designer.

Our approach goes beyond the well-known design heuristic of attempting to minimise the number of interface modes, as it identifies and, as we shall see, quantifies specific aspects of the interface design that may need to be redesigned. It is also general, amenable to automation (see below) and can be applied in the very early stages of the design process.

So far we have discussed partial behaviour in general terms. In the following sections we propose a concrete formalisation of partiality and use it to analyse some real-world interfaces.

3. QUANTIFYING PARTIALITY

Having defined partial behaviours as those that have minor inconsistencies and argued for their relevance to HCI, we now focus on quantifying the term 'minor'. Measuring *how* partial a behaviour is allows us to make judgements about those which might be misleading to the user. Here we present a simple measure based on the proportion of the interface's state space for which a particular behaviour holds.

We start with a very simple but general concrete model of user interfaces, based on labelled transition systems (LTS). The transition matrix of a LTS is the union of the transition matrices of the LTS associated with each label. We therefore have a set of matrices (that we call *button matrices*), that correspond with user actions — gestures, button presses, mouse clicks, whatever — and which represent an algebra defining the behavioural properties of the user interface [6].

For each user action $\mathcal{A}$, we define a button matrix M as follows (m_{ij} is the element in the ith row and the jth column of matrix M):

$$m_{ij} = \begin{cases} 1 & \text{if action } \mathcal{A} \text{ takes the interface} \\ & \text{from state } i \text{ to state } j; \\ 0 & \text{otherwise} \end{cases}$$

Similarly, interface states are modelled as row vectors. These models are easy to calculate with, are well supported by tools (e.g., *Mathematica*) and can handle non-determinism and guarding (when an action cannot be performed in a state).

By way of example, suppose we have matrices U and D, corresponding to the two user actions of pressing a button $\boxed{\wedge}$ and a button $\boxed{\vee}$. It is likely that $UD = I$ (the product of U and D is the identity matrix). A simple analysis of a system specification can automatically find many useful theorems like this — for more details see [2, 6]. Now on some mobile phone menus, pressing $\boxed{\wedge}$ when at the beginning of a menu does not move the selection point upwards, but a following $\boxed{\vee}$ *does* move down. For such a system, $UD \neq I$.

As argued above, it is productive to treat this as a partial behaviour, which we can denote as $UD \simeq I$. Furthermore, now that there is a concrete model, we can quantify this partiality by counting the proportion of the states that conform to this behaviour. The partiality π of a button matrix formula $A \simeq_\pi B$ over N states is defined as follows:

$$\begin{aligned} \delta_i &= \begin{cases} 0 & \text{if row } i \text{ of } A \text{ and } B \text{ are equal;} \\ 1 & \text{otherwise} \end{cases} \\ \pi &= 1 - \sum \delta_i / N \end{aligned}$$

In other words, π is the fraction of corresponding rows in A and B that are equal. We write this as $A \simeq_\pi B$; obviously $A \simeq_1 B \Leftrightarrow A = B$.

In our $\wedge/\vee$ example above, we could have these actions navigating a menu with, say, 20 states. There will be 18 states for which the actions work as inverses, and hence 18 rows of UD that are equal to I, so

$$UD \quad \simeq_{0.9} \quad I$$

We call such equations *partial theorems.* It is easy to find such theorems automatically. Indeed, we have built a design and analysis tool that can search for exact and partial theorems such as the one above [2].

We next give some more complete examples than this introductory sketch, and then address the problem that we have defined partiality (i.e., the 90%) as partiality over the system states, which may be arbitrarily related to the user's view of the interface, depending on how it is programmed, rather than related to issues of relevance to the user.

4. Example 1: An SMS Editor

The Nokia 3330 is a popular mobile phone, which has a typical SMS editor interface [5]. We use it here to illustrate a partial behaviour analysis, because the kind of problems it has are not unusual.

The editor state has five components: (*i*) The text typed so far; (*ii*) The capitalisation mode, which can be all lower (abc), initial-upper (Abc), or all upper (ABC); (*iii*) The text input mode, either T9 predictive text [3] or multi-press; (*iv*) The alphanumeric mode, either letter input or number input; (*v*) The rest of the text (to the right of the cursor).

In this short paper we ignore (*v*) and model (*i*) as three modes: the cursor is at the start of the message or after a terminating punctuation mark and a single space (sentence mode); after any other kind of space (word mode); it is after a non-space character (character mode). This creates $3 \times 3 \times 2 \times 2 = 36$ states in our model. The interface can only be in 24 of these states, so we may represent the state with 24 bit vector.

The user must employ the $\boxed{\#}$ key to change modes (*ii*)–(*iv*), with different outcomes depending on whether the key depression is long or short and, if short, whether it was preceded by a pause of at least one second. Therefore we model the physical $\boxed{\#}$ key as three conceptual actions: $\boxed{\text{Long}\#}$, $\boxed{\text{Short}\#}$ and $\boxed{\text{Pause}\#}$.

The model allows us to automatically explore the properties of the user actions, as described in [2], including finding partial theorems. One such property involves the $\boxed{\text{Long}\#}$ action, which is used to change the alphanumeric mode, and in our model is represented by a 24×24 matrix L. If we check the properties of this matrix (say, in *Mathematica*) we find no interesting exact theorems, but the following partial theorem is found (to 2DP):

$$L\,L \quad \simeq_{0.92} \quad I \qquad (1)$$

By looking at the non-equal rows of the matrix, we can diagnose why the partial identity is not exact. In letter input/predictive texting mode, there are two states for which $\boxed{\text{Long}\#}$ changes the capitalisation as well as the alphanumeric mode: in sentence mode it changes abc to Abc, and secondly, in character mode it changes ABC to abc. In the 22 other states $\boxed{\text{Long}\#}$ does not change the capitalisation mode.

The behaviour that makes (1) a partial theorem does not, as far as we can see, serve any purpose. Although text input alters the capitalisation mode in certain contexts (e.g., when a new sentence is started it changes to Abc) in order to automatically normalise sentence capitalisation, no text is entered by $\boxed{\text{Long}\#}$ $\boxed{\text{Long}\#}$. We assume that this partial behaviour is a by-product of the interface's implementation, just making it unnecessarily complex to use — though now we know from the analysis, a focused empirical investigation is suggested.

5. Example 2: A CD Player

As a brief second example, the Sanyo CDP-195 portable CD player is designed to play CDs in a number of different ways (e.g., random track play). The play mode is retained when the CD is stopped or paused, and can be altered by pressing the $\boxed{\text{P-Mode}}$ button. Using a 22 state model of the interface, the following partial theorem was generated automatically:

$$P^7 \quad \simeq_{0.95} \quad I$$

where P is the button matrix corresponding to $\boxed{\text{P-Mode}}$. This behaviour corresponds to the fact that the button cycles through 7 different play modes. Examining the P matrix revealed that a single state violates this behaviour. A redesign easily avoids this problem by merging two states, which previously played the same rôle, but appeared very different to the user.

6. The Effects of Modelling

Modelling a user interface using matrices requires a commitment to (*i*) an identification of the system's states and (*ii*) a numbering of states. Clearly, renumbering the states will not affect our analysis of the system's partial theorems. However, identifying states for the system commits us to particular equivalence classes with which to view its behaviour, which very well may affect the analysis.

Representing user actions as matrices means that each action is in fact a linear transformation over a vector space which represents the interface states [4] (where the vector space is over the field Z_2, i.e., 0 and 1). Given that changing the choice of states may change the analysis, we can distinguish between four distinct kinds of change from a state space A to a state space B:

Refactoring There is a linear map from A to B, which have the same dimension (a change of basis).

Abstraction There is a linear projection from A to B, with B having a lower dimension.

Refinement There is a linear projection from B to A, with A having a lower dimension.

Correction There is no linear map that transforms either A to B or B to A.

Importantly, the user will have a mental model of the interface that could be related to a formal interface model by one of these operations. (This does not necessarily mean a cognitive model in the technical sense, but a formal model that the user behaves *as if* they were following.) Next, we give an example result concerning partiality in one of these situations.

If a user's model is a refactored version of the interface, then the partiality of a button identity is bounded above by the term given in Theorem 1. This involves the *nullity* of a transformation, defined as the dimension of its kernel (i.e., the part of the vector space that it maps to $\mathbf{0}$) [4]. The significance of this result is that in this context our measure of partiality may differ from the user's perception, but there is a calculable bound on how partial they can perceive it to be. This bound corresponds to a 'representation free' partiality for models that are related by refactoring.

Theorem 1 (Nullity bound on partiality) *Given a button identity $A \simeq_\pi B$ over N states, with $\pi = p/N < 1$, then*

$$\pi \quad \leq \quad \frac{1}{N}\,\text{nullity}(A - B)$$

Proof The rows of A and B can be reordered so that for $i = 1..p$

$$\mathbf{e}_i.A \quad = \quad \mathbf{e}_i.B$$

where the $\mathbf{e}_i$ are linearly independent. So for the same i, $\mathbf{e}_i.(A - B) = \mathbf{0}$. From the definition of kernel space $\{\mathbf{e}_i\} \subseteq \text{Ker}(A - B)$ and so Span $\{\mathbf{e}_i\} \subseteq \text{Ker}(A - B)$. Considering the dimensions of these spaces

$$p \quad \leq \quad \text{nullity}(A - B)$$

The result follows. ■

7. Conclusions

We have introduced the concept of partial behaviour, along with a formalisation based on matrix algebra, and shown it is a useful, usable and automatable concept for critiquing user interface designs. Our simple, general and formal model of interaction, based on linear algebra, allows us to define partiality in a clear way — indeed, partiality makes no sense for many more complex domains, and we may not have thought of the formalisation as a viable research question but for its elegance in the domain.

However, using matrices requires a commitment to a state numbering, and clearly state numbers are of no relevance to users (and often not to programmers)! We can easily prove the choice of state numbers affects nothing, but *any* hidden choice is a bias we would like to avoid. Users do not know what states are; implementors (or user's mental models) may split some states the formal model treats as identical, and so on. We therefore wanted to find out whether our notion of partiality could survive a transformation in representation, and if so, what use it would be for design.

This paper shows that we have made progress towards exploring representation-independent notions of partiality. We intend to continue this work by using linear algebra to understand how partiality and other interesting properties of interface models behave under changes of representation. Another fruitful direction would be to use more sophisticated notions of partiality, such as weighting that accounts for how often the user is in each state, based on data or predicted usage statistics.

Acknowledgements

Harold Thimbleby is a Royal Society Wolfson Research Merit Award Holder, and acknowledges this support; Jeremy Gow is funded by EPSRC grant GR/S73723/01.

References

[1] Blandford A., Thimbleby H. and Bryan-Kinns N. (2003). "Understanding interaction traps." In Proc. HCI 2003: Design for Society, **2**, pp57–60.

[2] Gow J. and Thimbleby H. (2004). "MAUI: An interface design tool based on matrix algebra." In Jacob R., Limbourg Q. and Vanderdonckt J. (eds), Proc. 5th. Int. Conf. on Computer-Aided Design of User Interfaces (CADUI 2004), pp81–94. Kluwer.

[3] Grover D. L., King M. T., and Kuschler C. A. (1998). Patent No. US5818437, Reduced keyboard disambiguating computer.

[4] Morris A. O. (1978). Linear algebra: An introduction, 2nd edn., Van Nostrand Reinhold.

[5] Nokia UK website (2004). 3330 manual available from support section, `http://www.nokia.co.uk/`

[6] Thimbleby H. (2004). User interface design with matrix algebra, ACM TOCHI, forthcoming.