Mathematical Mathematical User Interfaces

Harold Thimbleby and Will Thimbleby

Department of Computer Science, University of Swansea, SWANSEA, Wales
harold@thimbleby.net, will@thimbleby.net

Abstract. Taking *Mathematica* and *xThink* as representatives of the state of the art in interactive mathematics, we argue conventional mathematical user interfaces leave much to be desired, because they separate the mathematics from the context of the user interface, which remains as unmathematical as ever. We put the usability of such systems into mathematical perspective, and compare the conventional approach with a novel declarative, gesture-based approach, exemplified by *TruCalc*, a novel calculator we have developed.

1 Introduction

TruCalc is a new calculator, with a gesture-based handwriting recognition user interface. This paper reviews its design principles and relates them to the requirements of mathematical user interfaces.

2 The Development of Mathematical User Interfaces

For thousands of years, we've been doing maths by using pencil and paper (or equivalent: quill and scroll, stick and sand—whatever). When calculating devices were invented, this helped us do calculations faster and more reliably, but we still did maths on paper. Comparatively recently, computers were invented, and for the first time we could replace pencils with typed text and get results written down automatically, and then, later, we replaced paper with screens. Mathematics displayed on screens can be manipulated more freely than ever before, yet most calculators running on computers emulate mechanical devices.

Turing famously presented a formal analysis of what doing mathematics entailed [17]. He argued any pencil and paper workings could be reduced, without loss of generality, to changing symbols one at a time from a fixed alphabet stored on an unbounded one dimensional tape. Symbols are changed according to the current state of the device, the current symbol on the tape, and elementary rules. The Turing Machine, which can be defined rigorously (and in various equivalent forms), was a landmark of mathematics and computing. Indeed, the Church-Turing Thesis essentially claims that all forms of computing, and hence mathematics, can be 'done' by a Turing Machine in principle.

Turing introduced his machine with the following discussion:

> "Computing is normally done by writing certain symbols on paper. In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-540-92698-6_37

J. Gulliksen et al. (Eds.): EIS 2007, LNCS 4940, pp. 520–536, 2008.

> be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper."
>
> A. M. Turing [17]

Here, Turing's use of the term 'computing' is historical; he is referring to human computation on paper.

While Turing is formally correct, good choice of notation is crucial to clear and efficient reasoning. Moreover, almost all notations (for example, subscripts) are two dimensional, as suits pencil and paper—and the human visual system. One view of the present paper is that the power—the 'Turing equivalence'—of typical mathematical user interfaces has blinded us to the importance of notation and interactive notation properly integrated with the way the user interface works. Users put up with one-dimensional and other limitations to interaction because the deeper ideas appear sufficiently well supported. A very interesting discussion of Turing Machines and interaction is [3], but the focus of this paper now turns to the design of interactive mathematical systems.

2.1 Conventional Mathematical Interaction

Without loss of generality, mathematicians use pencil, paper and optionally erasers. Pencils are used to draw forms, or to cross them out. Typically, adjacent forms are related by a refinement. Harder to capture formally, the mathematician's brain stores additional material, which is typically less organised than the representation on paper. One might argue that much of the mathematician's work is to find a relation between what is in their head and marks on paper. This is an iterative process. Finally, the concepts and previously unstated thoughts are mapped to some representation such as LaTeX, so that the organised and checked thoughts can be communicated effectively to other brains.

When this process is computerised, the forms are linearised into some character sequence. A string, typed onto 'paper' or a VDU left to right, is transformed by the computer inserting the values of designated expressions. A typical handheld calculator is an example of this style of interaction, though most only display numbers and not the operators—one of their limitations is that the user does not know whether the display is the current number being entered or a result from a previous computation.

Around the 1970s, the sequential constraint became relaxed: the underlying model remained incremental as before, but the user could 'scroll back' and edit any string. Now the values computed may have no relation to the preceding strings, because the user may have changed them: the old output may be incorrect relative to the current string.

More recently, from the late 1980s on, the user interface supported multiple windows, each separately scrollable and editable, each with an independent user interface much like a typographically tidied up 1970s VDU. Of course, this gives enormous flexibility for managing various objects of mathematical concern (proofs, tactics, notes...) [10], especially when supplemented with menus and keyboard commands, but the generality and power should not distract us from

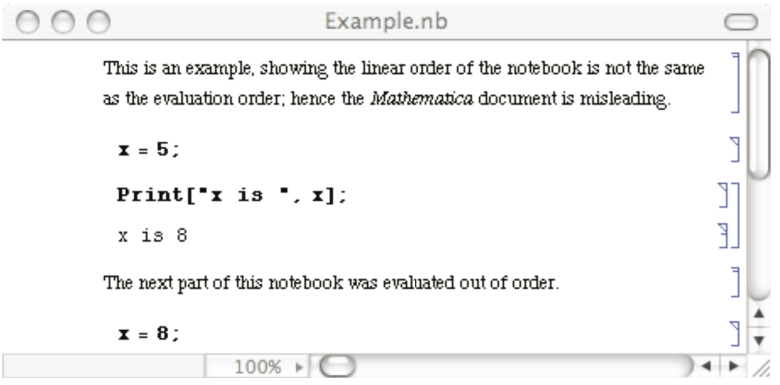


Fig. 1. Example of problematic interaction in *Mathematica*

the relation of the user interface to doing the mathematics itself. Normally we focus on the maths, and ignore the interface; it is just a tool to do the maths, not of particular mathematical interest itself.

Consider *Mathematica* [18]. A *Mathematica* notebook is a scrollable, editable document representing the string. Certain substrings in the notebook are identified, though the user can edit them at any time and in any order. A set of commands, typed or through menu selection, cause *Mathematica* to evaluate the identified substrings, and to insert the output of their evaluations. It is trivial to create *Mathematica* notebooks with confusing text like that shown Figure 1, which illustrates the inconsistency problem (is `x` 5 or 8?) as *Mathematica* separates the order of the visible document from the historical order of editing and evaluation. In the example above, the `x = 5` may have been edited from an earlier `x = 8`; the `Print` may have been evaluated after an assignment `x = 8` evaluated anywhere else in the notebook; or the `Print` may have been edited from something equivalent to `Print["x is 8"]`—and this is not an exhaustive list. In short, to use *Mathematica* a user needs to remember what sequence of actions were performed. (In fact, *Mathematica* helps somewhat as it can show when a result is possibly invalid.)

Although the presentation can be confusing, the flexibility is alluring. While the mathematician can keep the editing and dependencies clear in their head, the notebook (or some subset of it) will make sense.

Mathematica and many other systems add notational features so they can present results in conventional 2D notation. Instead of writing a linearised string, such as 1/2, the user selects a template $\frac{\blacksquare}{\blacksquare}$ from a palette of many 2D forms. The $\blacksquare$ symbols can then be over-typed by 1 and 2, to achieve (in this example), $\frac{1}{2}$. Such mechanisms allow the entry of forms such as

$$\int_0^\infty \sin x^2 \, e^{-x} \, dx \qquad \text{and} \qquad 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cdots}}}}}$$

as shown with relative ease. However, a problem is that the template continues to exist even though the user cannot see it. A simple example illustrates the

problem: editing $\frac{1}{2}$ to 12 is difficult, because the initially hidden template will reappear explicitly in intermediate steps such as $\frac{\blacksquare}{12}$ or $\frac{\blacksquare}{1}2$.

In *Mathematica* a function `TraditionalForm` achieves the inverse: presenting evaluations using standard 2D notation. While these 2D notations look attractive (and indeed are considerably clearer for complex formulae, especially for matrices, tensors and other such structures), they do not alter the semantics or basic style of interaction.

Padovani and Solmi [5] provide a good review of the interaction issues of using 2D notations, such as *Mathematica* and other systems use. They argue that 2D notation requires a model, namely the internal representation of the structure, which is not visible in the user interface. Hence, for the user to manipulate the 2D model new operations are required. The model itself is not visible, so inevitably 2D notation introduces modes and other complexities. That is, it looks good, but is hard to use. Editing operations are performed on non-linear structures (e.g., trees), but the displayed information does not uniquely identify the structure. Like the criticisms of *Mathematica* above, to use a 2D structure requires a user to remember how they built it; worse, what the user has to remember (Padovani and Solmi argue) does not correspond with the user's mental image of the mathematics being edited.

xThink is a different mathematical system [19], and its model is directly based on a 2D representation. *xThink* recognises the user's handwriting in standard notational format, and can compute the answer which is displayed adjacent to the hand-written sum. Provided *xThink* recognises the user's writing reliably, the internal model of the formula is exactly what the user wrote. Nothing is hidden. In this sense, *xThink* solves the problems Padovani and Solmi elaborate, though not all of the problems we attributed to *Mathematica* (as we shall see below).

A typical "page" from *xThink* is shown in Figure 2. Its advantage over *Mathematica*'s template-based approach is the ease and simplicity of entering mathematics, however its interaction style retains the problems of *Mathematica*'s—there is no guarantee the 'answers' are in fact answers to the adjacent formulae, and furthermore *xThink* has introduced new handwriting recognition problems; that is, the formula evaluated may not *ever* be one that was thought to have been written down!

xThink and *Mathematica* are only two examples, selected from a wide range of systems. Maple [2], for example, is closer to *Mathematica* in its computer algebra features, but closer to *xThink* in its handwriting recognition. However, Maple uses handwriting recognition to recognise isolated symbols which are written in a special writing pad—whereas *xThink* allows writing anywhere, but the writing has to be selected (by drawing a lasso around it) before it can be recognised. *xThink*, *Mathematica* and Maple are PC-based systems, and there are also many handheld mathematics systems, such as Casio's ClassPad [1], which allow pen-based input. However, rather than review individual systems, this paper now turns to principles underlying mathematical interaction.

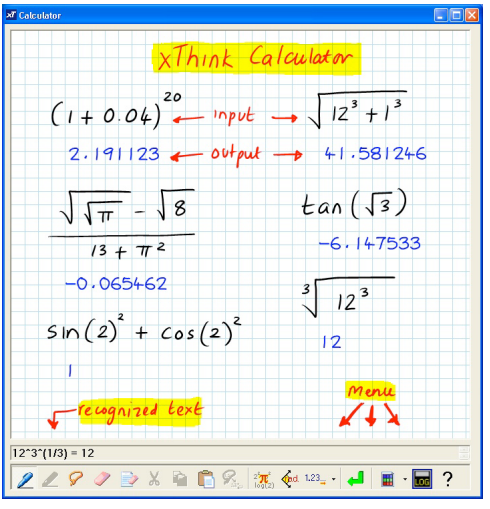


Fig. 2. Example of *xThink*, showing natural handwriting notation combined with calculated output. Picture from *xThink*'s web site [19]; the original is in several colours, making the input/output distinctions clearer than can be shown in greylevels. In the picture, *xThink* has just parsed a handwritten $\sqrt[3]{12^3}$, shown its interpretation at the bottom of the screen (as `12^3^(1/3)=12`), and has inserted a result in a handwriting-like font below the formula.

2.2 Principles for Mathematical Interaction

With such a long and successful history of procedural interaction it is hard to think that it could be improved; systems like *Mathematica* are Turing Complete (upto memory limitations). Interactive mathematical systems, such as *Mathematica* and *xThink*, are clearly very powerful and have a very general user interface. The book $A = B$ [6] gives some substantial examples of what can be achieved.

It is interesting to observe that the representations these mathematical system work with are *not* referentially transparent nor are they declarative. That is they only do mathematics that is 'delimited' in special ways, and the user has to 'suspend disbelief' outside of the theatre that is so delimited. As a case in point, we gave the example above of `x` not having the value it appeared to have (see Figure 1); even allowing for the semantics of assignment, there is no model like lvalues and rvalues that maintains referential transparency [9], without some subterfuge such as having a hidden subscript on all names—which, of course, must exist in the users' mind (if at all) if users are to do reliable mathematical reasoning.

Such Fregean properties as referential transparency[1] are key to reliable mathematical reasoning. Another is his idea of 'concept' that has no mental content, that is, a concept is not subjective. Most interactive systems *require* the user to conceptualise (i.e., make a mental model of) the interaction; they have modes, hidden state dependencies, delays, separated input and output and so on.

[1] Quine introduces the term referential opacity but attributes the idea to Frege [7].

It is ironic that modern mathematical systems are so flexible that they compromise the core Fregean principles—though [12] shows, under broad assumptions, any string-based (i.e., Turing equivalent) user interface interaction properties such as modelessness and undo are incompatible. Modelessness is, of course, an HCI term covering issues such as side effects, referential transparency, declarativeness, substitutivity, etc. Essentially, a purely functional interface is modeless; if one cannot have modelessness and undo (under the assumptions of [12]), any such user interface must be compromised for mathematical purposes. Such observations beg questions: is it possible to modify the style of interaction to preserve the core mathematical properties—and what would be gained by doing so?

3 Modern Mathematical Interaction

We will use *xThink* below to make a side by side comparison with our novel interface, *TruCalc*, to highlight the difference between a truly mathematical system and one that is not.

> ***Note.*** *xThink* is a commercial application available from [19] (PC only), and *TruCalc* from [16] (Mac, PC, Linux).

Both our calculator and *xThink*'s calculator from first glance appear to do the same things. In fact *xThink*'s calculator seems to be more powerful, it can handle annotation, multiple sums, more complex mathematics. Yet ignoring a bullet point comparison and the superficial similarity of the two programs, they are in fact very different.

Both calculators provide a user interface based on handwriting recognition. But this is where the similarity ends!

Our calculator, *TruCalc*, was designed from generative user interface principles [12]; in contrast, *xThink* seems to merely add the idea of utilising the affordance [4] of pen and paper without escaping *Mathematica*-style problems.

To better illustrate the differences between these two superficially similar interfaces we will describe the interaction a user employs to solve a simple sum, along with the potential pitfalls.

3.1 *xThink vs. TruCalc*

A first example problem we compare finding the value of "$(4 + 5)/3$" in *xThink* and in our calculator, *TruCalc*. In both, the user starts by writing the sum on the screen, using a pen (or using their fingers on suitable touch-sensitive screens).

1a In *xThink*, the user must press a button to change *xThink* into selection mode. The user can then select what they have written. They must now press another button to get the selected handwriting recognised. The handwriting is recognised and represented in a separate window, which the user must read to check the accuracy of the handwriting recognition. If the handwriting is misrecognised by *xThink* then without checking the small text at the

bottom of the screen the user can easily be fooled into thinking they have the correct answer. The text at the bottom of the screen is both small and linearised, losing the benefit of the handwritten 2D notation—for example Figure 2 shows the cube root of twelve cubed being calculated, it is printed as `12^3^(1/3)=12`.

1b In *TruCalc*, as the user writes, the hand-written characters and numbers are converted to typeset symbols *without any further user action.* The user feels as if they are writing in typeset characters, and confirming recognition is as natural as checking your own handwriting is legible.

2a In *xThink*, to determine the answer, the user must now press another button to evaluate the recognised formula, and the answer is then displayed somewhere on the screen. In Figure 2 all such answers have been positioned under their respective formulae.

2b In *TruCalc*, the typesetting *includes* solving the equation. In this case, the screen will show a typeset $\frac{4+5}{3} = 3$—the user wrote $\frac{4+5}{3}$ and the computer inserted $= 3$ *in the correct position.*

3a In *xThink*, to determine the answer, the user's input must be syntactically complete (an expression). For example, to find the value of $\sqrt{4}$ the user must write exactly this (and it must be recognised correctly).

3b In *TruCalc*, answers are provided even with incomplete expressions, as well as with equations. For example, to find the value of $\sqrt{4}$ the user can write $\sqrt{}$ then 4, or 4 then $\sqrt{}$, and they can write $=$ if they wish. In any case, the value 2 or $=2$ is also displayed. Furthermore, if the user wrote $\sqrt{} = 2$, then *TruCalc* would insert 4 appropriately, thus solving a type of equation where *xThink* would require the user to write 2^2 (which is notationally different).

4a In *xThink*, the user's handwriting can be altered and hence make the answer (here, 3) invalid—and it will remain invalid until the handwriting is re-selected, recognised and re-evaluated (and the old answer removed). Or several answers may accummulate if the user evaluates formulae and does not remove old answers.

4b In *TruCalc*, as typesetting *includes* solving the equation, the user could continue and write $=$ or $= 3$ themselves. In particular, if they wrote an equation, such as $\frac{4+}{3} = 3$, *TruCalc* would solve it, and insert (in this case) 5.

5a *xThink* provides no other relevant features for the purposes of this paper.

5b In *TruCalc*, the editing of the user's input is integrated into its evaluation. Thus the user can then continue to write over the top of this morphed equation, adding in bits that they consider are missing. For example, if the RHS 3 is changed to 30, the display would morph to $\frac{4+86}{3} = 30$.
It is possible to edit by inserting, overwriting and by drag-and-drop to a bin to delete a selection, or to other parts of the equation to move it. In all cases, the equation *preserves* its mathematical truth, as *TruCalc* continually revises it. *TruCalc* also provides a full undo function, which animates forwards and backwards in time—also showing correct equations.

3.2 In-Place Visibility

With *TruCalc* the replacement of the user's handwriting with typeset symbols not only provides an immediately neat and tidy (and correct) equation but also provides immediate visible feedback of what was recognised. The displayed typeset equation *is* the equation that the answer is shown. This in-place visibility removes confusion and misunderstanding over what the calculator is doing, and whether it has misrecognised bad handwriting.

In our experiments with *TruCalc* [14], one of the outstanding results was that whilst users made intermediate errors, *no* user stopped on a wrong answer. We believe this was because the calculation they were performing was entirely visible and unambiguous to them in an in-place 2D notation.

Without in-place visibility, the user may be unsure which results correspond with which inputs. This compromises mathematical reliability; the user has to rely on their head knowledge.

3.3 No Hidden State; Modelessness

Hidden state and modes compromise mathematical reasoning. Hidden state affects how to interpret input and output; specifically, modes are hidden state (e.g., knowledge of history) in the user's head that is needed to know how to control the user interface predictably.

Typically, a system does not show what mode it is in, but the mathematical interpretation of its display depends on the user knowing some hidden state. For example, in *xThink* to erase or move parts of the equation the user has to select different tools at the bottom of the screen, then when they have finished they have to remember they are in a special mode and reselect the pen tool. The *xThink* interaction style makes this cumbersome approach unavoidable in principle. The relative meanings of displayed results obviously changes when other images are modified; simply, they may become wrong.

The *xThink* user also has to be aware that once they have finished an equation they have to do more (press several buttons, select their text) this time switching mental modes from "entering" to "getting the answer." If they don't change modes (or of they don't change through the modes appropriately, or select inaccurately), there is either a wrong result or no result for the problem.

With *TruCalc* there are *no* hidden modes or state, and no user context switching. Not only is there no menu of different tools but there is no need to switch mental modes or to pause and press an Enter button to make things work. This greatly simplifies the user's mental model and reduces the effort required to use the calculator. *TruCalc* does have a few modes, for example a dragging mode, but these are clearly visible and they are directly initiated and controlled by the user.

Note that in-place visibility and modelessness together give a very strong—and easy to use—interpretation of WYSIWYG (what you see is what you get), as proposed in [11].

3.4 Instant Declarativeness

A system may show the mathematically right answer when the user asks for it; but until they ask for computation, the mathematics is strictly incorrect (or possibly shows a representation of a meta-'undefined'). In *TruCalc* the results are 'instantly' correct, with no user action required.

Declarative programming was popularised through Prolog. Essentially, the programmer writes true statements, 'declaring' them, and Prolog backtracks to solve the equations (sets of Horn clauses in Prolog). Prolog is thus a declarative language—though its user interface isn't.

Likewise, *TruCalc* is declarative. The user writes equations (or partial equations, taking advantage of the automatic syntax correction), and these are declarations that *TruCalc* solves (by numerical relaxation).

In Prolog, the user has to enter a query, typically terminated by a special character. Until that character is pressed, the output (if any) is incorrect. This inconsistency within the interface is what we are used to, even to the extent of accepting the sort of inconsistencies illustrated in Figure 1. But it requires the user to remember the past; they haven't pressed return or some other special character or menu selection yet. If they forget confusion happens.

TruCalc extends declarativeness to *instant declarativeness*, that is, an interface that is always true all of the time. No matter what the user writes the answer shown is *always* correct.

An instantly declarative interface implies that the calculator has to be showing something that is correct even if the user has not finished entering everything, or has a currently incorrect edit. Thus the calculator also has to cope intelligently with partial expressions like $\div 3{+}2$. In our case the calculator fills in place holders that alter the expression as little as possible. There are also problems like $1/0$ or overflow like $10^{10^{10^{\cdots}}}$—these too can be handled by correction (such as showing $1/0$ as $1/(0+1)$; see [13]), or by changing the algebra implemented by *TruCalc*.

This instant declarativeness removes the disparity between the input and the output, removing an enormous potential for user confusion and it also removes the need for the user remembering having to press the "equals" button (or some other change mode button) to get an answer.

The implementation of instant declarative user interfaces is only slightly more complex than conventional user interfaces; at least two threads are required, one for the user input, one for processing. Processing restarts every time the user extends or changes the input; in *TruCalc* there is a short delay, which allows the user to write an expression fluidly without visual interference from it being morphed into recognised text until they finish or pause.

3.5 Equal Opportunity

The power of *TruCalc*'s implementation of instant declarativeness combines powerfully with equal opportunity [8]. Unlike *xThink*, *TruCalc* does not distinguish in principle between the user's input and its own output. Each has 'equal opportunity' in the equation. This makes it possible to write on both sides of an equality.

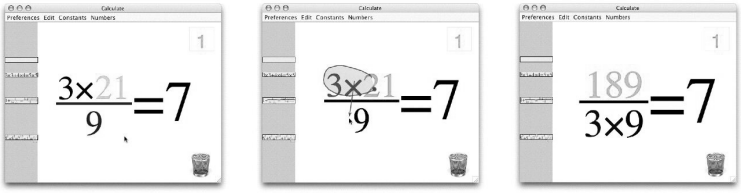


Fig. 3. Example of drag and drop interaction in *TruCalc*, shown as three consecutive screen-shots. Initially, the user has written $\frac{3\times}{9} = 7$; next, the user drags the $3\times$ numerator to the denominator; finally, *TruCalc* provides the correct numerator. The *only* user interaction to achieve this transformation is to draw the loop (shown in the middle figure) and drag it. Had the user had dragged the $3\times$ to the wastebasket, it would have been deleted, and the equation would be corrected to $\frac{54}{9} = 7$. (If a loop is drawn not containing anything to select, it is recognised as a zero).

The ability to change either the answer or the question lets a user solve problems simply that they would have struggled with otherwise. For example, "what power of 2 is 100" can be solved directly without logarithms. (For example, the user writes $2 = 100$, which is corrected to $2 = 100 - 98$, then writes a decimal point as the exponent of 2, which is where they want the answer. $2^{\cdot} = 100 - 98$ then morphs to $2^{6.643856} = 100$.)

Equal opportunity is not in itself a feature that is required for a highly mathematical user interface, but it is a natural generalisation (from expressions to equations) that significantly increases the power of the user interface for the user.

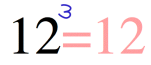
3.6 Rearranging

In *xThink*'s calculator it is possible to delete things or move them around but it is always an awkward process involving many mode changes and it is fairly limited in what it achieves. Moreover, *any* editing in *xThink* breaks the relation between written input and calculated output, and the user has to remember to re-evaluate an edited formula. Hence, in *xThink* the ability rearrange introduces modes and hidden state.

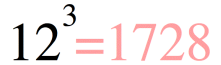
In *TruCalc* the ability to drag and drop an arbitrary part of the equation elsewhere is synchronised by *TruCalc*'s ability to morph the result into a new typeset equation. It is therefore possible to move parts of the equation around without regard for their size or shape, and the user *always* sees a fully correct equation.

More specifically, in *xThink* drag-and-drop is achieved by choosing the selection tool, drawing around the object, then dragging, then choosing the next tool to use; however, once moved, the formula typically needs explicitly selecting, recognising, and evaluating, as further steps for the user. In *TruCalc* drag-and-drop is achieved by drawing around an object and moving it. No mode change is required, and no action needs to be taken to evaluate the new formula. Figure 3 illustrates some simple examples.

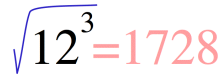
TruCalc has just recognised a handwritten 1, and shown the (at this moment) correct equation 1 = 1; the user is now writing 2 by hand.



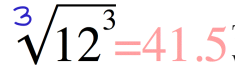
TruCalc has recognised the 2; the user is writing 3 as an exponent.



TruCalc has recognised the 3, and updated the RHS of the equation.



The user is writing a $\sqrt{\ }$ around the 12^3. Of course, the user could equally have started by writing the $\sqrt{\ }$, and then writing inside it.



The $\sqrt{\ }$ is recognised, the RHS is updated, and the user has started to write 3.

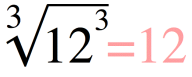


Fig. 4. A step-by-step, broken-down example of using *TruCalc* on the sum that *xThink* is shown solving in Figure 2, showing how a single equation changes as the user writes on it. This brief example does not show drag-and-drop, nor equational calculations. However, notice that *TruCalc* provides continual correct feedback; there are no hidden modes, no special commands—*TruCalc* just 'goes ahead' and provides in-place answers. The user feels as if they are writing in a formal typeface (here, Times Roman). This brief example does not show how *TruCalc* would handle solving equations, for instance if the user dragged the 12 onto the RHS. Had the user written an = themselves on the left of their formula, then the answers would have been shown on the LHS.

4 A Demonstration of *TruCalc*

Because *xThink* is not highly interactive, ironically, its screen shots (such as Figure 2) make it easier to understand than screen shots of *TruCalc*! *xThink*'s screen shots show handwriting input, the recognised input (shown in the bottom pane), and the result. Figure 2 shows several such examples. It looks straight forward—except, as we showed in Section 3.1, *constructing* the interesting display of Figure 2 requires transitions between many modes, and hence possible user errors. Figure 4 shows *TruCalc* solving the problem that *xThink* is shown solving in Figure 2; however, *xThink* solves the equation in one step and requires changing modes, whereas *TruCalc* solves continually, in place, and needs no modes at all. (In this short paper we do not illustrate how *TruCalc* can solve equations more powerfully than *xThink*—by combining rearranging with equal opportunity; see [13] for examples.)

5 Other Features of *TruCalc*

TruCalc provides other features that make it more powerful and easier to use. These features support, but are semantically unrelated to the highly interactive

way it does mathematics. Further discussion of *TruCalc*, beyond the scope of the present paper, can be found in [14] and [15].

5.1 Ink Editing

In *xThink*, the user writes a formula *then* asks for it to be recognised. In *TruCalc*, the formula being written is *continually* being recognised. This permits a very powerful, and natural, interaction style we call ink editing.

If the user writes '−' it is recognised as a minus sign. If they write 2 above it, the minus sign becomes a division bar. If they cross it out by a vertical stroke, it becomes a + sign.[2] None of these natural ink editing operations makes sense in a batch recogniser.

5.2 Dock

TruCalc provides a dock, with functionality similar to the dock in Mac OS X. That is, a whole or partial equation can be dragged to the dock, and it will be stored as an item. Conversely, any item in the dock can be clicked on, and it will replace the current equation. If an item is dragged out, it 'comes out' as a picture representing its value. Hence an equation such as $1 + 2 = 3$ might be dragged out of the dock and used, say, as an exponent, as in

$$2^{\boxed{1+2=3}} = 8$$

(the subequation is boxed, as it cannot be edited except by recalling it from the dock); such dock items can be used in many places in any other equation. The dock serves as a convenient declarative memory for the user.

The dock would be a very natural way to extend *TruCalc* to have variables, at least if entries in the dock could be named. Indeed, dock entries might be associated with URLs, and be able to represent internet resources—such as the current dollar/euro conversion rate, or standard numbers and equations, and so on.

5.3 Optionally Hidden Answers

TruCalc shows correct answers at all times, just as we have described it. However, for use in teaching, it is possible to hide the answer, and show an empty box. This indicates to a student that their answer is wrong or incomplete, and some correction is still required. Here is an example:

$$2 + \square = 3$$

where normally it would show $2 + 1 = 3$.

[2] The current implementation of ink editing is not complete; for example you cannot edit − to 4, or edit . to ! in the obvious ways yet.

5.4 Undo

TruCalc provides the ability to undo edits and alterations by means of a clock metaphor. A user grabs the clock hands and can 'rewind the time,' and as they do so the symbols and numbers animate back through time exactly as they were morphed. The morphing provides a temporal continuity between the different steps of the calculation, and it can be played backwards and forwards (i.e., undo and redo).

5.5 Possible Extensions to *TruCalc*

TruCalc can be extended in many ways. We give a few examples:

1. The dock could be on a web site, and made multiuser so several people can collaborate. The dock could also have a palette of functions ($\log, \sin$ etc) that, like the current equations, could be dragged into the working equation.
2. The back-end could be replaced with (for example) the *Mathematica* kernel so it was extensible. Currently, *TruCalc* only does complex numerical arithmetic; it could provide an interface to anything *Mathematica* etc can do.
3. Unlike *xThink*, *TruCalc* currently provides no way for a user to write things that are *not* recognised; formulae cannot be annotated, arrows cannot be drawn, and so on. A teacher would probably like another colour which can be used to draw freely with but which *TruCalc* does not interpret.

There are many obvious developments: complete handwriting recognition, to extend *TruCalc* to standard function notation (such as log); restrictions for teaching purposes (*TruCalc* uses complex arithmetic); multiple equations on the screen, like *xThink*. And so on.

However, what *TruCalc* does is show how effective—both reliable and indeed enjoyable (see §6.1)—a user interface for mathematics can be when the interaction, the user interface, itself respects the principles of mathematics.

6 Mathematical Mathematical Interfaces Lead into HCI

HCI is the science and art of making user interfaces more effective (and enjoyable) for humans (though HCI techniques have also been used to improve user interfaces for farm animals!).

TruCalc allows the user to write an equation e involving complex numbers from $\mathbb{C}$ and elementary arithmetic operators. *TruCalc* has no variable names, but uses slots; thus, in conventional terms, the equations can contain variables without repetition—future versions of *TruCalc* may include variable names as they are of course useful for many purposes, not least in providing mnemonics for the slots as currently used.

The variety of solutions S of e is intended to be $S(e, \mathbb{C})$, except the current version implements $\mathbb{C}$ by $\mathbb{C}_J$, the obvious approximate representation of $\mathbb{C}$ using pairs of Java double precision floating point numbers.

With these clarifications, we can express some important HCI issues:

1. What should *TruCalc* do when $S(e, \mathbb{C}_J)$ does not determine a unique solution? Currently *TruCalc* uses heuristics to try to find solutions that are principal values, identities of operators, and so on. For example $\times = 10$ will be solved by $10 \times 1 = 10$, using the right identity of $\times$. On the other hand, $10^{\frac{1}{7}} \times 10^{\frac{1}{7}} = 10$ has no solution as currently implemented, because *TruCalc* effectively tries to solve $1/x = 0$.
2. What should *TruCalc* do when $S(e, \mathbb{C}_J) = \emptyset$? *TruCalc*'s solution is to show ? symbols (or $?+?i$); however, an earlier version [13] modified the equation so that at least one solution could be found. Neither solution, we feel, is entirely satisfactory, since $S(e, \mathbb{C}_J) = \emptyset$ can occur as a transient step in entering a solvable equation—for example, to enter $1/0.1$ either requires contortions or the intermediate step $1/0$.
3. What should *TruCalc* do when there is a *humanly*-obvious algebraic solution, but $S(e, \mathbb{C}_J) = \emptyset$? For example, the very easily entered LHS

$$2^{2^{2^{2^2}}} = ?+?i$$

   fails because it is a 19,729 digit decimal number, which is in $\mathbb{C}$ but not in $\mathbb{C}_J$—but the equation could be solved as

$$2^{2^{2^{2^2}}} = 2^{65536}$$

   or in many other equivalent symbolic ways. Which is best? Should the user have choices, and if so, how? A symbolic approach would also be a good way to solve equations the user enters containing $1/0$ terms.
4. Can users choose $S(e, \mathbb{R}), S(e, \mathbb{Q}), S(e, \mathbb{Z}), S(e, \mathbb{N})$, for instance for elementary teaching? What about $S(e, \mathbb{Z}_{12})$ for clock numbers, or $S(e, F_p)$, and other interesting domains, say predicate logic or even chess?
5. Improving the handwriting recognition would allow the solution of larger classes of equations, for instance that include transcendental functions.
6. *TruCalc* uses $=$ as an operator over $\mathbb{C}_J$, not $\mathbb{C}$. This can result in (apparently) peculiar results such as the following:[3]

$$\begin{aligned} \pi &= 335/113 \\ \pi &= 3.142 \\ 3.142 &= 1571/500 \\ \pi &= 3.142 - 4.073 \times 10^{-4} \end{aligned}$$

   Perhaps *TruCalc* should use an operator $\simeq$ when the equality is approximate? (Although results that are approximate in $\mathbb{C}_J$ may be exact in $\mathbb{C}$!)

[3] The last example shows 4.073×10^{-4} which in an earlier version would have been presented in the standard Java format as $4.073E-4$, a 'buggy' notation, because a user could not enter E themselves, so it failed equal opportunity. Here, equal opportunity is seen to be a *generative* design principle: given the existing features, it suggested improvements.

7. *TruCalc* could explicitly show, where it is the case, that numbers are approximate. For example, $\pi =_{[3]} 3.142$ could be the notation to indicate the equality is correct to three decimal places. If the user changed the subscript 3, they would be changing the precision of the displayed number. Chaitin however suggested that it would be more in keeping with the direct manipulation style of *TruCalc* to allow the user to drag the righthand extension of decimals: so if the user drags the '...' to the right in the equation $\pi = 3.142\ldots$ it could become $\pi = 3.141592653589793\ldots$; and dragging the '...' left would put it back to $\pi = 3.1\ldots$, for example.

In summary, an interesting part of the 'HCI of *TruCalc*' can be expressed as the relation between $S(e, \mathbb{C}_J)$, the solutions the implementation provides for an equation e, and $S(e, \mathbb{H})$, what the user expects.

6.1 Enjoyment

Finally, it surprised us that *TruCalc* was fun to use—we had developed it from principles and had not anticipated the strong feeling of engagement it supports. It integrates body movement, handwriting, and instant *satisfaction*, that children and post-doc mathematicians find exciting. Elsewhere we have reported on our usability surveys, a topic that is beyond the scope of this paper [14]. More recently *TruCalc* was exhibited at the Royal Society Summer Science Exhibition, where it was used by thousands of visitors, children, parents, teachers, to math postdocs. An exit survey was completed by 420 participants (and we insisted that anybody who took a survey form completed it, to avoid under-reporting of negative results) had 90% **liked** or **really liked** *TruCalc*, and nobody (0%) **disliked** it.

7 Conclusions

Current leading mathematical systems are capable of a remarkable range of mathematics. With *Mathematica*, a market leading example of an interactive computer algebra system, we are able to solve problems we could not do without it. It is easy to confuse these mathematical capabilities with usability. So much power seems harnessed that the power seems usable.

This 'power leverage' blinds us to the fundamental non-mathematical nature of these user interfaces. Often clear mathematical principles like referential transparency and declarativeness are lost in modes, history dependence, context sensitivity, and so on. The failure of these principles in conventional mathematical user interfaces undermines our ability to reason reliably or mathematically.

xThink makes use of the affordance of pen and paper to create an interface that solves partially some of the interface issues. But it still ignores basic mathematical principles when applied to interaction. It gains the affordance of paper, at the expense of introducing evaluation modes (and uncertainty in the handwriting recognition).

We have shown in *TruCalc* that it is possible to create an interface that supports basic principles throughout the user interface; it has no hidden state, is modeless, instantly declarative, and so on—or in Frege *et al.'s* metamathematical terms, substitutional, referentially transparent, and so on. Adhering closely to these mathematical principles do not compromise the power of *TruCalc*; it is in principle as powerful mathematically as *xThink* and other conventional systems (though obviously the two systems vary in detail, such as in the choice of built-in functions they support)). Further, we have shown that by supporting these principles that the calculator is easier, more enjoyable, fun and usable—a paradigm shift in usability.

Acknowledgements. Harold Thimbleby was supported by a Royal Society-Wolfson Research Merit Award, and Will Thimbleby by a Swansea University studentship. The design of *TruCalc* is covered by patents. Paul Cairns, Greg Chaitin, James McKinna, John Tucker and very many anonymous participants in demonstrations and lectures gave us very useful comments. The Exhibition of *TruCalc* at the Summer Science Exhibition at the Royal Society was funded by EPSRC under grant EP/D029821/1, and Gresham College.

This paper was originally an invited talk at the Mathematical User-Interfaces Workshop 2006 (`http://www.activemath.org/~paul/MathUI06`), but did not appear in the proceedings.

References

1. Casio, Casio ClassPad 300 Resource Center (2006), `http://www.classpad.org`
2. Garvan, F.: The MAPLE Book. CRC Press, Boca Raton (2001)
3. Goldin, D.Q., Keil, D.: Persistent Turing Machines as a Model of Interactive Computation. Foundations of Information and Knowledge Systems, 116–135 (2000)
4. Norman, D.A.: Affordances, Conventions and Design. Interactions 6(3), 38–43 (1999)
5. Padovani, L., Solmi, R.: An Investigation on the Dynamics of Direct-Manipulation Editors for Mathematics. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 302–316. Springer, Heidelberg (2004)
6. Petkowšek, M., Wilf, H.S., Zeilberger, D.: A = B. A K Peters (1996)
7. Quine, W.V.O.: Word and Object. MIT Press, Cambridge (1960)
8. Runciman, C., Thimbleby, H.: Equal opportunity interactive systems. Int. J. Man-Mach. Stud. 25(4), 439–451 (1986)
9. Tennent, R.D.: Principles of Programming Languages. Prentice-Hall, Englewood Cliffs (1981)
10. Théry, L., Bertot, Y., Kahn, G.: Real Theorem Provers Deserve Real User-Interfaces. In: Proc. Fifth ACM Symposium on Software Development Environments, pp. 120–129 (1992)
11. Thimbleby, H.: What You See is What You Have Got—A User-Engineering Principle for Manipulative Display? First German ACM Conference on Software Ergonomics. In: Proc. ACM German Chapter, vol. 14, pp. 70–84 (1983)
12. Thimbleby, H.: User Interface Design. Addison-Wesley, Reading (1990)
13. Thimbleby, H.: A New Calculator and Why it is Necessary. Computer Journal 38(6), 418–433 (1996)

14. Thimbleby, W.: A Novel Pen-based Calculator and Its Evaluation. In: Proc. ACM NordiCHI 2004, pp. 445–448 (2004)
15. Thimbleby, W., Thimbleby, H.: A Novel Gesture-Based Calculator and Its Design Principles. In: Proc. BCS HCI Conference, vol. 2, pp. 27–32 (2005)
16. Thimbleby, W., Thimbleby, H.: TruCalc (2006), http://www.cs.swan.ac.uk/calculatorshttp://www.cs.swan.ac.uk/calculators
17. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. In: Proc. London Mathematical Society, Series 2, 42, 230–265 (1936/7) (corrected Series 2, 43, 544–546 (1937))
18. Wolfram, S.: The Mathematica Book, 4th edn., Cambridge (1999)
19. xThink, xThink Calculator (2006), http://www.xThink.com/Calculator.htmlhttp://www.xThink.com/Calculator.html