Mathematical mathematical user interfaces

Harold Thimbleby & Will Thimbleby
Department of Computer Science
University of Swansea
SWANSEA, Wales

DRAFT

Abstract

All Turing Complete calculators are mathematically equivalent, and therefore mathematical user interfaces need not innovate beyond what is absolutely necessary. Typically, user interfaces are character-based and imperative, with mice used to select windows, and modes used to control the hidden complexities of the system.

Using *Mathematica* and *xThink* as representatives of the state of the art in interactive mathematics, we show 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, then we compare the conventional approach with our highly interactive approach, as exemplified by *TruCalc*.

1 Introduction

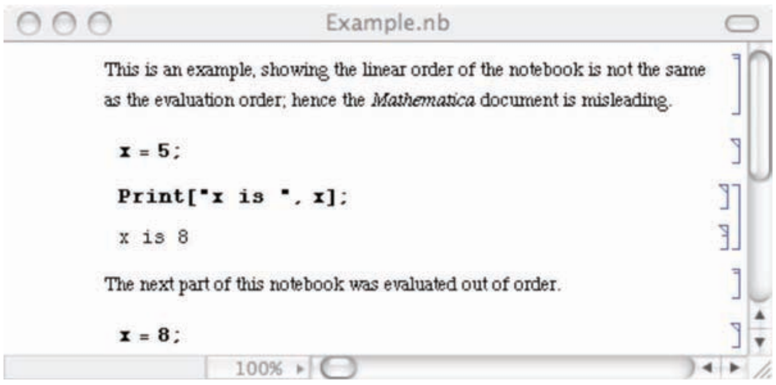
For thousands of years, we've been doing maths by using pencil and paper (or equivalent: quill and scroll, stick and sand — whatever). When calculating devices were invented, this helped us think, but we still did maths on paper. Then, comparatively recently, computers were invented, and for the first time we could replace pencils with typed text and get results written down automatically, and then, later, we could replace paper with screens

Turing famously presented a formal analysis of what doing mathematics entailed [11]. 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 and the current symbol on the tape. 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 is that all forms of computing, and hence mathematics, can be 'done' by a Turing Machine in principle.

"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 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 [11]

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



This is an example, showing the linear order of the notebook is not the same as the evaluation order; hence the *Mathematica* document is misleading.

```
x = 5;

Print["x is ", x];

x is 8
```

The next part of this notebook was evaluated out of order.

```
x = 8;
```

Figure 1: Example of problematic interaction in *Mathematica*.

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.

2 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 mapping from what is in their head onto marks on paper, which is an iterative process, and then to map that and previously unstated thoughts to a linear representation such as LaTeX, so that the organised and checked thoughts can be communicated 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. Around and after the 1970s, the sequential constraint was relaxed: the underlying model is incremental as before, but the user can '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 supports multiple windows, each 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. . .) [6], especially when supplemented with menus and keyboard commands, but the generality and power should not distract us from 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*. 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 similar systems add notational features, generally so that 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

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

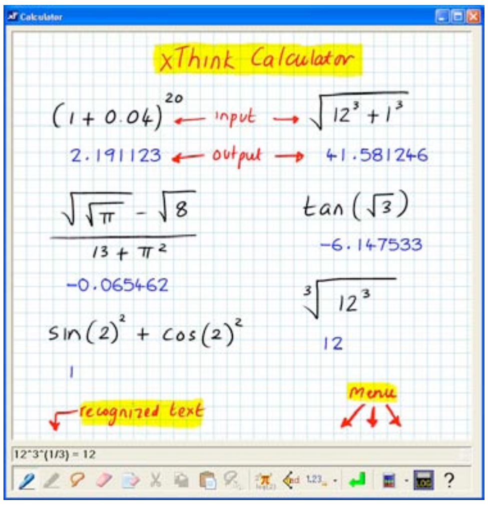
exactly as shown here with relative ease.

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 [2] 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 [12], 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!



$(1+0.04)^{20}$

2.191123

$\sqrt{12^3+1^3}$

41.581246

$\frac{\sqrt{\sqrt{\pi}}-\sqrt{8}}{13+\pi^2}$

-0.065462

$\tan(\sqrt{3})$

-6.147533

$\sin(2)^2+\cos(2)^2$

1

$\sqrt[3]{12^3}$

12

```
12^3^(1/3) = 12
```

Figure 2: Example of *xThink*, showing natural handwriting notation combined with calculated output. Picture from *xThink*'s web site [12]; 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.

3 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 $A = B$ [3] gives some substantial examples of what can be achieved.

Notwithstanding their completeness, 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 referentially transparency [5], without some subterfuge such as having a hidden subscript on all names — which, of course, must exist in the user mind (if at all) if users are to do reliable mathematical reasoning.

Such Fregean properties as referential transparency 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.

It is ironic that modern mathematical systems are so flexible that they compromise the core Fregean principles — though [7] shows, under broad assumptions, any string-based (i.e., Turing equivalent) user interace 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 [7]), 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?

4 Highly mathematical interaction

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

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 [7]; in constrast, *xThink* seems to merely add the idea of utilising the affordance [1] 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.

4.1 *xThink v 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 special button to get the handwriting recognised. The handwriting is recognised in a separate window, which the user must read to check the accuracy of the handwriting recognition. If the handwriting is miss-recognised 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 the cube root of a half is printed as `(1/2)^(1/3)`.

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 press another button, and the answer is 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. 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.

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 them, or to other parts of the equation to move them. 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.

4.2 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 miss-understanding over what the calculator is doing, and whether it has miss-recognised bad handwriting.

In our experiments with *TruCalc* [9], 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.

4.3 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 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 press the $\boxed{\text{Enter}}$ button, this time switching mental modes from "entering" to "getting the answer." If they don't change modes, there is no result shown for the problem.

With *TruCalc* there are *no* modes or hidden 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 $\boxed{\text{Enter}}$ button to make things work. This greatly simplifies the user's mental model and reduces the effort required to use the calculator.

4.4 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 [8]), or by changing the algebra implemented by *TruCalc*.

This instant declarative-ness 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 to get an answer.

4.5 Equal opportunity

The power of *TruCalc*'s implementation of instant declarativeness combines powerfully with equal opportunity. 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.

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.

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.

4.6 Rearranging

In *xThink*'s calculator it is possible to delete things or move them around but it is always an awkward process involving several 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-evaluated 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.

5 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 4.1, *constructing* the interesting display of Figure 2 requires transitions between many modes, and hence possible user errors. Figure 3 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.)

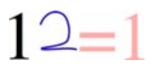
6 Other features of *TruCalc*

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 [9] and [10].

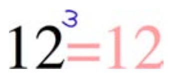
7 Conclusions

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.

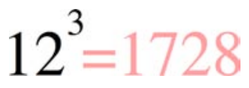
This 'power leverage' blinds us to the fundamental non-mathematical nature of these user interfaces. Often clear mathematical principles like referential transparency and declarativeness are lost in modes, history dependence, context sensitivity, and so on. The failure of these



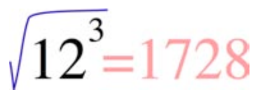
TruCalc has just recognised a handwritten 1, and shown the (at this moment) correct equation $1 = 1$; the user is now writing 2 by hand.



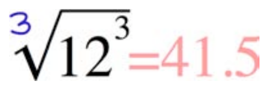
TruCalc has recognised the 2; the user is writing 3 as an exponent.



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.

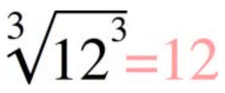


Figure 3: A step-by-step, broken-down example of using *TruCalc* on the sum that *xThink* is shown solving in Figure 2, showing how a single equation changes as the user writes on it. Notice that *TruCalc* provides continual correct feedback; there are no modes, no special commands — *TruCalc* just 'goes 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.

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 Frege's basic principles throughout the user interface; it has no hidden state, is modeless, instantly declarative, and so on. 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 their 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 is 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. James McKinna gave us useful comments.

References

[1] D. A. Norman. “Affordances, Conventions and Design,” *Interactions* **6**(3):38–43, ACM Press, 1999.

[2] L. Padovani & R. Solmi, “An Investigation on the Dynamics of Direct-Manipulation Editors for Mathematics,” *Proc. MKM 2004*, LNCS **3119**:pp302–316, 2004.

[3] M. Petkowvšek, H. S. Wilf & D. Zeilberger, $A = B$, A K Peters, 1996.

[4] Runciman C., Thimbleby H., “Equal opportunity interactive systems,” *Int. J. Man-Mach. Stud.* **25**(4):439–4, 1986.

[5] R. D. Tennent, *Principles of Programming Languages*, Prentice-Hall, 1981.

[6] L. Théry, Y. Bertot & G. Kahn, “Real Theorem Provers Deserve Real User-Interfaces,” *Proc. Fifth ACM Symposium on Software Development Environments*, pp120–129, 1992.

[7] H. Thimbleby, *User Interface Design*, Addison-Wesley, 1990.

[8] H. Thimbleby, “A New Calculator and Why it is Necessary,” *Computer Journal*, **38**(6):pp418–433, 1996.

[9] W. Thimbleby, “A Novel Pen-based Calculator and Its Evaluation.” *Proc. ACM NordiCHI 2004*, pp445–448, 2004.

[10] W. Thimbleby & H. Thimbleby, “A Novel Gesture-Based Calculator and Its Design Principles,” *Proc. BCS HCI Conference*, **2**:pp27–32, 2005.

[11] A. M. Turing, “On computable numbers, with an application to the Entscheidungsproblem,” *Proc. London Mathematical Society*, Series 2, **42**:pp230–65, 1936/7 (corrected Series 2, **43**:pp544-6, 1937).

[12] xThink, `http://www.xThink.com/Calculator.html`, 2006.