

Dealing with life in the cells: An experimental study

David Wakeling¹

Bioinformatics Group, University of Exeter, Exeter, United Kingdom

Abstract

Cell biologists have access to the “parts list” data for the cells of many organisms, but limited understanding of how these parts go together to make a cell “come alive”. A cell simulator could improve this understanding considerably. Our goal is to make cell simulation as convenient as spreadsheet calculation by making the simulator resemble a spreadsheet. In this paper, we describe our prototype simulator and report on our early experience with it.

Key words: cell simulation, spreadsheet

1 Introduction

Cell biologists have access to “parts list” data for the cells of many organisms, but limited understanding of how these parts go together to make a cell “come alive”. A cell simulator could improve this understanding considerably, as Tomita remarks

To say that we understand the overall behaviour of the cell, we must be able to answer questions such as: ‘How would the cell behave if we change the environment, for example, by adding or decreasing a certain substance?’ and ‘What is the result if a certain gene gets knocked out or over-expressed?’ Slightly more sophisticated questions include ‘What gene needs to be inserted for the cell to behave in such a way?’ and ‘What is the ideal culture medium in which to maximize the cell’s ability to do such a thing?’ [10].

Our goal is to make cell simulation as convenient as spreadsheet calculation by making the simulator resemble a spreadsheet. In this paper, we describe our prototype simulator and report on our early experience with it. The paper is organised as follows. Section 2 provides a brief reminder of how ordinary differential equations are solved numerically. Section 3 describes our simulator. Section 4 gives its evaluation and typing rules. Section 5 presents an example

¹ Email: D.Wakeling@exeter.ac.uk

model. Section 6 shows how this model can be simulated with our simulator. Section 7 shows how it can be simulated with the E-Cell simulator. Section 8 reports the results of a small an experimental study. Section 9 considers some closely related and possible future work. Section 10 concludes.

2 Solving ordinary differential equations numerically

An *ordinary differential equation* takes the form $dx/dt = f(t, x)$, and describes how the *dependent variable*, x , representing some value of interest changes with the *independent variable*, t , usually representing time. An *initial value problem* is one where the value of $x = x_0$ is specified at time $t = t_0$. A simulation must solve systems of such equations numerically, and one common way to do so is to use the fourth-order Runge-Kutta method, summarised by the equations

$$\begin{aligned} k_1 &= hf(t_n, x_n) \\ k_2 &= hf(t_n + \frac{h}{2}, x_n + \frac{k_1}{2}) \\ k_3 &= hf(t_n + \frac{h}{2}, x_n + \frac{k_2}{2}) \\ k_4 &= hf(t_n + h, x_n + k_3) \\ x_{n+1} &= x_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} \\ t_{n+1} &= t + h \end{aligned}$$

where the constant h is the time step size.

3 Our prototype simulator

Our prototype simulator incorporates several ideas.

3.1 A new approach to functions

Recently, Peyton Jones *et. al.* proposed a new approach to functions in spreadsheets [6]. Functions are entered into a worksheet², with some cells allocated for the arguments, and another for the result. See figure 1. Here, the workbook has a worksheet for the function `f2c` that converts from Fahrenheit to Celsius. By (our) convention, the first few cells in column A of a worksheet that defines a function are allocated for the argument values, and the one below those is allocated for the result formula.

² Throughout this paper, we use the terminology and features of Microsoft's popular Excel spreadsheet for our discussion and examples. Other spreadsheets are similar.

	A	B	C
1		Fahrenheit argument	
2	= B4 * 5 / 9	Celsius result	
3			
4		= A1 - 32	

Fig. 1. A worksheet defining a function.

3.2 A history mechanism

Wray and Fairbairn introduced a *history mechanism* in their *Nas* spreadsheet [13]. Here, the value of a cell may depend on its own “last” value, and calculation takes place when a **Tick** button is pressed. Thus, a cell might contain the value and formula $42 = \text{last} + 1$ which, after pressing **Tick**, would change to $43 = \text{last} + 1$. There must always be a value in a cell with a formula involving **last**.

3.3 A built-in function for solving ordinary differential equations

Spreadsheets usually provide a number of *built-in functions* for common mathematical, statistical and scientific operations. In the same way, a built-in function can be provided for solving ordinary differential equations. To solve an equation of the form $dx/dt = f(t, x)$ with an initial value $x = x_0$ at $t = t_0$, a cell might contain the value and formula $x_0 = \text{ODE}(f(t, \text{last}), h)$, where t is a reference to a cell containing the value and formula $t_0 = \text{last} + h$, and h is a reference to a cell containing the value of the step size. Each ODE application takes one step towards solving the equation. Again, there must always be a value in a cell with a formula involving an ODE application, which must always be outermost.

Most often, a *system* of ordinary differential equations must be solved, in which several equations are defined in terms of one another. In a functional setting, each of these equations must be passed the current values of the others as additional arguments, $a_1, \dots, a_n$. A definition of ODE using the Runge-Kutta method is as follows

$$\begin{aligned}
 &\text{ODE}(f(t, a_1, \dots, a_n, x), h) \\
 &= x + k_1/6 + k_2/3 + k_3/3 + k_4/6 \\
 &\quad \text{where} \\
 &\quad k_1 = hf(t, a_1, \dots, a_n, x) \\
 &\quad k_2 = hf(t + h/2, a_1, \dots, a_n, x + k_1/2) \\
 &\quad k_3 = hf(t + h/2, a_1, \dots, a_n, x + k_2/2) \\
 &\quad k_4 = hf(t + h, a_1, \dots, a_n, x + k_3)
 \end{aligned}$$

Of course, other definitions are possible using other methods.

4 Evaluation and typing rules

In programming language research, valuable clarity comes from formalising rules for evaluation and typing, and we do something similar for our prototype simulator.

4.1 Terms and types

A *workbook* maps from names to worksheets, and a *worksheet* from cell references to cells. A *cell* is a pair of expressions representing a value and a formula, and an *expression* is either a label, a number, a cell reference, the history variable, or the application of a name to an argument. See figure 2. In reality, of course, it is essential to allow the basic mathematical operators,

$$\begin{array}{lll}
 w & ::= & \{g_i \Rightarrow s_i\}_{i=1}^n \quad \text{---} \quad \text{workbooks} \\
 s & ::= & \langle r_i \Rightarrow c_i \rangle_{i=1}^m \quad \text{---} \quad \text{worksheets} \\
 c & ::= & (e_1, e_2) \quad \text{---} \quad \text{cells} \\
 e & ::= & l \mid n \mid r \mid \text{last} \mid g(e) \quad \text{---} \quad \text{expressions}
 \end{array}$$

Fig. 2. The syntax of terms.

parentheses for grouping, built-in functions, and application of a worksheet to more than one argument. These extensions present no difficulty, and are omitted here only to save space.

A *type* is either a label, number or function type. See figure 3.

$$T ::= \text{Label} \mid \text{Number} \mid (\text{Number}) \rightarrow \text{Number}$$

Fig. 3. The syntax of types.

4.2 Evaluation rules

An *evaluation judgement* takes the form $s : t \Downarrow s' : t'$, and says that evaluating a term t in a state s yields a term t' in a state s' . A state may be made up of a mapping Γ from names to worksheets, a mapping Δ from cell references to cells and a value ϕ to be used for the history variable. To the left of “ $\Downarrow$ ”, $M[a \mapsto b]$ indicates extension of a mapping, and $M[\dots, a \mapsto b, \dots]$ reveals some part of it. To the right of “ $\Downarrow$ ”, such details are omitted, since any changes can easily be inferred from elsewhere.

Evaluation judgements can be used to write *evaluation rules* for terms. The rule E-BOOK says that evaluating a workbook involves evaluating *only* the visible worksheet (this is the one being “ticked”). See figure 4. The rule E-SHEET says that evaluating a worksheet involves evaluating *all* of its cells (this must be done in *natural order*). See figure 5. There are two evaluation

$$\text{E-BOOK} \frac{\Gamma[g_i \mapsto s_i]_{i=1}^n : s_{\checkmark} \Downarrow \Gamma' : s'_{\checkmark}}{\Gamma : \{\dots, g_{\checkmark} \Rightarrow s_{\checkmark}, \dots\}_{i=1}^n \Downarrow \Gamma' : \{\dots, g_{\checkmark} \Rightarrow s'_{\checkmark}, \dots\}_{i=1}^n}$$

Fig. 4. The evaluation rule for workbooks.

$$\text{E-SHEET} \frac{\Gamma; \Delta[r_i \mapsto c_i]_{i=1}^m : c_i \Downarrow \Gamma'; \Delta' : c'_i}{\Gamma : \langle r_i \Rightarrow c_i \rangle_{i=1}^m \Downarrow \Gamma' : \langle r_i \Rightarrow c'_i \rangle_{i=1}^m, r_i \text{ in natural order}}$$

Fig. 5. The evaluation rule for worksheets.

rules for cells. See figure 6. The rules E-CELL1 and E-CELL2 say that evaluating

$$\begin{aligned} \text{E-CELL1} & \frac{}{\Gamma; \Delta : (v, ?) \Downarrow \Gamma; \Delta : (v, ?)} \\ \text{E-CELL2} & \frac{\Gamma; \Delta; v : f \Downarrow \Gamma'; \Delta' : e}{\Gamma; \Delta : (v, f) \Downarrow \Gamma'; \Delta' : (e, f)} \end{aligned}$$

Fig. 6. The evaluation rules for cells.

a cell is trivial if it has a value, but no formula; otherwise, the formula must be evaluated to give the value. There are five evaluation rules for expressions. See figure 7. The rules E-LAB and E-NUM say that evaluating constants is

$$\begin{aligned} \text{E-LAB} & \frac{}{\Gamma; \Delta; \phi : l \Downarrow \Gamma; \Delta : l} \\ \text{E-NUM} & \frac{}{\Gamma; \Delta; \phi : n \Downarrow \Gamma; \Delta : n} \\ \text{E-REF} & \frac{}{\Gamma; \Delta[\dots, r \mapsto (v, f), \dots]; \phi : r \Downarrow \Gamma; \Delta : v} \\ \text{E-HIST} & \frac{}{\Gamma; \Delta; \phi : \text{last} \Downarrow \Gamma; \Delta : \phi} \\ & \frac{\Gamma; \Delta; \phi : x \Downarrow \Gamma'; \Delta' : y}{\Gamma; \Delta; \phi : g(x) \Downarrow \Gamma''; \Delta'' : e} \\ \text{E-APP} & \frac{\Gamma'[\dots, g \mapsto s, \dots]; s[\mathbf{A1} \mapsto (y, ?)]; \phi : \mathbf{A2} \Downarrow \Gamma''; s'' : e}{\Gamma; \Delta; \phi : g(x) \Downarrow \Gamma''; \Delta'' : e} \end{aligned}$$

Fig. 7. The evaluation rules for expressions.

trivial. The rule E-REF says that a cell reference evaluates to the value of a cell that must itself already have been evaluated because evaluation is done in natural order. The rule E-HIST says that **last** evaluates to the value of the cell whose formula caused it to be evaluated. The rule E-APP says that evaluating an application amounts to evaluating the result cell of the function worksheet when its argument cell contains the argument value.

4.3 Typing rules

A *typing judgement* takes the form $A \vdash t :: T$, and says that under the assumptions A , a term t has type T . The assumptions are made up of a mapping Λ from names to types, and a mapping Ξ from cell references to types.

Typing judgements can be used to write *typing rules* for terms. The rule T-BOOK says that a workbook has (arbitrarily) the type of the visible worksheet. See figure 8. The rule T-SHEET says that a worksheet has a type formed from

$$\text{T-BOOK} \frac{\Lambda[\dots, g_i :: T_i, \dots] \vdash s_{\sqrt{}} :: T_{\sqrt{}}}{\Lambda \vdash \{\dots, g_{\sqrt{}} \Rightarrow s_{\sqrt{}}, \dots\}_{i=1}^n :: T_{\sqrt{}}}$$

Fig. 8. The typing rule for workbooks.

those of the argument and result cells, which must be Numbers. See figure 9.

$$\text{T-SHEET} \frac{\begin{array}{c} \Lambda; \Xi[\mathbf{A1} :: \text{Number}] \vdash c_i :: T_i \\ \Lambda; \Xi[\mathbf{A1} :: \text{Number}, r_i :: T_i] \vdash c_2 :: \text{Number} \end{array}}{\Lambda \vdash \langle \mathbf{A1} \Rightarrow c_1, \mathbf{A2} \Rightarrow c_2, r_i \Rightarrow c_i \rangle_{i=3}^m :: (\text{Number}) \rightarrow \text{Number}}$$

Fig. 9. The typing rule for worksheets.

There are two typing rules for cells. See figure 10. The rules T-CELL1 and T-

$$\begin{array}{c} \text{T-CELL1} \frac{\Lambda; \Xi \vdash v :: T}{\Lambda; \Xi \vdash (v, ?) :: T} \\ \\ \text{T-CELL2} \frac{\Lambda; \Xi \vdash f :: \text{Number}}{\Lambda; \Xi \vdash (v, f) :: \text{Number}} \end{array}$$

Fig. 10. The typing rules for cells.

CELL2 say that a if a cell has value, but no formula, then its type is that of the value; otherwise, it is that of the formula, which must be Number. There are five typing rules for expressions. See figure 11. The rules T-LAB and T-NUM say that constants have the obvious basic types. The rule T-REF says that a cell reference has the type given in the assumptions, and the rule T-HIST says that `last` must have type Number. The rule T-APP says that the application of a worksheet to an argument of type Number has type Number.

5 An example model

Hodgkin and Huxley modelled the current flow through the surface membrane of the nerve fibre of a squid as the sum of the *capacitative*, *sodium*, *potassium* and *leakage* currents [4].

$$\begin{array}{c}
\text{T-LAB} \frac{}{\Lambda; \Xi \vdash l :: \text{Label}} \\
\text{T-NUM} \frac{}{\Lambda; \Xi \vdash n :: \text{Number}} \\
\text{T-REF} \frac{}{\Lambda; \Xi[\dots, r :: T, \dots] \vdash r :: T} \\
\text{T-HIST} \frac{}{\Lambda; \Xi \vdash \text{last} :: \text{Number}} \\
\text{T-APP} \frac{\Lambda; \Xi \vdash x :: \text{Number} \quad \Lambda; \Xi \vdash g :: (\text{Number}) \rightarrow \text{Number}}{\Lambda; \Xi \vdash g(x) :: \text{Number}}
\end{array}$$

Fig. 11. The typing rules for expressions.

5.1 The capacitative current

The capacitative current, I_{cap} , is given by

$$I_{\text{cap}} = C_m \times \frac{dV_m}{dt}$$

where C_m is the membrane capacitance, and V_m the transmembrane potential.

5.2 The sodium current

The sodium current, I_{Na} , is given by

$$I_{\text{Na}} = g_{\text{Na,max}} \times m^3 \times h \times (V_m - E_{\text{Na}})$$

where $g_{\text{Na,max}}$ is the maximum conductance of sodium, and m and h satisfy the equations

$$\frac{dm}{dt} = \alpha_m \times (1 - m) - \beta_m \times m$$

$$\frac{dh}{dt} = \alpha_h \times (1 - h) - \beta_h \times h$$

The rate coefficients were found by curve-fitting to be

$$\alpha_m = \frac{0.1 \times (V_m + 25.0)}{\exp(0.1 \times (V_m + 25.0)) - 1.0}$$

$$\beta_m = 4 \times \exp(V_m/18.0)$$

$$\alpha_h = 0.07 \times \exp(V_m/20.0)$$

$$\beta_h = \frac{0.1}{\exp(0.1 \times (V_m + 30.0)) + 1.0}$$

5.3 The potassium current

The potassium current, I_K , is given by

$$I_K = g_{K,\max} \times n^4 \times (V_m - E_K)$$

where $g_{K,\max}$ is the maximum conductance of potassium, and n satisfies the equation

$$\frac{dn}{dt} = \alpha_n \times (1 - n) - \beta_n \times n$$

The rate coefficients were again found by curve-fitting to be

$$\alpha_n = \frac{0.01 \times (V_m + 10.0)}{\exp(0.1 \times (V_m + 10.0)) - 1.0}$$

$$\beta_n = 0.125 \times \exp(V_m/80.0)$$

5.4 The leakage current

The leakage current, I_L , is given by

$$I_L = g_L \times (V_m - E_L)$$

where E_L is the reversal potential for the leakage current.

5.5 The total current

Combining the equations, the total current across a cell membrane, I , is given by

$$I = C_m \times \frac{dV_m}{dt} + I_{Na} + I_K + I_L$$

This can be rearranged to

$$\frac{dV_m}{dt} = \frac{I - I_{Na} - I_K - I_L}{C_m}$$

6 Simulation with our prototype simulator

In our simulator, the ordinary differential equations of the example model are entered into a workbook. See figure 12. The **run** worksheet drives the simulation. Cells A4 — A7 deal with calculation of the dependent variables **Vm**, **m**, **h** and **n**, and cells A2 and C1 with calculation of the dependent variable, **t**. The remaining cells are used for intermediate calculations, constant values and labels. Other worksheets are used in the calculation of the dependent variables. To save space, we have shown only the **calc_m** worksheet, as **calc_h**, **calc_n** and **calc_vm** are very similar. Simulation is performed by pressing a “recalculate” button and specifying the number of steps (or “ticks”). Our simulator is written from scratch in Haskell using the WxHaskell library.

	A	B	C	D	E
1			5.0e-3		step
2	0.0 = last + C1				t
3					
4	-75.0 = ODE(calc_vm(A2, B9, B10, B11, last), C1)				Vm
5	5.0e-2 = ODE(calc_m(A2, A4, last), C1)				m
6	0.6 = ODE(calc_h(A2, A4, last), C1)				h
7	0.325 = ODE(calc_n(A2, A4, last), C1)				n
8					
9		= C17 * A5 ^ 3.0 * A6 * (A4 - C14)			INa
10		= C18 * A7 ^ 4.0 * (A4 - C15)			IK
11		= C19 * (A4 - C16)			IL
12					
13			-75.0		ER
14			= C13 + 115.0		ENa
15			= C13 - 12.0		EK
16			= C13 + 10.613		EL
17			120.0		gNaMax
18			36.0		gKMax
19			0.3		gL

The run worksheet.

	A	B	C	D	E
1		t			
2		Vm			
3		m			
4	= B6 * (1.0 - A3) - B7 * A3				
5					
6		= 0.1 * (A2 + 25.0) / (EXP(0.1 * (A2 + 25.0)) - 1.0)			
7		= 4.0 * EXP(A2 / 18.0)			

The calc_m worksheet.

Fig. 12. Model entry with our prototype simulator.

7 Simulation with the E-Cell simulator

One of the most advanced cell simulators available is E-Cell, developed by Tomita and his team since 1996 [9]. In the E-Cell simulator, the ordinary differential equations of the example model are entered in a text file. See figure 13. Here, the single stepper object, ODE, is an ODE45Stepper — the

```
Stepper ODE45Stepper( ODE ) { }

System System( / ) {
  StepperID ODE;
  Variable Variable( m ) { Value 5.0E-2; }
  Variable Variable( h ) { Value 0.6; }
  Variable Variable( n ) { Value 0.325; }
  Variable Variable( Vm ) { Value -75.0; }

  Process ExpressionFluxProcess( proc_m ) {
    Expression "( (0.1 * (Vm.Value + 25.0)) /
                  (exp(0.1 * (Vm.Value + 25.0)) - 1.0) ) *
                  (1 - m.Value)
                  - (4 * exp(Vm.Value / 18.0) * m.Value)";

    VariableReferenceList [ m Variable::m 1 ]
                          [ Vm Variable::Vm 0 ];
  }

  # ... similar processes to calculate h, n and Vm
}
```

Fig. 13. Model entry with the E-Cell simulator.

one recommended for solving systems of ordinary differential equations. It calls process objects and determines the next time step interval. The system object, **System**, contains variable and process objects that describe the system of equations. For each variable object, the **Value** property is set to the initial value. For each process object, the **Expression** property is set to an expression that changes the value of a variable, and the **VariableReferenceList** property is set to indicate those variables that it reads ('0'), and those that it reads or writes ('1'). Notice that we have had to break the **Expression** string to fit it on the page. Again to save space, we have shown only the process **proc_m**, as **proc_h**, **proc_n** and **proc_Vm** are similar. Simulation is performed by loading a text file into either a batch or an interactive session monitor. E-Cell is written in C++ and Python.

8 An experimental study

It is essential that cell biologists be able to enter their models into a simulator quickly and accurately. Otherwise, they will either give up on simulation and continue to use pencil-and-paper, or they will generate meaningless simulation results whose cause might be difficult to determine. A small experimental study was therefore conducted into the effectiveness of both our prototype simulator interface and the E-Cell simulator interface.

8.1 *Subjects*

Eight volunteer subjects took part in the study, ranging in age from 20 to 37 years old. Four were BSc Computer Science students, two were MSc Bioinformatics students, one was a Research Assistant in Bioinformatics, and one was a Research Assistant in Neural Networks. All had some experience of spreadsheets and text editors, but none of cell simulators. They were thus considered to have a representative skill-set.

8.2 *Apparatus*

A week in advance of the experiment, the subjects were issued with a four-page manual covering both simulators, and containing an example of how to use them to simulate a model with two ordinary differential equations. This manual was deliberately terse and example-based, reflecting the typical provision of documentation and the availability of examples. The test problem was to enter the four ordinary differential equations for the model given in [12].

8.3 *Procedure*

First, each subject was asked to fill out a short questionnaire, answering questions about themselves and their proficiency with spreadsheets, text editors and cell simulators. Next, they were given a five minute demonstration of the simulators. Finally, the procedure for the study was explained and the test problem was revealed. Afterwards, each subject was invited to comment on their experience.

8.4 *Data collection*

The subjects could choose to use either simulator first. Their choice was recorded, together with their claimed proficiency with spreadsheets and text editors. For our prototype, “begin” time was recorded when the subject started the simulator, and “end” time when they had successfully recalculated their workbook. The number of error dialogue boxes encountered, and the nature of the errors was also recorded. For E-Cell, “begin” time was recorded when the subject started the text editor, and “end” time when they had successfully loaded their file into the session monitor. The number of load

commands issued, and the nature of any errors was also recorded. Afterwards, the workbooks and files were examined and the number of mistakes that they contained was recorded.

8.5 Results

Table 14 and Table 15 show the results for our prototype and for E-Cell.

subject identifier	spreadsheet proficiency	time taken	static errors	dynamic errors
A	5	27	1	0
B	3	31	1	0
C	3	55+	13	1
D	3	31	3	0
E	4	35+	6	0
F	3	36	14	1
G	3	30	3	0
H	1	34	3	0

Fig. 14. Results for our prototype simulator.

subject identifier	text editor proficiency	time taken	static errors	dynamic errors
A	4	36	7	0
B	5	28	8	3
C	5	66+	17	4
D	3	69+	12	4
E	1	30	4	1
F	4	49	10	0
G	4	27	1	0
H	4	35	12	0

Fig. 15. Results for the E-Cell simulator.

In these tables, proficiency is measured from 0 (low) to 5 (high), and the time taken is measured in minutes, with a trailing “+” indicating that the

subject gave up. Static errors are counted as the number of error dialogue boxes or load commands, and dynamic errors as the number of mistakes.

8.6 Discussion

Regardless of their claimed proficiency with spreadsheets and text editors, all but one of the subjects chose to use E-Cell first, and their times taken to enter the test model with both simulators were broadly similar. No clear preference was expressed for either.

In the case of our simulator, two of the static errors were caused by accidental cycles between cells, and four by dangling cell references. These were eventually fixed, although our curt error messages of the form “A1: cycle exists” and “A1: dangling reference” proved difficult to comprehend. The remaining static errors were type (in fact, arity) errors. After some thought, most subjects fixed these, but the two who gave up did so because they had entirely misunderstood the worksheet argument-passing convention, and found error messages of the form “worksheet does not have type (Number,Number,Number,Number) -> Number” incomprehensible. Both of the dynamic errors were mistyped constants.

In the case of E-Cell, the majority of static errors were caused by missing parentheses, semicolons and the like. These were fixed after a few repeats of the the edit-load cycle. Of the two subjects who gave up, one did so because they declared their constants outside of a process rather than inside it, leading to an incomprehensible error message about unknown property slots, and the other did so because they used identifiers instead of `ExpressionFluxProcess`, leading to an incomprehensible message about loading shared object files. All of the dynamic errors were caused by misunderstandings about the role of `VariableReferenceList` integers.

9 Related and future work

A spreadsheet interface was used in an early version of the E-Cell simulator [9] and is still being used in the JigCell simulator [11]. In both cases, the spreadsheet describes chemical reactions, with rows used for objects (such as reactants and reactions) and columns for properties (such as names and types). The advantage of a constrained spreadsheet like this is that clear rules can be given about what should go in which cells, and these rules can be checked. The disadvantage is that these additional rules must be learned, and so much of the attractive simplicity of a free-form spreadsheet is lost. Actual experience with the spreadsheet interface has been mixed. On the one hand, it was abandoned in E-Cell, and so presumably was not a success. On the other hand, it continues to be used in JigCell, with encouraging results.

During our study, several subjects complained that our prototype was missing copy-and-paste, which they especially wanted to copy a table of all con-

stants onto all worksheets. They could also often be seen pointing at the screen, or heard muttering as they worked out which cell references to use because, although able to comment on the role of cells by placing labels nearby, they were unable to refer to them by those labels. Some subjects even engaged in rapid switching between worksheets to check function arities because only one worksheet is visible at a time. None of these problems would have arisen with Excel, which in future we plan to use as the interface to our simulator.

Our typing rules are in the tradition of functional programming, largely because the author is a former functional programmer. As our study has shown, though, they are not well suited to spreadsheets. Recently, there has been a growing interest in checking the consistency of spreadsheets by reasoning about their *units* [1,2,3]. In principle, any value declares a unit (for example, a label “Month” declares a unit Month). In practice, it is those values serving as column or row headers which provide the declarations that can serve as the basis for *unit inference* (for example, a label “October” in a column with a header label “Month” has the unit Month[October]). As Erwig and Burnett remark, unit inference provides a good way to make use of the “implicitly explicit” information already present for the purposes of documentation [3]. In future, we plan to build on this work to develop evaluation and typing rules to ensure that *types* (such as numbers and labels), *dimensions* (such as litres and volts) and *units* (such as reactants and catalysts) are used consistently.

Of course, other cell simulators exist with textual (for example, Jarnac [7]), diagrammatic (for example, GEPASI [5]) and graphical (for example, Virtual Cell [8]) user interfaces. Once we have improved our simulator as outlined, we plan to repeat our study, including simulators like these too.

10 Conclusions

In this paper, we have described our prototype cell simulator and reported on our early experience with it. The simulator interface is a spreadsheet that incorporates a new approach to functions, a history mechanism and a built-in function for solving ordinary differential equations. A small experimental study showed that it was not as effective as it might be. In future, we plan to redesign our prototype, to develop more suitable typing rules, and to undertake another study.

Acknowledgements

We are grateful to the anonymous referees for some useful comments and suggestions.

References

- [1] Ahmad, Y., T. Antoniu, S. Goldwater and S. Krishnamurthi, *A type sysetm for statically detecting spreadsheet errors*, in: *IEEE International Conference on Automated Software Engineering* (2003), pp. 174–183.
- [2] Antoniu, T., P. A. Steckler, S. Krishnamurthi, E. Neuwirth and M. Felleisen, *Validating the unit correctness of spreadsheet programs*, in: *Proceedings of the International Conference on Software Engineering* (2004), pp. 439–448.
- [3] Erwig, M. and M. Burnett, *Adding apples and oranges*, in: *Proceedings of the International Symposium on Practical Aspects of Declarative Languages* (2002), pp. 173–191, INCS 2257.
- [4] Hodgkin, A. L. and A. F. Huxley, *A quantitative description of membrane current and its application to conduction and excitation in nerve*, *Bulletin of Mathematical Biology* **52** (1990), pp. 25–71.
- [5] Mendes, P., *Biochemistry by numbers*, *Trends in Biochemical Sciences* **22** (1997), pp. 361–363.
- [6] Peyton Jones, S., A. Blackwell and M. Burnett, *A user-centred approach to functions in Excel*, in: *Proceedings of the International Conference on Functional Programming* (2003), pp. 165–176.
- [7] Sauro, H. M., *Jarnac: A system for interactive metabolic analysis*, in: *Proceedings of the International Meeting on BioThermoKinetics* (2000), pp. 221–228.
- [8] Schaff, J. and L. M. Loew, *The virtual cell*, in: *Pacific Symposium on Biocomputing*, 1999, pp. 228–239.
- [9] Takahashi, K., N. Ishikawa, Y. Sadamoto, H. Sasamoto, S. Ohta, A. Shiozawa, F. Miyoshi, Y. Naito, Y. Nakayama and M. Tomita, *E-Cell 2: Multi-platform E-Cell simulation system*, *Bioinformatics* **19** (2003), pp. 1727–1729.
- [10] Tomita, M., *Whole-cell simulation: A grand challenge of the 21st century*, *Trends in Biotechnology* **19** (2001), pp. 205–210.
- [11] Vass, M., C. A. Shaffer, J. J. Tyson, N. Ramakrishnan and L. T. Watson, *The JigCell model builder: A tool for modeling intra-cellular regulatory networks* (2002).
- [12] Wodarz, D. and V. A. A. Jansen, *A dynamical perspective of CTL cross-priming and regulation implications for cancer immunology*, *Immunology Letters* **86** (2003), pp. 213–227.
- [13] Wray, S. C. and J. Fairbairn, *Non-strict languages — programming and implementation*, *The Computer Journal* **32** (1989), pp. 142–151.