End-user development via sheet-defined functions

Peter Sestoft[*]

Jonas Druedahl Rask

Simon Eikeland Timmermann

ABSTRACT

We have developed an implementation of *sheet-defined functions*, a mechanism that allows spreadsheet users to define their own functions, using only spreadsheet concepts such as cells, formulas and references, and no external programming languages. This position paper presents the motivation and vision of this work, describes the features of our prototype implementation, and outlines future work.

Keywords

Spreadsheets, end-user development, functional programming

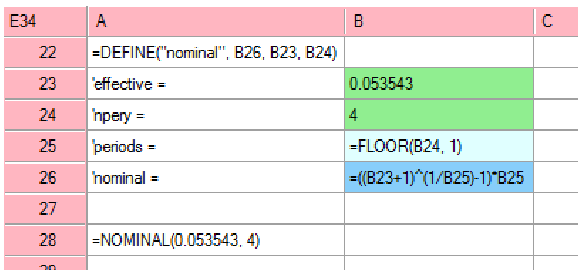
1. INTRODUCTION

Spreadsheet programs such as Microsoft Excel, OpenOffice Calc, Gnumeric and Google Docs are used by millions of people to develop and maintain complex models in finance, science, engineering and administration. Yet Peyton Jones, Burnett and Blackwell [9] observed that spreadsheet programs lack even the most basic abstraction facility — a way to encapsulate an expression as a reusable function — and proposed a design for such a mechanism.

We have implemented this idea in the form of *sheet-defined functions*. A user may define a function `F` simply by declaring which (input) cells will hold `F`'s formal parameters and which (output) cell will compute `F`'s result, as a function of the input cells. The function definition may involve arbitrary additional cells, spreadsheet formulas, calls to built-in functions, and calls to other sheet-defined functions. Figure 1 shows an example. In the example, values are referred to by cell (such as B25). A mechanism that allows for symbolic names (such as "periods") instead could be added, but Nardi speculates that end user developers would not necessarily find that better [7, page 44].

Augustsson et al. from Standard Chartered Bank provide further support for the utility of such abstraction mechanisms, saying about the traditional combination of Excel and externally defined functions that "change control, static type checking, abstraction and reuse are almost completely lacking" [1].

[*]sestoft@itu.dk, IT University of Copenhagen, Denmark



E34	A	B	C
22	=DEFINE("nominal", B26, B23, B24)		
23	'effective =	0.053543	
24	'npery =	4	
25	'periods =	=FLOOR(B24, 1)	
26	'nominal =	=((B23+1)^(1/B25)-1)*B25	
27			
28	=NOMINAL(0.053543, 4)		

Figure 1: A sheet-defined function implementing Excel's `NOMINAL` built-in. Cells B23 and B24 are input cells, cell B26 is the output cell, and B25 holds an intermediate result. The call to `DEFINE` in cell A22 creates the function. Cell A28 contains a call to the defined function. It takes around 200 ns to execute it, of which 80 ns is due to exponentiation (^). As shown in cell A28, a sheet-defined function is called just like a built-in or VBA function.

2. THE VISION

The ultimate goal of this work is to allow spreadsheet users themselves to develop and evolve libraries of user-defined functions to support sophisticated spreadsheet models. Defining a function requires only well-known spreadsheet concepts such as cell, cell reference and function, and no external programming languages. Therefore experimentation and adaptation of user-defined functions remain under the control of the spreadsheet users and domain experts, who need not wait for an IT department to understand, describe, implement and test the desired changes.

Any spreadsheet computation can be turned into a sheet-defined function. This ensures conceptual and notational simplicity. Moreover, it means that new user-defined functions may arise by refactoring of a spreadsheet model as it evolves. As a spreadsheet model becomes more refined and complex, it may be observed that the same cluster of formulas appears again and again. Such a cluster of formulas may then be encapsulated in a sheet-defined function, and each formula cluster replaced by a call to that function. This both simplifies the spreadsheet model and improves its

Table 1: Time to compute the cumulative distribution function of the normal distribution $N(0,1)$.

Implementation	Time/call (ns)
Sheet-defined function	118
C#	47
C (gcc 4.2.1 -O3)	51
Excel 2007 VBA function	1925
Excel 2007 built-in `NORMSDIST`	993

maintainability, because a bug-fix or other improvement to the function will automatically affect all its uses, unlike the traditional situation when there are multiple copies of the same cluster of formulas.

Sheet-defined functions may be shared with other users in the same department or application domain, without preventing them from making their own improvements — because the domain knowledge is not locked into the notation of a "real" programming language, but one that presumably is familiar to users and that they are (more) comfortable experimenting with.

Sheet-defined functions support end-user "tinkering" to develop models and workflows that are appropriate within their application domain [7]. Clearly not all spreadsheet users will be equally competent developers of sheet-defined functions, and clearly not all software should be developed in this way. However, judging from the huge popularity of spreadsheets within banks, finance, management, science and engineering, the immediate response and the user control offered by spreadsheets are attractive features. Also, from anecdotal evidence, structured use of spreadsheets is a flexible, fast and cheap alternative to "big bang" professional IT projects.

3. THE FUNCALC PROTOTYPE

We have created a prototype implementation of sheet-defined functions, called Funcalc. The implementation is written in C#, is quite compact (12,000 lines of code) and compiles sheet-defined functions to .NET bytecode [3] at run-time. As shown by Table 1 execution efficiency is very good; this is due both to local optimizations performed by our function compiler and to Microsoft's considerable engineering effort in the underlying .NET just-in-time compiler.

Funcalc features include:

- a "normal" interpretive spreadsheet implementation;
- a compiled implementation of sheet-defined functions;
- recursive functions and higher-order functions;
- functions can accept and return array values in addition to numbers and string;
- automatic specialization, or partial evaluation [12];
- facilities for benchmarking sheet-defined functions.

Because Funcalc supports higher-order functions, the value contained in a cell, say A42, may be a function value. This value may be called as `APPLY(A42,0.053543,4)` using built-in function `APPLY`.

Function values are built by applying a sheet-defined function to only some of its arguments, the absent arguments being given as `NA()`; the resulting function value will display as `NOMINAL(#NA,4)` or similar.

Such a function value may be specialized, or partially evaluated, with respect to its available (non-`#NA`) arguments. The result is a new function value with the same behavior but potentially better performance because the available argument values have been inlined and loops unrolled in the underlying bytecode. For more information, see [5] and [12]. Specialization provides some amount of incremental computation and memoization, and we do not currently have other general mechanisms for these purposes.

A forthcoming book [13] gives many more details of the implementation, more examples of sheet-defined functions, and a manual for Funcalc. A previously published paper [15] presents a case study of reimplementing Excel's built-in financial functions as sheet-defined functions.

A comprehensive list of US spreadsheet patents is given in a forthcoming report [14].

Table 2: Time to call a square root function; includes recalculation time.

Calling	Time/call (ns)
Sheet-defined function from Funcalc	400
Excel built-in from Excel	160
.NET function from Excel/Excel-DNA	4,900
VBA function from Excel	12,000

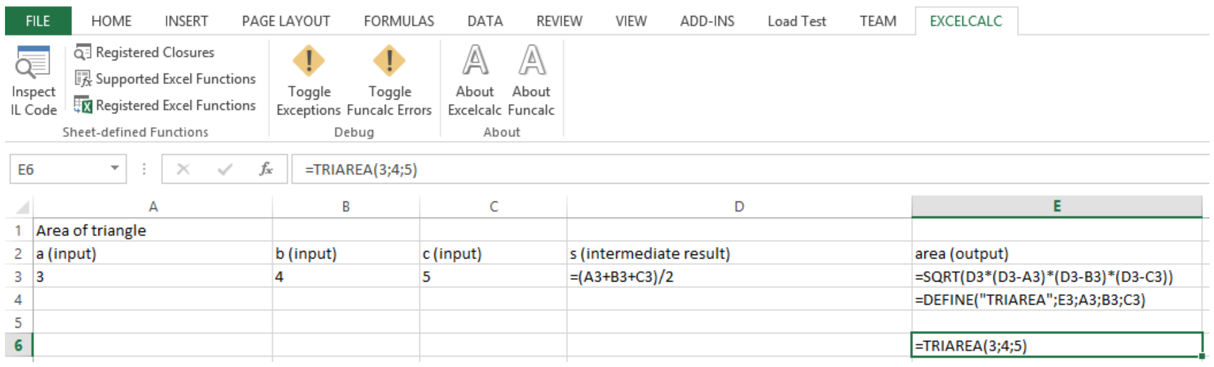
4. INTEGRATION WITH EXCEL

In ongoing work [10] we integrate sheet-defined functions with the widely used Microsoft Excel spreadsheet program, rather than our Funcalc prototype, as illustrated in Figures 2 and 3. This enables large-scale experimentation with sheet-defined functions because they can be defined in a context that is familiar to spreadsheet users and provides charting, auditing, and so on.

The main downside is that calling a sheet-defined function from Excel is much slower than from the Funcalc implementation (yet apparently faster than calling a VBA function); see Table 2. However, the sheet-defined function itself will execute at the same speed as in Funcalc. This work uses the Excel-DNA runtime bridge between Excel and .NET [4].

5. FUTURE WORK

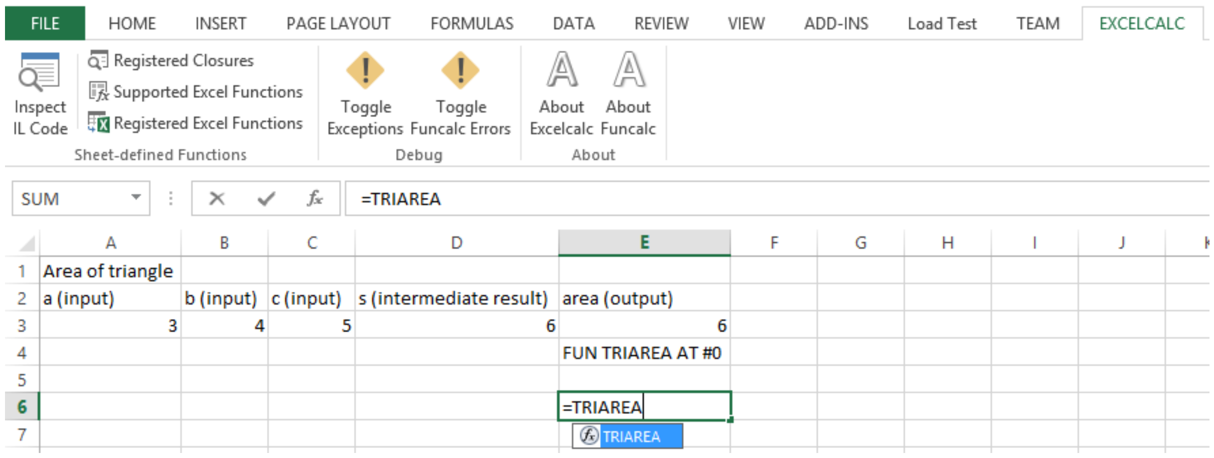
So far we have focused mostly on functionality and good performance. We emphasize performance because we want sheet-defined functions to replace not only user-defined functions written in VBA, C++ and other external languages, but to replace built-in functions also. Domain experts in finance, statistics and other areas of rather general interest should be able to develop well-performing high-quality functions themselves and not have to rely on Microsoft or other vendors to do so.



E6 =TRIAREA(3;4;5)

	A	B	C	D	E
1	Area of triangle				
2	a (input)	b (input)	c (input)	s (intermediate result)	area (output)
3	3	4	5	=(A3+B3+C3)/2	=SQRT(D3*(D3-A3)*(D3-B3)*(D3-C3))
4					=DEFINE("TRIAREA";E3;A3;B3;C3)
5					
6					=TRIAREA(3;4;5)

Figure 2: Funcalc as Excel plug-in, showing formulas of sheet-defined function `TRIAREA` **with input cells A3, B3 and C3, intermediate cell D3, and output cell E3. The call to** `DEFINE` **in cell E4 creates the function. Through the new "Excelcalc" menu one can interact with the underlying Funcalc implementation and the Excel-Funcalc bridge (mostly for development purposes).**



SUM =TRIAREA

	A	B	C	D	E	F	G	H	I	J
1	Area of triangle									
2	a (input)	b (input)	c (input)	s (intermediate result)	area (output)					
3	3	4	5	6	6					
4					FUN TRIAREA AT #0					
5										
6					=TRIAREA					
7										

Figure 3: Same sheet as in Figure 2, here showing values rather than formulas. Note the editing in progress of a call to sheet-defined function `TRIAREA` **in cell E6.**

However, a well-performing implementation of sheet-defined functions is just the beginning: one should investigate additional infrastructure and practices to support their use. For instance, how can we extend the natural collaboration around spreadsheet development [8] in a community of users to cover also libraries of sheet-defined functions; how can we support versioning and merging of such libraries in a way that does not preclude individual users' tinkering and experimentation; how can we support systematic testing; and so on.

Our concept of sheet-defined functions should be subjected to a systematic usability study; the study conducted by Peyton-Jones, Blackwell and Burnett [9] assumed that functions could not be recursive, whereas ours can.

Finally, sheet-defined functions lend themselves well to parallelization, because they are pure (yet strict, an unusual combination) so that computations can be reordered and performed speculatively, and often exhibit considerable explicit parallelism. In fact, they resemble dataflow languages such as Sisal [6]. Presumably some of the 1980es techniques used to schedule dataflow languages [11] could be used to perform spreadsheet computations efficiently on modern multicore machines. The result might be "supercomputing for the masses", realizing Chandy's 1984 vision [2].

6. CONCLUSION

We have presented a prototype implementation of sheet-defined functions and outlined some future work. Our hope is that such functionality will become available in widely used spreadsheet programs, or via a full-featured version of the plugin described in Section 4, and will enable spreadsheet users to develop their own computational models into reusable function libraries, without loss of computational efficiency and without handing over control to remote IT departments or software contractors. Moreover, there seems to be a technological opportunity to harness the power of multicore machines through spreadsheet programming.

7. REFERENCES

[1] L. Augustsson, H. Mansell, and G. Sittampalam. Paradise: A two-stage DSL embedded in Haskell. In *International Conference on Functional Programming (ICFP'08)*, pages 225–228. ACM, September 2008.

[2] M. Chandy. Concurrent programming for the masses. (PODC 1984 invited address). In *Principles of Distributed Computing 1985*, pages 1–12. ACM, 1985.

[3] Ecma TC39 TG3. *Common Language Infrastructure (CLI). Standard ECMA-335.* Ecma International, sixth edition, June 2012.

[4] Excel DNA Project. Homepage. At http://exceldna.codeplex.com/ on 28 February 2014.

[5] N. D. Jones, C. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation.* Prentice Hall, 1993. At http://www.itu.dk/people/sestoft/pebook/pebook.html on 9 June 2013.

[6] J. McGraw et al. Sisal. Streams and iteration in a single assignment language. Language reference manual, version 1.2. Technical report, Lawrence Livermore National Labs, March 1985.

[7] B. A. Nardi *A small matter of programming. Perspectives on end user programming.* MIT Press, 1993.

[8] B. A. Nardi and J. R. Miller Twinkling lights and nested loops: distributed problem solving and spreadsheet development. *International Journal of Man-Machine Studies*, 34:161–184, 1991.

[9] S. Peyton Jones, A. Blackwell, and M. Burnett. A user-centred approach to functions in Excel. In *ICFP '03: Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming*, pages 165–176. ACM, 2003.

[10] J. D. Rask and S. E. Timmermann. Integration of sheet-defined functions in Excel using C#. Master's thesis, IT University of Copenhagen, 2014. (Expected June 2014).

[11] V. Sarkar and J. Hennessy. Compile-time partitioning and scheduling of parallel programs. In *ACM SIGPLAN '86 Symposium on Compiler Construction*, pages 17–26, June 1986.

[12] P. Sestoft. Online partial evaluation of sheet-defined functions. In A. Banerjee, O. Danvy, K. Doh, and J. Hatcliff, editors, *Semantics, Abstract Interpretation, and Reasoning about Programs*, volume 129 of *Electronic Proceedings in Theoretical Computer Science*, pages 136–160, 2013.

[13] P. Sestoft. *Spreadsheet Implementation Technology. Basics and Extensions.* MIT Press, 2014. ISBN 978-0-262-52664-7. (Expected August 2014). 313 pages.

[14] P. Sestoft. Spreadsheet patents. Technical Report ITU-TR-2014-178, IT University of Copenhagen, 2014. ISBN 978-87-7949-317-9. (To appear).

[15] P. Sestoft and J. Z. Sørensen. Sheet-defined functions: implementation and initial evaluation. In Y. Dittrich et al., editors, *International Symposium on End-User Development, June 2013*, volume 7897 of *Lecture Notes in Computer Science*, pages 88–103, 2013.