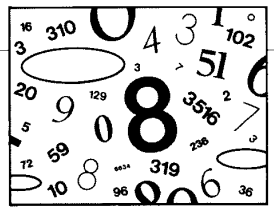

A 3D Spreadsheet Based on Intensional Logic

Weichang Du and **William W. Wadge**, *University of Victoria*

Basing spreadsheets on implicit relations among cells and adding a temporal dimension leads to a powerful problem-solving tool.

Originally devised as calculation aids with simultaneous update capabilities, electronic spreadsheets have developed into a user-friendly problem-solving tool suitable for many applications. However, most commercial spreadsheets have been designed for business applications and for use by computer novices. For such users, a good interface is certainly one of the most important considerations in spreadsheet design. But, while these spreadsheets have good interfaces, they are not as powerful as high-level programming languages and thus usually have relatively low calculating ability and poor problem-solving power. This limits the potential of the conventional spreadsheet as a problem-solving paradigm for a wide variety of applications.

We propose a spreadsheet design based on *intensional* logic — a type of logic concerned with assertions and other expressions whose meaning depends on an implicit context. Our intensional spreadsheet is a declarative spreadsheet with a simple formal semantics: intensional semantics. We thus expect it to be suitable for a wide range of applications and for various users, including those who have had conventional programming experience.

The most important feature of this spreadsheet design is that it considers the whole spreadsheet to be a single entity that varies in three dimensions — two spatial and one temporal. The spreadsheet is denoted by a single variable, called the spreadsheet variable, that you can use in expressions just like any other variable.

This variable is not, however, defined by a single equation (as in conventional functional languages). Instead, it is defined by a whole family of expressions, one for each cell in the spreadsheet plane. In other words, there is one defining expression for each spatial point determined by a pair of coordinates. The value of the spreadsheet variable at a particular cell is the value, at that point, of the defin-

0740-7459/90/0500/0078/$01.00 © 1990 IEEE

ing expression associated with that cell. This is true at all moments in time.

The defining expressions may be constants, but they generally will contain occurrences of the spreadsheet variable. Also, they may, and usually will, use intensional (spatial and temporal) operators so the value of the spreadsheet variable in a given cell at a given moment in time can be defined in terms of its value in other cells at other moments in time.

Intensional spreadsheet

Our intensional 3D spreadsheet, like a conventional spreadsheet, is a display-oriented calculation tool. The main part of the spreadsheet on the screen is a group of cells arranged into rows and columns. Figure 1 shows what a part of the spreadsheet looks like on screen. For convenience, our spreadsheet has cells in the full plane — all four quadrants. This extension is simple, given our approach, and saves the user having to translate into a single quadrant problems that naturally involve negative spatial coordinates.

Cells are named using a simple system of directional indicators: "D" for down, "U" for up, "L" for left, and "R" for right. For example, a cell at coordinate U2L2 lies in the second quadrant and a cell at U1R1 lies in the first quadrant. The coordinates are relative to the cell at the origin, where the horizontal and vertical axes intersect.

Design principle. When users program the spreadsheet, they can think of the whole spreadsheet as an entity denoted by a spreadsheet variable, instead of as a group of cells. The spreadsheet variable, like temperature in the example in the box on p. 82, is an intensional variable that may have different values in different contexts. In this 3D spreadsheet design, "different contexts" means different points in space and time. The spreadsheet variable varies in three dimensions (two spatial and one temporal). The value of the spreadsheet variable at one point in space and time may depend on, and differ from, its values at other points in space and time.

Although we think of the spreadsheet variable as a single entity, we define it piecewise by associating defining expressions with the cells in the plane.

You define the spreadsheet variable by associating defining expressions with some individual cells and, possibly, by giving a global default definition. The definition (or formula) of a cell at a particular spatial point defines the spreadsheet variable at that cell for all moments in time:

In conventional spreadsheets, cell definitions refer to other cells explicitly. But in the intensional spreadsheet, the definition of the spreadsheet variable at a cell refers to others through intensional operations.

When the value of the spreadsheet variable is needed at a cell, the definition at that cell will be evaluated at the given moment.

In conventional spreadsheets, cell definitions refer to other cells explicitly. But in the intensional spreadsheet, the definition of the spreadsheet variable at a cell refers to others through intensional operations. An intensional operation in a definition switches the current context (here, a cell at a moment), which lets it refer to values of the spreadsheet in other contexts.

The language used to write the spreadsheet definitions is Plane Lucid,[1] an extension of the Lucid dataflow language.[2] Plane Lucid lets values of expressions vary in space as well as in time; it provides spatial and temporal operators to combine values from different contexts.

Spatial programming. The spreadsheet variable is defined over the spreadsheet plane, which has two spatial dimensions. The most important spatial operators are Left, Right, Up, and Down.

The unary operator Right shifts the context one point to the right in the spreadsheet plane. Generally speaking, the value of the expression Right E in a cell definition, where E is an arbitrary expression, is the value E would have in the cell immediately to the right. For example, if S is the spreadsheet variable and a cell is defined as Right S, the result of evaluating the cell is the value of the cell that is on the immediate right. You can apply Right to any expression, not just one consisting of the spreadsheet variable. This expression may in turn contain other intensional operations.

The operations Left, Up, and Down are similarly defined.

The following example shows how you can use the spatial operators to define a spreadsheet. Consider a gasoline company that has many stations in different areas. Each station sells three kinds of gas — regular, unleaded, and super unleaded — and has two kinds of services — self-service and full-service. Suppose that each station may have a different price for regular gas, but that the price for unleaded gas is always 1 percent more than regular and the price for super unleaded gas is always 2.5 percent more than regular. Further suppose that full service costs two cents more per liter than self-service. Figure 2 describes a spreadsheet for calculating gas prices for this company.

The relative references to other cells in conventional spreadsheets are similar to expressions with intensional operators.

Name	L3	L2	L1	0	R1	R2	R3
U3							
U2							
U1							
0							
D1							
D2							
D3							

Time =

Figure 1. A window of the intensional spreadsheet. "Name" is any user-defined name.

G	0	R1	R2	R3	R4
0			'unleaded ($/L)'	'regular ($/L)'	'super ($/L)'
D1	'station 1'	'self'	1.01 * right G	0.450	1.025 * left G
D2		'full'	0.02 + up G	0.02 + up G	0.02 + up G
D3					
D4	'station 2'	'self'	1.01 * right G	0.460	1.025 * left G
D5		'full'	0.02 + up G	0.02 + up G	0.02 + up G
D6					
D7	'station 3'	'self'	1.01 * right G	0.470	1.025 * left G
D8		'full'	0.02 + up G	0.02 + up G	0.02 + up G

Figure 2. A spreadsheet for gas prices.

For example, the operation Right *G* is similar to the reference RC[+1] in the commercially available Microsoft Multiplan spreadsheet because they both refer to the value of the cell on the right.

But there is a significant difference between these two approaches. The former is the result of the intensional operator being applied to the spreadsheet variable, while the latter is a reference to an individual cell. You can apply the intensional operator to a more complicated expression that may include other intensional operations; for example Left (*S*–Prev Sum (*S***S*,10)). In a conventional spreadsheet, however, a relative reference accesses only the value of a particular cell.

While these spatial operators switch local or neighboring contexts, sometimes global context switching is also necessary. For example, the evaluation of some cells may require values at some absolute cells instead of at cells relative to the current cell. To handle this, we introduced two spatial operators: Side and Edge. The operation Side *E* always returns the value of the expression *E* evaluated at the cell with the horizontal coordinate 0 and the same vertical coordinate. Similarly, the operation Edge *E* always returns the value of *E* evaluated at the cell with the vertical coordinate 0 and the same horizontal coordinate.

For example, Figure 3 is a spreadsheet to calculate the outer product of a four-element column vector $U = [3\ 5\ 7\ 9]$ and a three-element row vector $V^T = [2\ 4\ 6]$. The definition of the outer product of an n-element column vector U and an m-element row vector V^T is an $n \times m$ matrix C where $c_{ij} = u_i * v_j$.

In the spreadsheet in Figure 3, *C* is the spreadsheet variable, the operation Side *C* returns the column vector *U*, and the operation Edge *C* returns the row vector *V*. Therefore, the definition Side *C* * Edge *C* in a cell defines the value of the cell as the product of the element of *U* at the same row and the element of *V* at the same column.

Temporal programming. The addition of a temporal dimension to the spreadsheet is especially useful when you consider an object varying not only in space but also in time. If a user can do with a 2D spreadsheet what he can do on a piece of paper, a user can do with this 3D intensional spreadsheet whatever he can do on many, even infinitely many, pieces of paper that have some temporal relations among them.

In principle, the spreadsheet represented by the spreadsheet variable varies in a 3D space, which we call the spreadsheet space. Spreadsheet space consists of infinitely many spreadsheet planes that themselves are also infinite. At any time, the screen can show only a window of a particular plane of the spreadsheet space. The value of a spreadsheet plane is actually a field, which is the matrix of all values of the spreadsheet variable at all cells at a particular time.

There are at least two ways to think of a variable that varies in space and time. You can think of each cell as a value that varies in time so the spreadsheet as a whole is a field of time-varying entities. Or you can think of the values at each point in time as constituting a field of values so the spreadsheet as a whole is a time-varying field of values. One analogy is the world's weather, which varies from place to place and changes every day.

When the spreadsheet variable varies in time, the cell definition is considered to be the definition of the spreadsheet variable at that cell for the whole temporal dimension.

We provided temporal operators to define the spreadsheet intensionally in the temporal dimension. The temporal operators Next, Prev (for "previous"), and First are analogous to the spatial operators Right, Left, and Side in the horizontal dimension and to Up, Down, and Edge in the vertical dimension.

C	0	R1	R2	R3
0	'U/V'	2	4	6
D1	3	side C * edge C	side C * edge C	side C * edge C
D2	5	side C * edge C	side C * edge C	side C * edge C
D3	7	side C * edge C	side C * edge C	side C * edge C
D4	9	side C * edge C	side C * edge C	side C * edge C

Figure 3. A spreadsheet program calculating the outer product of two vectors.

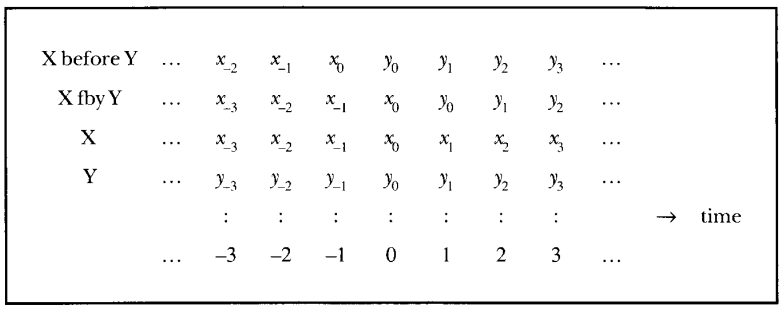

Figure 4. The behavior of the global temporal operators Fby and Before.

To define an expression that may vary in the whole temporal dimension, we provided two other global intensional temporal operators: Fby (for "followed by") and Before.

The binary temporal operator Fby combines two operations, the Prev operation and the identity operation (which does not change context). The value of expression *X* Fby *Y* at a given time and a given spatial position is the value that *Y* had at the previous moment in time at the same spatial position if the given time is positive; otherwise, it is the value *X* had at the same time and spatial position.

Similarly, the global temporal operator Before combines the temporal operation Next and the identity operation. Figure 4 illustrates the behavior of Fby and Before operations. In the diagram, x_i and y_i are the values of expression *X* and *Y* at time *i*, respectively; x_i or y_i may be two (spatial) objects instead of scalar values.

As a simple example, suppose you want a cell to act as a counter so the value of the cell is, at nonnegative times, the time index. The definition 0 Fby *S*+1 does this. It says that the value of the cell at time 0 or earlier is 0; otherwise, it is 1 plus the value of the cell at the previous time. The slightly more sophisticated definition *S*–1 Before (0 Fby *S*+1) specifies the cell value as equal to the time index even at negative times.

You can of course mix temporal and spatial operators. As a simple example, the definition 0 Fby *S* + Up *S* specifies that the value of the cell in which it appears is the sum of the values of the cell above, from time 0 up to but not including the current time. In other words, the cell contains a running total of the values that have appeared above.

Reconsider the gas-price example from the previous section. Suppose that the spreadsheet defined in Figure 2 gives the gas prices at the beginning of the year for that company. Later on, because of inflation and other reasons, the cost of gas will increase by some amount; say, for each month, the price of gas increases 1 percent for regular, 1.2 percent for unleaded, and 1.5 percent for super unleaded.

To solve this problem, you need not change the prices every month; instead you just need to define every cell representing the price of a kind of gas as (initial_price) Fby *G* * (increase_rate) for each station. When this expression is evaluated after initial time 0 (or the first month), the second operand ("*G* * (increase_rate)") will be evaluated at the previous time (month) at the same spatial position — the value of the spreadsheet variable *G* at that cell (or the price of gas) at the current time will be the value of *G* at previous time increased by the increase rate. Thus, for every month, the spreadsheet can automatically calculate new prices.

The spreadsheet in Figure 5a is the redefined spreadsheet. In this spreadsheet program, each plane (associated with a temporal point) shows the gas prices in the corresponding month. Figures 5b and 5c show the prices for the first two months (at times 0 and 1).

User-defined functions. In conventional spreadsheets, the facilities for defining functions are usually provided only to developers, not users. The developers can define various functions to meet special demands. However, in the intensional spreadsheet, users can define various functions just as in high-level language programs. A user-defined function is defined globally, so it can be called in definitions of any cells of the spreadsheet variable.

The definitions take the form of equations. The equation's left side consists of the function name and formal parameters, while the right side is the defining expression. This is the style used in many functional languages. Recursion is allowed.

The function may be purely arithmetic, such as the factorial function defined as

fac(*n*) = if n<1 then 1 else n*f(n–1) fi;

When the factorial function is applied to argument *A*, which varies in space and time, it is applied to each value in each cell at each moment in time. The value of Fac(*A*) at a given space-time point is the factorial of the value of *A* at the same space-time point; Fac is said to work "pointwise."

The definitions can also use spatial or temporal operators, so users can define their own intensional operators. This is probably the single most important advantage that the intensional spreadsheet has over conventional ones. Unfortunately, the user-defined functions must be first-order — no functions may appear as parameters. Our demand-driven scheme

Intensional logic

Intensional logic is concerned with assertions and other expressions whose meaning depends on an implicit context. This type of logic was originally developed to help understand natural languages. For example, consider this expression about the local weather forecast: "five degrees more than yesterday's temperature in the eastern neighboring province." This expression obviously denotes a numerical value. This value clearly depends on a numerical quantity called "temperature." It also depends on when and where it was said, even though there is no explicit reference to either of the temporal or spatial parameters.

Although the value of the expression depends on that of the temperature, you cannot obtain this value by performing purely arithmetic operations on today's temperature. This verbal expression corresponds to a mathematical expression of the form $e(y(t))+5$, where t is the temperature and $y(t)$ is the value that t had yesterday and $e(y(t))$ is the value that $y(t)$ has in the province to the east. But it is obvious that there are no numerical functions y or e that make the value of the mathematical expression correspond to that of the verbal expression.

Consider an example: What could $y(95)$ be? It would have to be the temperature yesterday given only that today's temperature is 95. If you try to treat operators like y and e as mathematical functions, you would soon encounter some paradoxes. For example, suppose that you play golf every day. Let t stand for temperature, as before, and let g be your golf score. Suppose that today they were equal. In symbols, $t=g$. But then the basic properties of equality let you conclude that $y(t)=y(g)$, which asserts that yesterday's temperature is the same as yesterday's golf score — and this may not be true (indeed, why should it be?).

Clearly, these values are context-sensitive and thus must be handled in context. A direct and natural way to formally capture these context-sensitive values is based on the distinction between the *extension* and the *intension* of expressions. An expression's extension is the value in a given context. In the above example, the temperature is 95, and 95 is its extension. A natural-language expression can obviously have different extensions in different contexts. The intension gathers all these extensions and captures how the extensions depend on their contexts. In other words, the intension is the function that assigns to each context the value of the expression in that context.

The paradoxes disappear once you realize that the intension is the true meaning of a natural language. In the temperature example, the intension of temperature is essentially a table giving the temperature of each province on each day; part of the table might look like that shown in Table A. The intension of "five degrees more than yesterday's temperature in the eastern neighboring province" is a similar table, which might look in part like that in Table B.

The meaning of the expression does depend on the meaning of temperature, if you take the intensions to be these meanings. In fact, you can obtain the intension of the temperature from the expression by adding five to all the temperatures in the first table, advancing all the dates in the leftmost column by one day, and shifting all the provinces in the top row by one position to the east.

You can even find mathematically respectable functions y and e that accurately capture the meaning of this phrase. The function y, which maps intensions to intensions, simply increases all the dates by one day; the function e, which also maps intensions to intensions, simply shifts all the provinces to the east by one position.

These functions are respectable in the sense that the arguments determine the results, which before did not seem to be true. You can also spot the flaw in the paradox given above. The fact that today's temperature and today's golf score happen to be the same does not let you conclude that t and g are equal. You must distinguish between what logicians call *contingent* and *necessary* identities. Contingent identity means only that their extensions happen to coincide in a given context. Necessary identity means that this always happens: Their intensions are equal. The correct rule for equality is that if t and g are necessarily equal, so are $y(t)$ and $y(g)$. Contingent equality of t and g is not enough, even to derive the contingent equality of $y(t)$ and $y(g)$.

You could, of course, formalize these ideas in a conventional, extensional logic system in which variables and expressions denote context-to-value functions. In practice, however, this approach is unnatural and impractical because intensions are complicated mathematical objects. Nor would anyone in his right mind think of temperature as denoting an infinite table nor consider statements about the temperature to be assertions about infinite tables.

Intensional programming is a more natural approach for these ideas. "Intensional programming" here means programming in a language that is also a formal system based on intensional semantics. Intensional-language programmers must be encouraged to think intensionally and should be provided with context-switching operators that let values from different contexts be combined without explicit context manipulation.

Table A.
Partial table of temperatures of each province for a range of days.

Date	Alberta	Saskatchewan	Manitoba	Ontario	Quebec
12/30/89	–11	–17	–16	–14	–15
12/31/89	–12	–16	–14	–15	–13
1/ 1/90	–13	–16	–13	–12	–14
1/ 2/90	–17	–20	–18	–19	–21
1/ 3/90	–14	–16	–14	–15	–14

Table B.
The intension of "five more degrees than yesterday's temperature in the eastern neighboring province" based on the data in Table A.

Date	British Columbia	Alberta	Saskatchewan	Manitoba	Ontario
12/31/89	– 6	–12	–11	– 9	–10
1/ 1/90	– 7	–11	– 9	–10	– 8
1/ 2/90	– 8	–11	– 8	– 7	– 9
1/ 3/90	–12	–15	–13	–14	–16
1/ 4/90	– 9	–11	– 9	–10	– 9

G	0	R1	R2	R3	R4
0			'unleaded ($/L)'	'regular ($/L)'	'super ($/L)'
D1	'station 1'	'self'	(1.01*right G) fby (G*1.012)	0.450 fby (G*1.01)	(1.025*left G) fby (G*1.015)
D2		'full'	0.02+up G	0.02+up G	0.02+up G
D3					
D4	'station 2'	'self'	(1.01*right G) fby (G*1.012)	0.460 fby (G*1.01)	(1.025*left G) fby (G*1.015)
D5		'full'	0.02+up G	0.02+up G	0.02+up G
D6					
D7	'station 3'	'self'	(1.01*right G) fby (G*1.012)	0.470 fby (G*1.01)	(1.025*left G) fby (G*1.015)
D8		'full'	0.02+up G	0.02+up G	0.02+up G

(a)

G	0	R1	R2	R3	R4
0			unleaded ($/L)	regular ($/L)	super ($/L)
D1	station 1	self	0.455	0.450	0.461
D2		full	0.475	0.470	0.481
D3					
D4	station 2	self	0.465	0.460	0.472
D5		full	0.485	0.480	0.492
D6					
D7	station 3	self	0.475	0.470	0.482
D8		full	0.495	0.490	0.502

Time = 0

(b)

G	0	R1	R2	R3	R4
0			unleaded ($/L)	regular ($/L)	super ($/L)
D1	station 1	self	0.460	0.455	0.468
D2		full	0.480	0.475	0.488
D3					
D4	station 2	self	0.470	0.465	0.479
D5		full	0.490	0.485	0.499
D6					
D7	station 3	self	0.480	0.475	0.489
D8		full	0.500	0.495	0.509

Time = 1

(c)

Figure 5. (a) A spreadsheet for gas prices in a year. **(b)** The calculated result of the spreadsheet at the first month and **(c)** at the second month.

G	0	R1	R2	R3	R4	R5
0			'unleaded ($/L)'	'regular ($/L)'	'super ($/L)'	'average'
D1	'station 1'	'self'	(1.01*right G) fby (G*1.012)	0.450 fby (G*1.01)	(1.025*left G) fby (G*1.015)	avg(G,3)
D2		'full'	0.02+up G	0.02+up G	0.02+up G	avg(G,3)
D3						
D4	'station 2'	'self'	(1.01*right G) fby (G*1.012)	0.460 fby (G*1.01)	(1.025*left G) fby (G*1.015)	avg(G,3)
D5		'full'	0.02+up G	0.02+up G	0.02+up G	avg(G,3)
D6						
D7	'station 3'	'self'	(1.01*right G) fby (G*1.012)	0.470 fby (G*1.01)	(1.025*left G) fby (G*1.015)	avg(G,3)
D8		'full'	0.02+up G	0.02+up G	0.02+up G	avg(G,3)
D9						
D10		'max'	max(G,9)	max(G,9)	max(G,9)	max(G,9)

Figure 6. The updated gas-price spreadsheet from Figure 5a with user-defined functions.

cannot (for now) handle higher order programs.

Using the gas-price example, suppose that the user wants to know, for each month, the average gas price of each station and the maximum gas price of each kind of gas for all stations. You can design three functions — Sum, Avg, and Max — to satisfy the user's requirement. The spreadsheet code (formulas) would be

```
sum(x,n) = if n <= 0 then 0
  else left x + sum(left x,n–1) fi;
avg(x,n) = sum(x,n) / n;
max(x,n) = if n eq 1 then up x
  elseif up x > M then up x
  else M fi
  where M = max(up x, n–1);
end;
```

The spreadsheet screen would be like that in Figure 6.

In this spreadsheet program, the spreadsheet variable *G* has a default definition of 0. This means that all cells not otherwise given a defining formula (of which there are infinitely many) are constantly 0.

A parameter in a function definition is an intension, a 3D object whose value depends on a given position and time, although this value may be constant in some dimension(s).

For example, the parameter *x* of the function Max may vary in space. You may consider the expression Up *x*, which is the actual parameter in the body of the recursive definition, as a new variable whose values are equal to *x*'s except that the horizontal axis is shifted up one point. In other words, a function takes as its argument the whole intension of the actual parameter, not just the particular value (extension) at the time and space point of the call.

Global variables. Consider the spreadsheet variable as a kind of 3D (infinite) array, where a spreadsheet program is similar to a program in an ordinary high-level language where only one array variable (corresponding spreadsheet variable) is used. All computations in a spreadsheet program are performed by operating on the cells of the spreadsheet variable, which is also how conventional spreadsheets work.

But, in many cases, using only one variable in a program — even if it is an array variable — seems inadequate. Our spreadsheet thus lets the user add global variables. You define a variable globally with a single expression. For example, if global variable *g* is defined as *g*=5, *g* is the variable with the value 5 at every cell at any time. In programming, you can use global variables to help define the spreadsheet variable.

A global variable may also vary in the spreadsheet space. So programmers may define a global variable in two spatial dimensions, we provide four global spatial operators: Hsby (for "horizontally succeeded by"), Vsby (for "vertically succeeded by"), Hbf (for "horizontally before"), and Vbf (for "vertically before"). These are similar to the global temporal operators Fby and Before. The spatial operator Hsby, for example, combines the spatial operation Left with the identity operation in the same way that the temporal operator Fby combines Prev with identity. Figures 7 and 8 illustrate the meaning of these operators.

Figure 9 is a an example spreadsheet that does the same work as the gas-price spreadsheet in Figure 6, but cells are defined by global variables and user-defined functions. The formulas in the spreadsheet are

```
month = 1 fby (month + 1);
regular(initial) = initial fby (1.01 * G);
unleaded = (1.01 * right G) fby (1.012 * G);
super = (1.025 * left G) fby (1.015 * G);
full = 0.02 + up G;

sum(x,n) = if n <= 0 then 0
  else left x + sum(left x,n–1) fi;
avg(x,n) = sum(x,n) / n;
max(x,n) = if n eq 1 then up x
  elseif up x > M then up x
  else M fi
  where M = max(up x, n–1);
end;

average = avg(G,X–2);
maximum = max(G,Y–2);

G = 0;
X = 0 hsby (1 + X);
Y = (Y + 1) vbf 0;
```

In the spreadsheet, the value of the global variable Month is initially 1; with each advance of time, it increases by 1. The value of the global variable *X* is equal to the horizontal coordinate in the positive direction (otherwise, it is 0), while the value of the global variable *Y* is equal to

X hbf Y	...	x_{-2}	x_{-1}	x_0	y_0	y_1	y_2	y_3	...		
X hsby Y	...	x_{-3}	x_{-2}	x_{-1}	x_0	y_0	y_1	y_2	...		
X	...	x_{-3}	x_{-2}	x_{-1}	x_0	x_1	x_2	x_3	...		
Y	...	y_{-3}	y_{-2}	y_{-1}	y_0	y_1	y_2	y_3	...		
		:	:	:	:	:	:	:		→	horizontal
	...	–3	–2	–1	0	1	2	3	...		

Figure 7. The behavior of the global spatial operators Hsby and Hbf.

X vbf Y	...	x_{-2}	x_{-1}	x_0	y_0	y_1	y_2	y_3	...		
X vsby Y	...	x_{-3}	x_{-2}	x_{-1}	x_0	y_0	y_1	y_2	...		
X	...	x_{-3}	x_{-2}	x_{-1}	x_0	x_1	x_2	x_3	...		
Y	...	y_{-3}	y_{-2}	y_{-1}	y_0	y_1	y_2	y_3	...		
		:	:	:	:	:	:	:		→	vertical
	...	–3	–2	–1	0	1	2	3	...		

Figure 8. The behavior of the global spatial operators Vsby and Vbf.

the absolute value of the vertical coordinate in the negative direction (otherwise, it is 0).

There is clearly no good reason to restrict global variables to having only global definitions (although earlier versions of our design did have such a restriction, thus the name "global").

You can also define a global variable pointwise with cells, like the spreadsheet variable. Its global definition, if there is one, is treated as the default definition. In this sense, the only difference between the spreadsheet variable and the global variables is that the values on the screen are those of the spreadsheet variable.

In fact, our design has eliminated even this distinction. Although we have described "the" spreadsheet variable, this was a simplification to aid the exposition. You can designate any global variable as the current spreadsheet variable, whose values are required and shown on the screen.

Multispreadsheets. Because of the concept of multiple spreadsheet variables, our spreadsheet is actually a multispreadsheet. In this multispreadsheet, you can link spreadsheets simply by using their names in the defining expressions. For example, one cell of spreadsheet *P* might have the defining expression Up *P*– Up *Q*. This links the cell in question with a cell in *Q*. The defining expressions of *Q* may have references to *P* and to other variables as well.

Consider again the gas-price example. Suppose that there are three gas companies — *G*1, *G*2, and *G*3 — that all have stations in the same areas. The gas prices of *G*1 and *G*2 are calculated in the spreadsheets with the same format of the previous figures. Gas company *G*3, however, wants to beat the minimum gas prices of *G*1 and *G*2 by 1 percent in each area.

G	0	R1	R2	R3	R4	R5
0			'month = '	month		
D1			'unleaded ($/L)'	'regular ($/L)'	'super ($/L)'	'average'
D2	'station 1'	'self'	unleaded	regular(0.45)	super	average
D3		'full'	full	full	full	average
D4						
D5	'station 2'	'self'	unleaded	regular(0.46)	super	average
D6		'full'	full	full	full	average
D7						
D8	'station 3'	'self'	unleaded	regular(0.47)	super	average
D9		'full'	full	full	full	average
D10						
D11		'maximum'	maximum	maximum	maximum	maximum

Figure 9. An updated spreadsheet for gas prices with global variables.

G	0	R1	R2	R3	R4	R5
0			'month = '	month		
D1			'unleaded ($/L)'	'regular ($/L)'	'super ($/L)'	'average'
D2	'station 1'	'self'	best	beat	beat	average
D3		'full'	beat	beat	beat	average
D4						
D5	'station 2'	'self'	beat	beat	beat	average
D6		'full'	beat	beat	beat	average
D7						
D8	'station 3'	'self'	beat	beat	beat	average
D9		'full'	beat	beat	beat	average
D10						
D11		'maximum'	maximum	maximum	maximum	maximum

Figure 10. An updated spreadsheet for gas prices with spreadsheet consolidation.

The following program does this work for *G3* by defining *G3*'s spreadsheet in terms of those of *G1* and *G2* (we do not define the functions or global variables already defined in Figure 9):

```
min(x,y) = if x < y then x else y fi;
beat = min(G1,G2) * 0.99;
```

Figure 10 shows the spreadsheet screen.

In the program, inputs are dealt with so that when an undefined global variable appears in the definition of a cell, it is considered as an input variable. An input variable, like other variables, also varies in space and time. The values of an input variable are required individually: When a value of the input variable at some cell and moment is demanded during the evaluation, the user will be asked to input the value for the input variable at that cell at that moment.

An example

As an example of using the intensional 3D spreadsheet, consider the formalization of a systolic algorithm for matrix multiplication, as shown in Figure 11. The algorithm is as follows:

- Input: An $m \times n$ matrix A and an $n \times l$ matrix B are read into an $m \times l$ array C of processors, as suggested in Figure 11.
- Output: The product of the input matrices A and B. The output is found in the processor array C.
- Method: Each processor C_{ij} is initially 0. At each time step, C_{ij} does the following: It (1) inputs two data items from the processors to its left and above or from the input matrices, respectively; (2) multiplies the two input data items and adds the result to the value stored by C_{ij}; (3) passes the input data items to the processors to its right and below, respectively, so they can be inputs for the neighboring processors at next time step; and (4), for each C_{ij}, after $n+i+j-2$ time steps, the result of $A \times B(i,j)$ is held by C_{ij}.

To simulate this systolic algorithm, imagine that each processor has three registers: LR (for "from left to right"), UD (for "up to down") and Value. LR receives data from its left neighboring processor or from the input and passes the data to its right neighbor. Similarly, UD receives data from the processor above or from the input and passes the data to its neighbor below. Value holds the current value for the processor that may be used for further computation or is the final result.

In a spreadsheet program simulating the algorithm, consider each cell in the region (1,1) to (*m*,*l*) as representing a processor. You can use three corresponding variables — *lr*, *ud*, and *value* — to represent the three registers. These three variables vary in space as well as time. The

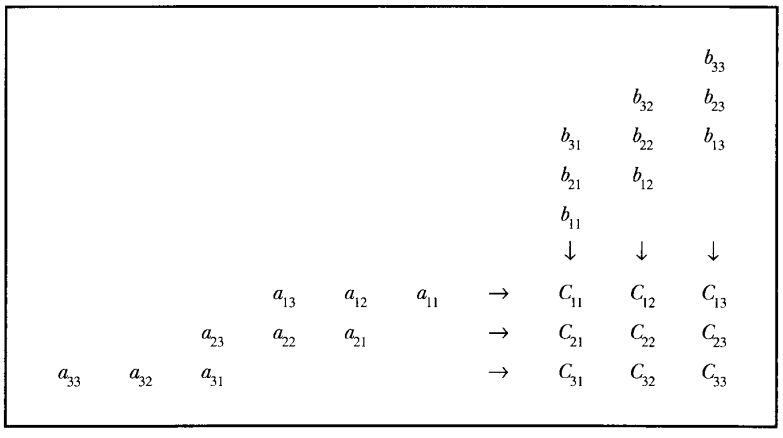

Figure 11. Systolic matrix multiplication.

PA	L4	L3	L2	L1	0	R1	R2	R3
U4								inpv(b33)
U3							inpv(b32)	inpv(b23)
U2						inpv(b31)	inpv(b22)	inpv(b13)
U1						inpv(b(21)	inpv(b12)	inpv(0)
0						inpv(b11)	inpv(0)	inpv(0)
D1			inph(a13)	inph(a12)	inph(a11)	value	value	value
D2		inph(a23)	inph(a22)	inph(a21)	inph(0)	value	value	value
D3	inph(a33)	inph(a32)	inph(a31)	inph(0)	inph(0)	value	value	value

Figure 12. A spreadsheet simulating systolic matrix multiplication.

value of *lr*, for example, at a particular cell and a particular time is the contents of the LR register at that point in the grid and at that stage in the computation. The variable *lr*, like *ud* and *value*, is an example of an intension that is best thought of as a field of time-varying entities. These variables are defined globally. If you select one of them as the current spreadsheet variable, the display will show the contents of the registers in question as the computation proceeds.

The following spreadsheet program simulates the systolic multiplication of two 3×3 matrices, in which the user wants to observe the *value* registers of the processors in the array:

```
lr = 0 fby (if j eq 1 then left PA else left lr fi);
ud = 0 fby (if i eq 1 then up PA else up ud fi);
value = 0 Fby (if T >= i+j+1 then value
  else next lr \ (*** next ud + value fi);

inph(x) = x fby left PA;
inpv(x) = x fby up PA;
PA = 0;

i = (i + 1) vbf 0;
j = 0 hsby (j + 1);
T = 0 fby (T + 1);
```

Figure 12 shows the spreadsheet screen.

In the program, *PA* is the spreadsheet variable whose default definition is zero. There are three auxiliary variables: *i*, *j*, and *T*. The variables *i* and *j* are row index and column index, respectively, and *T* is time index.

To simulate data flowing through the processor array from left to right and from above to below, you define two functions: Inph and Inpv. At initial time (time 0), the calls of these functions return the values of their arguments (matrix elements). At each following time step, they return values of *PA* at cells on their left or above at the previous time step. According to the systolic algorithm, variable *lr*, representing the LR register in a processor, is initially 0; at each following time step it inputs data from the input matrix *A* if the processor is in the first column. Otherwise, it receives data from *lr* of its left neighboring processor. UD works similarly.

The variable *value*, representing the Value register in a processor, is initially defined as 0. At each time step before the final result is computed for that processor, it has the value of the sum of its previous value and the product of the current values of *lr* and *ud* in the same processor. Then, after $i+j+1$ time steps, it holds the computed result for the processor until the computation of the whole processor array finishes.

Figure 13 shows the computation of this spreadsheet at time steps 0, 2, and 7 in the simulation of the systolic multiplication of these two matrices:

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 7 & 9 & 11 \\ 13 & 15 & 17 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

Demand-driven model

We have implemented the intensional spreadsheet with a demand-driven computation model. In demand-driven computation, an operation is performed only if all its operands, which are data items, are available and there is a demand for the result of the operation. If there is a demand for an operation that depends on an operand whose value is not available, a demand for that value is sent to the operation that produces it. When, as a result of such demands, there is a value available for each operand, the operation is performed (consuming the operands), and its result is sent to the operations that need the result.

Our spreadsheet interpreter has the structure shown in Figure 14. The basis of the spreadsheet interpreter is the calculating machine that performs ordinary operations, like the CPU in a conventional computer.

During evaluation, the horizontal and vertical registers hold the horizontal and vertical coordinates for the current cell, and the time register holds the current time. By performing intensional operations, the contents of these registers may change. The interpreter's value warehouse and definition warehouse contain values and definitions, respectively, for various variables; these values and definitions can be accessed by their owners' names, spatial positions, and time.

The entire computation of a spreadsheet program is driven by the demands of the user who wants to evaluate one or more cells in the spreadsheet. In computing the value of the spreadsheet variable at some cell at some time, the interpreter may be led to compute various variables at possibly different cells and moments.

To do this, it first checks the value warehouse to see if the value of a demanded variable at the current cell and time has been computed. If it has, the interpreter returns the value to the operation that demands it.

Otherwise, the definition of the variable associated with the current cell will be fetched from the definition warehouse; the fetched definition is then evaluated and its result is stored in the value warehouse and returned to the demanding operation. The entire computation ends when all demands are satisfied.

PA	L4	L3	L2	L1	0	R1	R2	R3
U4								18
U3							16	12
U2						14	10	6
U1						8	4	0
0						2	0	0
D1			5	3	1	0	0	0
D2		11	9	7	0	0	0	0
D3	17	15	13	0	0	0	0	0

(a)

PA	L4	L3	L2	L1	0	R1	R2	R3
U4								0
U3							0	0
U2						0	0	18
U1						0	16	12
0						14	10	6
D1			0	0	5	26	4	0
D2		0	0	11	9	14	0	0
D3	0	0	17	15	13	0	0	0

(b)

PA	L4	L3	L2	L1	0	R1	R2	R3
U4								0
U3							0	0
U2						0	0	0
U1						0	0	0
0						0	0	0
D1			0	0	0	96	114	132
D2		0	0	0	0	240	294	348
D3	0	0	0	0	0	240	474	564

(c)

Figure 13. (a) Calculation of Figure 12 at time 0, **(b)** at time 2, and **(c)** at time 7.

Intensional logic lets you approach spreadsheet programming from a new point of view. You are no longer forced to think of the cells in a spreadsheet as individual, independent variables. Instead, you can think of the spreadsheet as a single entity that varies in space and time. With intensional operators, you can write spreadsheet programs in a purely algebraic form. This makes the structure of spreadsheet programs logical and clear, and it makes the designs of spreadsheet programs more problem-oriented, since the concepts of space and time used are abstracted from the real world.

At the same time, by basing the design on a well-understood logic, our approach can enrich spreadsheet systems with many of the advantages of mainstream high-level languages, like user-defined functions.

In fact, the intensional spreadsheet provides an ideal programming environment for the language Plane Lucid. This environment realizes the interactive and incremental style of programming in Plane Lucid, as well as the rapid prototyping of Plane Lucid programs.

Another important advantage in basing the spreadsheet on a declarative language is that you can specify the semantics

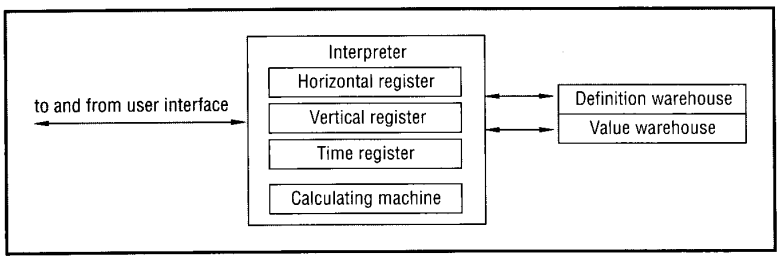

Figure 14. Spreadsheet interpreter's structure.

(meaning) of spreadsheet programs with the same well-understood least-fixpoint method used to define Plane Lucid itself. This semantics lets you in turn derive formal rules for verifying and transforming spreadsheet programs.

Thus, spreadsheet programming can become just as reliable and respectable as programming in a more traditional high-level language. If anything, spreadsheet programs would be even more respectable, since most widely used languages are (unlike Plane Lucid) imperative, not declarative; they are based on commands and their effects rather than on expressions and their values. As a result, their formal properties are much more complex.

The intensional approach also helps you uncover extensions to the basic design by considering richer coordinate systems. One obvious extension is to add a third spatial dimension so the user can define a time-varying cube of values. Not quite so obvious is the possibility of extra time dimensions, which would let the user specify nested iterations. In fact, from a purely semantic point of view, there is no difficulty in having infinitely many extra space or time dimensions.

We could also allow spatial nesting, where a single cell at the top level could be opened to reveal an entire plane of cells. Each cell at the top level could hide its own plane, and the cells in the second level could each encapsulate planes on a third level, and so on. The value in a cell at a given level could be defined in terms of cells in its subspreadsheet. To do this, we would need extra operators to switch levels. Spatial nesting would allow a truly structured, top-down approach to the construction of spreadsheet programs.

On the application side, you can consider the intensional spreadsheet as a parallel-programming tool for synchronous parallel computations. One example is systolic matrix multiplication. In parallel programming, you may use the spreadsheet's temporal dimension as the central clock that synchronizes the computation at each time point. You may also consider each cell in the spreadsheet plane to represent a processor; the definitions of various variables at the cell would specify the processor's computational behavior; and the intensional operations in the definitions would specify the communication between the processor and other processors corresponding to other cells.

The intensional spreadsheet also has potential to be the basis of a multidimensional constraint language. The current implementation already lets you think of the definitions in the cells as a set of spatial and temporal constraints on the values of the cells. After evaluation, the values in the cells satisfy the constraints specified in the spreadsheet program.

However, the current intensional spreadsheet is not a true constraint language, since the definition of a cell deterministically defines only a single value of the cell at a moment in time, instead of being a general constraint that defines ranges of values that satisfy the constraint. To develop a constraint-satisfaction system on the intensional spreadsheet, we need to extend the semantics of Plane Lucid so cells can contain nondeterministic and nondirectional constraints on their values and those of their neighbors. ❖

Acknowledgments

This research was sponsored in part by the National Scientific and Engineering Research Council of Canada. Du is also partially supported by a British Columbia Advanced Systems Foundation Graduate Research Scholarship.

References

1. W. Du and W.W. Wadge, "An Intensional Language as the Basis of a 3D Spreadsheet Design," *Proc. Int'l Conf. Computer Languages*, CS Press, Los Alamitos, Calif., 1988, pp. 2-9.
2. W.W. Wadge and E.A. Ashcroft, *Lucid, the Dataflow Programming Language*, Academic Press, London, 1985.

Weichang Du is a PhD candidate at the University of Victoria. His research interests include application languages, nonprocedural programming, parallel programming, and dataflow.

Du received a BS in computer science and engineering from Beijing Computer Institute and an MS in computer science from the University of Victoria. He is a student member of the IEEE Computer Society.

William W. Wadge is an associate professor of computer science at the University of Victoria. His research interests include nonprocedural languages, dataflow, intensional logic, semantics, and data types.

Wadge received a BA in mathematics from the University of British Columbia and a PhD in mathematics from the University of California at Berkeley.

Address questions about this article to the authors at Computer Science Dept., University of Victoria, PO Box 1700, Victoria, B.C. V8W 2Y2, Canada.