Bidirectional Spreadsheet Formulas

A. Cunha **N. Macedo** H. Pacheco **N. Souza**

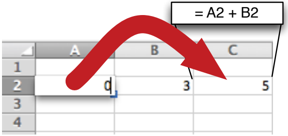
HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY
INESCTEC

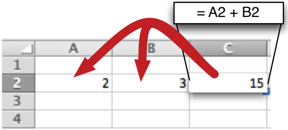
Universidade do Minho

FATBIT Workshop 2013
July 17, Braga, Portugal

Motivation

- Spreadsheet formulas are inherently unidirectional;
- However, sometimes we want to tweak the input data to attain a particular output:
  - forecast of profit margins;
  - calculation of tax deductions;
  - bet winnings calculators;
- We want to specify the *output*, and have the *input updated* accordingly.





Motivation

- Profit forecasting example:

=B2+C2

=IF(F2>0;#F2;"Loss")

	A	B	C	D	E	F	G
1	**Name**	**Cost**	**Taxes**	**Profit %**	**T. Cost**	**Profit**	**Print**
2	A	50	3	1,2	53	10,6	10,6
3	B	20	2	2	22	22	22
4	C	90	10	0,5	100	-50	Loss

=#D2*E2−E2

- Ad-hoc solutions:
  - Write the function in the backward direction instead;
  - Resort to auxiliary functions;
  - Manually modify the input until the desired output is attained.

Spreadsheet formulas as lenses

- Lenses are a popular bidirectional transformation framework;
- Forward $\mathsf{get} : S \rightarrow V$ and backward $\mathsf{put} : S \times V \rightarrow S$ transformations:

$$\mathsf{get}\,(\mathsf{put}\,s\,v) = v \qquad (\textsc{PutGet})$$
$$\mathsf{put}\,(\mathsf{get}\,s)\,s = s \qquad (\textsc{GetPut})$$

- PutGet guarantees that the user update is preserved;
- GetPut guarantees that the system is stable;
- Is undoability desired/feasible?

$$\mathsf{put}\,(\mathsf{get}\,s)\,(\mathsf{put}\,s\,v') = s \qquad (\textsc{Undo})$$

Design decisions

- Online setting: updates on a cell are automatically propagated:
  - Single cell updates rather than spreadsheet updates;
  - Duplication is supported to a certain degree;
- Conservative updating: User marks the updatable input cells:
  - Cells that may be updated marked by #;
  - The bidirectional layer "does no (unexpected) harm";
- White-box: Backward transformation (and invariants) specified as a spreadsheet formula:
  - Allows the user to better understand the transformation (and eventually parameterize it).

Function examples: $+$

- Catalog of bidirectionalized functions;
- Behavior of the transformation depends on the # cells;
- E.g., addition $+ : Number \times Number \to Number$:
  - $C = \#A + B$

    $$A \leftarrow \mathsf{put}_{\blacksquare + B}\, C\, (A, B) = \mathtt{C} - \mathtt{B}$$

  - $C = A + \#B$

    $$B \leftarrow \mathsf{put}_{A + \blacksquare}\, C\, (A, B) = \mathtt{C} - \mathtt{A}$$

  - $C = \#A + \#B$

    $$A \leftarrow \mathsf{put}_{\blacksquare + \square}\, C\, (A, B) = \mathtt{C}/2$$
    $$B \leftarrow \mathsf{put}_{\square + \blacksquare}\, C\, (A, B) = \mathtt{C}/2$$

Function examples: len

- E.g., len : $String \rightarrow Int$:
  - $B = \text{len}(\#A)$:

$$\text{put}_{\text{len}(\blacksquare)}\ B\ A = \begin{array}{ll} \texttt{if}(\texttt{B} \leq \texttt{len}(\texttt{A}); & \texttt{left}(\texttt{A}; \texttt{B}); \\ & \texttt{A\&repeat}(\text{``}\texttt{x}\text{''}; \texttt{B} - \texttt{len}(\texttt{A}))) \end{array}$$

Function examples: if

- E.g., if:
  - $D = \text{if}(A, \#B, \#C) : \mathit{Bool} \times a \times b \to a \cup b$:

$$\text{put}_{\text{if}(A,\blacksquare,\square)}\, D\,(A, B, C) = \mathtt{if(A; D; B)}$$
$$\text{put}_{\text{if}(A,\square,\blacksquare)}\, D\,(A, B, C) = \mathtt{if(not\,A; D; C)}$$

- Non-updatable conditions (for now).

Formula chaining

- Let us consider only formula chaining:
  - cells contain *values* or *functions* applied only to *cell references*;
  - function nesting $f\,(g\,(A))$ can be decomposed into this shape;
- It suffices to bidirectionalize individual cells:
  - let a cell B be updated as $B \leftarrow b$;
  - if B is a function cell $B = f(A_1, \ldots, A_n)$
  - update every #-tagged A_i as

$$A_i \leftarrow \mathrm{put}_{f(\square_1,\ldots,\blacksquare_i,\ldots,\square_n)}\, B\,(A_1, \ldots, A_n)$$

  - each updated cell will react and recompute its forward or backward formulas.

Chaining example

- $D = \#C + \#B$, $C = \mathsf{len}(\#A) = 5$, $B = 10$, $A = \text{“}hello\text{”}$;
- $D \leftarrow 8$;
- $\quad C \leftarrow \mathsf{put}_{\blacksquare + \square}\, 8\,(10, 5) = 4$
- $\quad B \leftarrow \mathsf{put}_{\square + \blacksquare}\, 8\,(10, 5) = 4$
- $\qquad A \leftarrow \mathsf{put}_{\mathsf{len}(\blacksquare)}\, 4\,\text{“}hello\text{”} = \text{“}hell\text{”}$

Consistency rules

- All # paths must eventually lead to value cells (i.e., an updatable cell);
- Circularity is not allowed $\Rightarrow$ the lens laws ensure convergence;
  - already handled by spreadsheet applications;
- Cells referenced more than once cannot be marked with #;
- No # marks on if conditions.

Updatability

- The previous technique would work fine if all functions were *surjective*;
  - (actually, since spreadsheets are not typed there are technically no surjective functions);
- How to detect/handle updates outside the *range* of a function?

Demo

Demo I

Invariants

- For each function cell A, an *invariant* Φ_A must be inferred...
- ... which is the *range* of the function over the invariants of its *domain*;
- The user is allowed to specify additional invariants on *source* cells (e.g., Excel's *Data Validation* feature);
- Invariants are propagated through formula chaining.

Normalized invariants

- Invariants are represented by sets of values and abstract set representations:
  - $\mathit{Invariant} \in \mathcal{P}(\mathit{Clause})$
  - $\mathit{Clause} \in \mathit{Number}|\mathit{Int}|\mathit{Text}|\mathit{Bool}$
  - $\mathit{Number} \in \mathbb{R}|[\mathbb{R}..\mathbb{R}]|\mathtt{Univ}_{\mathbb{R}}$
  - $\mathit{Int} \in \mathbb{Z}|\langle\mathbb{Z}..\mathbb{Z}\rangle|\mathtt{Univ}_{\mathbb{Z}}$
  - $\mathit{Text} \in \Sigma^*|\mathtt{Len}_{\mathit{Int}}|\mathtt{Univ}_{\Sigma^*}$
  - $\mathit{Bool} \in \mathtt{True}|\mathtt{False}|\mathtt{Bool}$
- Inspired by existing spreadsheet data constraints (Excel's *Data Validation*).

Normalized invariants

- Invariants are manipulated through abstract interpretation;
- Required operations:
  - $\cup$
    - $[\![\{\langle 0..20\rangle\} \cup \{10, \langle 20..30\rangle\}]\!] \rightsquigarrow \{\langle 0..30\rangle\}$
    - $[\![\{\mathsf{Len}_5\} \cup \{\text{``}hello\text{''}, \text{``}hi\text{''}\}]\!] \rightsquigarrow \{\mathsf{Len}_5, \text{``}hi\text{''}\}$
  - $\cap$
    - $[\![\{\langle 0..20\rangle\} \cap \{10, \langle 20..30\rangle\}]\!] \rightsquigarrow \{10, 20\}$
    - $[\![\{\mathsf{Len}_5\} \cap \{\text{``}hello\text{''}, \text{``}hi\text{''}\}]\!] \rightsquigarrow \{\text{``}hello\text{''}\}$
  - $-$
    - $[\![\{\langle 0..20\rangle\} - \{10, \langle 20..30\rangle\}]\!] \rightsquigarrow \{\langle 0..9\rangle, \langle 11..19\rangle\}$
    - $[\![\{\mathsf{Len}_5\} - \{\text{``}hello\text{''}, \text{``}hi\text{''}\}]\!] \rightsquigarrow$ problematic!
  - $\in$
    - $[\![5 \in \{\langle 0..20\rangle\}]\!] \rightsquigarrow \texttt{True}$
    - $[\![\text{``}hi\text{''} \in \{\mathsf{Len}_5\}]\!] \rightsquigarrow \texttt{False}$

Invariants: $+$

- $x, y, z \in Number$:
  - $[\![[x..y] + z]\!] \rightsquigarrow [x + z..y + z]$
  - $[\![\mathtt{Univ}_{\mathbb{R}} + z]\!] \rightsquigarrow \mathtt{Univ}_{\mathbb{R}}$
  - $[\![x + y]\!] \rightsquigarrow x + y$
- $A \in \{[0..10]\}$ and $B \in \{[10..20]\}$:
  - $C = \#A + B$;
    - $\Phi_{\blacksquare + B} = \{x + B | x \leftarrow \Phi_A\}$, since B constant;
    - $C \in \{[0 + B..10 + B]\}$.
  - $C = \#A + \#B$
    - $\Phi_{\blacksquare + \blacksquare} = \{x + y | x \leftarrow \Phi_A, y \leftarrow \Phi_B\}$, since A and B free;
    - $C \in \{[10..30]\}$;

Invariants: len

- $x \in \mathit{Text}$:
  - $[\![\mathsf{len}(\mathtt{Len}_{\mathsf{n}})]\!] \rightsquigarrow n$
  - $[\![\mathsf{len}(x)]\!] \rightsquigarrow \mathsf{len}(x)$
  - $[\![\mathsf{len}(\mathtt{Univ}_{\Sigma^*})]\!] \rightsquigarrow \langle 0..\rangle$
- $\Phi_{\mathsf{len}(\blacksquare)} = \{\mathsf{len}(x) | x \leftarrow \Phi_A\}$;
- $B = \mathsf{len}(\#A)$, $A \in \{\mathtt{Len}_3, \text{``}hello\text{''}\}$
  - $B \in \{3, 5\}$;

Invariants: if

- The if condition is presented as a normalized invariant;
  - may now be defined over #-marked cells in the branches;
- $C = \text{if}(\Psi_A, \#A, \#B)$;
- $\Phi_{\text{if}(\Psi_A,\blacksquare,\blacksquare)} = (\Phi_A \cap \Psi_A) \cup (\Phi_A - \Psi_A \neq \emptyset)?\Phi_B : \emptyset$;
- $\Psi_A = [0..10]$, $A \in \{[0..20]\}$ and $B \in \{[-10..10]\}$;
- $D \in \{[-10..10]\}$.

put synthesis

- put must now be synthesized from the source invariants;
- Guarantees that, given *valid target* values, it produces *valid source* values;
- The synthesized put must be updated when invariants change;
- Sometimes there is some freedom in the synthesis;
  - For $\Phi \in \mathit{Invariant}$, $\overline{\Phi}$ selects a value;
  - default value?
  - user specified value?

put synthesis: len

$$
\begin{array}{l}
\forall c_B \in \Phi_B \\
\quad \boxed{\mathtt{if}(\llbracket \mathtt{B} \in c_B \rrbracket;} \\
\quad case(S = len_{c_B} \cap \Phi_A) of \\
\qquad \mathtt{Univ}_{\Sigma^*} : \boxed{\begin{array}{ll} \mathtt{if}(\mathtt{B} \leq \mathtt{len}(\mathtt{A}); & \mathtt{left}(\mathtt{A}, \mathtt{B}); \\ & \mathtt{A}\&\llbracket \overline{\mathtt{Len}_{B-\mathsf{len}(A)}} \rrbracket)); \end{array}} \\
\qquad \mathtt{Len}_{\langle i..j \rangle} : \boxed{\begin{array}{ll} \mathtt{if}(\mathtt{B} \leq \mathtt{len}(\mathtt{A}); & \mathtt{left}(\mathtt{A}, \mathtt{B}); \\ & \mathtt{A}\&\llbracket \overline{\mathtt{Len}_{B-\mathsf{len}(A)}} \rrbracket)); \end{array}} \\
\qquad otherwise : \forall v_i \in S \\
\qquad\quad \begin{array}{ll} if\ (i < \#S) & \boxed{\begin{array}{ll} \mathtt{if}(\mathtt{v_i} = \mathtt{left}(\mathtt{A}, \mathtt{B}); & \mathtt{v_i}; \\ \mathtt{if}(\mathtt{A} = \mathtt{left}(\mathtt{v_i}, \mathtt{len}(\mathtt{A})); & \mathtt{v_i}; \end{array}} \\ else & \boxed{\mathtt{v_i}));} \end{array}
\end{array}
$$

put synthesis: len

- $B = \text{len}(\#A)$, $A \in \{Len_4, \text{“}abc\text{”}, \text{“}xyz\text{”}\}$, $B \in \{3, 4\}$

$$
\begin{aligned}
\text{put}_{\text{len}(\blacksquare)}\, B\, A = & \texttt{if}(\texttt{B} = 3;\\
& \quad \texttt{if}(\text{“}\texttt{abc}\text{”} = \texttt{left}(\texttt{A}; \texttt{B}); \text{“}\texttt{abc}\text{”};\\
& \quad \texttt{if}(\texttt{A} = \texttt{left}(\text{“}\texttt{abc}\text{”}; \texttt{B}); \text{“}\texttt{abc}\text{”};\\
& \qquad \text{“}\texttt{xyz}\text{”}));\\
& \texttt{if}(\texttt{B} = 4;\\
& \quad \texttt{if}(\texttt{B} \leq \texttt{len}(\texttt{A});\\
& \qquad \texttt{left}(\texttt{A}; \texttt{B});\\
& \qquad \texttt{A}\&\texttt{repeat}(\text{“}\texttt{x}\text{”}; \texttt{B} - \texttt{len}(\texttt{A}))));)
\end{aligned}
$$

Cell ranges

- For functions over *cell ranges*, $\#$ marks all cells in the input range;
- E.g., $B = \mathsf{sum}(\#(A_0 : A_n))$, for $\mathsf{sum} : [\mathit{Number}] \rightarrow \mathit{Number}$;
- $\Phi_B = \{a_0 + \cdots + a_n | a_0 \leftarrow \Phi_{A_0}, \ldots, x_n \leftarrow \Phi_{A_n}\}$;
- $\forall i \in [0..n] : \mathsf{put}_{\mathsf{sum}(\blacksquare_i)}\, B\, (A_0 : A_n)$.

Formula nesting

- Nested functions $g\,(f\,(A))$ can be decomposed into formula chaining with auxiliary cells;

$$A1 = f(g(A2)) \quad \rightarrow \quad A1 = f(Ax) \quad Ax = g(A2)$$

- Semantically equivalent to the composition of lenses:

$$\mathrm{put}_{g \cdot f}\, B\, A = \mathrm{put}_f\, (\mathrm{put}_g\, B\, f(A))\, A$$

Demo

Demo II