Bidirectional Spreadsheet Formulas

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

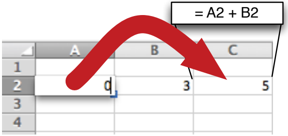
HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY
INESCTEC

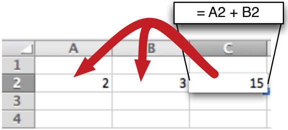
Universidade do Minho

FATBIT Workshop 2013
October 3, 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 $\mathsf{get} : S \to V$ and backward $\mathsf{put} : S \times V \to S$ 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 $+ : Number \times Number \rightarrow Number$:
  - $C = \#A + B$

    $$A \leftarrow \mathsf{put}_{\blacksquare + B}\; c\,(a, b) = \mathsf{c} - \mathsf{b}$$

  - $C = A + \#B$

    $$B \leftarrow \mathsf{put}_{A + \blacksquare}\; c\,(a, b) = \mathsf{c} - \mathsf{a}$$

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

    $$A \leftarrow \mathsf{put}_{\blacksquare + \square}\; c\,(a, b) = \mathsf{c}/2$$
    $$B \leftarrow \mathsf{put}_{\square + \blacksquare}\; c\,(a, b) = \mathsf{c}/2$$

Function examples: LEN

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

$$\text{put}_{\text{LEN}(\blacksquare)}\ b\ a = \begin{array}{ll} \texttt{IF(b} \leqslant \texttt{LEN(a);} & \texttt{LEFT(a;b);} \\ & \texttt{a\&REPEAT("x";b} - \texttt{LEN(a)))} \end{array}$$

Function examples: IF

- E.g., IF:
  - $D = \mathsf{IF}(A, \#B, \#C) : Bool \times a \times b \rightarrow a \cup b$:

$$\mathsf{put}_{\mathsf{IF}(A,\blacksquare,\square)}\ d\ (a, b, c) = \mathtt{IF}(\mathtt{a}; \mathtt{d}; \mathtt{b})$$
$$d\ (a, b, c)\mathsf{put}_{\mathsf{IF}(A,\square,\blacksquare)}\ d\ (a, b, c) = \mathtt{IF}(\mathtt{NOT}\,\mathtt{a}; \mathtt{d}; \mathtt{c})$$

- 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

$$A_i \leftarrow \mathsf{put}_{f(\square_1,\ldots,\blacksquare_i,\ldots,\square_n)}\; b\,(a_1, \ldots, a_n)$$

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

Chaining example

- $D = \#C + \#B$, $C = \mathtt{LEN}(\#A) = 5$, $B = 10$, $A =$ "`hello`";
- $D \leftarrow 8$;
- $\quad C \leftarrow \mathsf{put}_{\blacksquare + \square}\, 8\, (10, 5) = 4$
- $\quad B \leftarrow \mathsf{put}_{\square + \blacksquare}\, 8\, (10, 5) = 4$
- $\qquad A \leftarrow \mathsf{put}_{\mathtt{LEN}(\blacksquare)}\, 4$ "`hello`" = "`hell`"

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 in the call graph cannot be marked #;
- 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 *source cells*;
- 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:
  - $\mathit{Invariant} \in \mathcal{P}(\mathit{Clause})$
  - $\mathit{Clause} \in \mathit{Number}|\mathit{Int}|\mathit{Text}|\mathit{Bool}$
  - $\mathit{Number} \in [\mathbb{R}..\mathbb{R}[|]\mathbb{R}..\mathbb{R}]|[\mathbb{R}..\mathbb{R}]|[\mathbb{R}..[|]..\mathbb{R}]|\mathtt{Univ}_{\mathbb{R}}$
  - $\mathit{Int} \in \langle\mathbb{Z}..\mathbb{Z}\rangle|\langle\mathbb{Z}..\langle|\rangle..\mathbb{Z}\rangle|\mathtt{Univ}_{\mathbb{Z}}$
  - $\mathit{Text} \in \Sigma^*|\mathtt{len}_{\mathit{Int}}|\mathtt{Univ}_{\Sigma^*}$
  - $\mathit{Bool} \in \mathtt{True}|\mathtt{False}|\mathtt{Bool}$
- Inspired by existing spreadsheet data constraints (Excel's *Data Validation*).

Normalized invariants

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

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

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

Invariants: if

- The if condition is presented as a normalized invariant;
  - may now be defined over #-marked cells in the branches;
- $\text{IF}\ (A \leqslant B; \#A; \#B)$ is interpreted as $\text{IF}\ ((]..B], [A..[); \#A; \#B)$;
- $\Psi_A = [0..10]$, $A \in \{[0..20]\}$ 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;
- Requires a *traceability* R between target and source invariants;
- Sometimes there is some freedom in the synthesis;
  - For $\Phi \in \mathit{Invariant}$, $\mathsf{sel}\ (\Phi, a)$ selects a value from Φ close to a;
  - default value?
  - user specified value?

put synthesis: LEN

$\boxed{\textbf{case } b \textbf{ of}}\ \forall\,(\Phi,\psi) \in R : \boxed{\psi_i}$

$\boxed{\textbf{if } (b \leqslant \text{LEN } a)}$

$\textbf{if } (\text{Univ} \in \Phi \vee \exists\, \text{len}_x \in \Phi)$

$\boxed{\text{LEFT } (b, a)}$

else

$\forall\,(\phi_i : \sum^* \in \Phi) : \boxed{\textbf{if } (\text{LEFT } (b, a) = \phi_i)\ \phi_i}$

$\boxed{\textbf{else } \phi_n}$

else

$\textbf{if } (\text{Univ} \in \Phi \vee \exists\, \text{len}_x \in \Phi)$

$\boxed{a\&}\ \text{sel } (\text{len}_{b - \text{LEN } a}, a)$

else

$\forall\,(\phi_i : \sum^* \in \Phi) : \boxed{\textbf{if } (\text{LEFT } (b, \phi_i) = a)\ \phi_i}$

$\boxed{\textbf{else } \phi_n}$

put synthesis: LEN

$B = \mathrm{LEN}(\#A)$, $A \in \{len_{\langle 6..10\rangle}, \text{“}\texttt{hello}\text{”}, \text{“}\texttt{hallo}\text{”}\}$, $B \in \{\langle 5..10\rangle\}$

$$
\begin{array}{l}
\mathrm{put}_{\mathrm{LEN}\ (\blacksquare)}\ \{(\{\texttt{"hello"}, \texttt{"hallo"}\}, 5), (\{\mathrm{len}_{[6..10]}\}, [6..10])\}\ (b, a) = \\
\quad \textbf{case}\ b\ \textbf{of} \\
\qquad 5 \rightarrow \quad \textbf{if}\ (b \leqslant \mathrm{LEN}\ a) \\
\qquad\qquad\quad \textbf{if}\ (\mathrm{LEFT}\ (b, a) = \texttt{"hallo"})\ \texttt{"hallo"} \\
\qquad\qquad\quad \textbf{if}\ (\mathrm{LEFT}\ (b, a) = \texttt{"hello"})\ \texttt{"hello"} \\
\qquad\qquad\quad \textbf{else}\ \texttt{"hello"} \\
\qquad\qquad \textbf{else} \\
\qquad\qquad\quad \textbf{if}\ (\mathrm{LEFT}\ (b, \texttt{"hallo"}) = a)\ \texttt{"hallo"} \\
\qquad\qquad\quad \textbf{if}\ (\mathrm{LEFT}\ (b, \texttt{"hello"}) = a)\ \texttt{"hello"} \\
\qquad\qquad\quad \textbf{else}\ \texttt{"hello"} \\
\qquad [6..10] \rightarrow \textbf{if}\ (b \leqslant \mathrm{LEN}\ a) \\
\qquad\qquad\quad \mathrm{LEFT}\ (b, a) \\
\qquad\qquad \textbf{else} \\
\qquad\qquad\quad a\ \&\ \mathrm{sel}\ (\mathrm{len}_{b - \mathrm{LEN}\ a})
\end{array}
$$

Invariants

- How far can normalized invariants take us?
- Cannot describe the exact range of, e.g.:
  - A^2 over integers;
  - CONCATENATE, e.g., CONCATENATE (len_2; "x");
  - IF for conditions that depend on A and B.
- Expand the invariant grammar (e.g., allow regular expressions)?
- Allow overestimations?
  - put would fail for values outside the range;
  - no updatability guarantees;
- Allow underestimations?
  - would cut values for which put was well-behaved;
  - may still not be viable with current invariants.

Cell ranges

- For functions over *cell ranges*, $\#$ marks all cells in the input range;
- E.g., $B = \mathsf{SUM}(\#(A_0 : A_n))$, for $\mathsf{SUM} : [\mathit{Number}] \rightarrow \mathit{Number}$;
- $\Phi_B = \{a_0 + \cdots + a_n | a_0 \leftarrow \Phi_{A_0}, \ldots, x_n \leftarrow \Phi_{A_n}\}$;
- $\forall i \in [0..n] : \mathsf{put}_{\mathsf{SUM}(\square_0 \ldots \blacksquare_i \ldots \square_n)}\, b\,(a_0 : a_n)$.

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:

$$\mathsf{put}_{g \cdot f}\; b\, a = \mathsf{put}_f\; (\mathsf{put}_g\; b\, f(a))\; a$$

Demo

Demo II