Type-safe Evolution of Spreadsheets

Jácome Cunha Joost Visser Tiago Alves João Saraiva

Universidade do Minho, Portugal, {jacome,jas}@di.uminho.pt
Software Improvement Group, The Netherlands, {j.visser,t.alves}@sig.eu

FASE (ETAPS) 2011

March 26 – 3 April

Agenda

1. Motivation
2. An Example Scenario
3. Modeling Spreadsheets
4. Two-Level Transformation (2LT)
5. Modeling Spreadsheets, Again
6. Evolution Rules
   - Combinator Rules
   - Semantic Rules
   - Layout Rules
7. Conclusions

Motivation

- Spreadsheets are widely used;
- Spreadsheets are notoriously error-prone;
- Many errors are introduced when changing data;
- But many others when changing the structure;
- Models capturing the interdependencies between data can help;
- Co-evolution of models and instances.

An Example

- Budget for travel, hotel and local transportation expenses.

	A	B	C	D	E	F	G	H	I
1	Budget		Year			Year			
2			Year=2010			Year=2011			
3	Category	Name	Qnty	Cost	Total	Qnty	Cost	Total	Total
4		travel	2	200	400	2	450	900	1300
5		hotel	5	100	500	8	80	640	1140
6		local travel	4	20	80	2	35	70	150
7	Total				980			1610	2590

- At the beginning of each year, it needs to be modified to accommodate the next year;
- Several steps are necessary:
  add three new rows, labels, update formulas, etc.
- Very prone to errors.

An Example - Changing the Model

- For expenses before and after tax, additional columns need to be inserted in the block of each year.

Cost	Tax tariff	After tax	Total
200	0,12	224	400
100	0,2	120	500
20	0,2	24	80
			980

- The user needs to change all the year in the spreadsheet.

ClassSheets: Specifying Spreadsheets

- Erwig and Engels have introduced *ClassSheets* to specify spreadsheets;
- *ClassSheets* allow to express business object structures within a spreadsheet using concepts from the OO paradigm;

	A	B	D	E	F	...	G
1	**Budget**		**Year**				
2			year=2010				
3	**Category**	Name	Qnty	Cost	Total		Total
4		name="abc"	qnty=0	cost=0	total=qnty*cost		total=SUM(total)
⋮							
5	Total				total=SUM(total)		total=SUM(**Year**.total)

Modeling our Example

- Class to represent a year;
- Class to represent budget line;
- Class to represent the relationship between them.

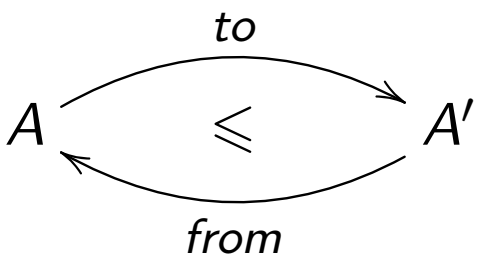
	A	B	D	E	F	...	G
1	**Budget**		**Year**				
2			year=2010				
3	**Category**	Name	Qnty	Cost	Total		Total
4		name="abc"	qnty=0	cost=0	total=qnty*cost		total=SUM(total)
⋮							
5	Total				total=SUM(total)		total=SUM(**Year**.total)

Cost	Tax tariff	After tax	Total
cost=0	tax tariff=0	after tax=cost+ cost*tax tariff	total=qnty*cost
			total=SUM(total)

Co-evolution of Model and Instances

- We have a model to specify spreadsheets: *ClassSheets*;
- And we have the instances: spreadsheet data;
- We need to synchronize them;
- We use data refinements.

Two-Level Transformation (2LT)



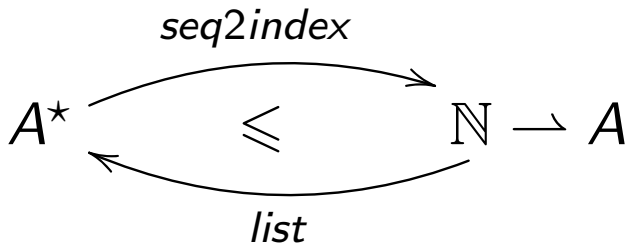
A, A'	data type and transformed data type
to	witness function of type $A \to A'$ (injective and entire relation)
from	witness function of type $A' \to A$ (surjective, possibly partial)

$from \circ to = id_A$

- Implementation of data refinements theory;
- Two-level rewiring system;
- Transforming a type into another also produces data migration functions;
- We can compose refinements.

2LT Rules - An Example

Maps from natural numbers to some type, $\mathbb{N} \rightharpoonup A$, are the implementation of lists of that type, $A^\star$:



- *seq2index* creates a map where the keys are the indexes of the elements of the list;
- *list* collects the elements in the map.

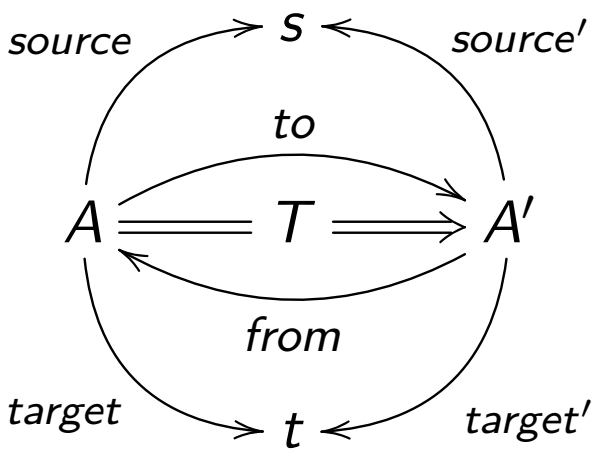
$$\mathit{seq2index}\ [\texttt{'a'},\texttt{'z'},\texttt{'x'}] = \{1 \mapsto \texttt{'a'}, 2 \mapsto \texttt{'z'}, 3 \mapsto \texttt{'x'}\}$$
$$\mathit{list}\ \{1 \mapsto \texttt{'a'}, 2 \mapsto \texttt{'z'}, 3 \mapsto \texttt{'x'}, 4 \mapsto \texttt{'k'}\} = [\texttt{'a'},\texttt{'z'},\texttt{'x'},\texttt{'k'}]$$

Spreadsheet Models in 2LT

- 2LT designed to algebraic data type;
- We have created a representation for spreadsheet models based on *ClassSheets*;
- Reused the 2LT framework: when specifying a model transformation, we get for free functions to migrate data back and forward.

Spreadsheet Models in 2LT - References

- References pose a particular challenge;
- We implemented them as a pair of selection functions: one selects the cell which is the reference and another select the referenced cell.



source	Projection over type A identifying the reference
target	Projection over type A identifying the referenced cell

$$source' = source \circ from$$
$$target' = target \circ from$$

Evolution Rules

- Combinators: defined as helper rules
- Semantic: rules that add information to the model
- Layout: rules that do not add information to the model, just change its arrangement

Combinators - Pull up all the references

- All references must be at the top level;
- For example,

$$A \times B_{\phi} \text{ becomes } (A \times B)_{\phi}$$

Combinators - Apply after and friends

- Applies another rules after something (*e.g.*, a label, a block)
- For example,

$$\mathit{after}\ \texttt{"Cost"}\ (\mathit{addColumn}\ \texttt{"New Column"}\ 0)$$

Semantics - Insert a column

- A column is a composed by a label `"l"` and a default value v:

$$insertCol\ \texttt{"l"}\ v$$

- The references are projection function;
- We can not reference the exiting spreadsheet;
- The formula is inserted as undefined and is correctly set after:

$$(once\ (insertCol\ \texttt{"After Tax"}\ FRef)) \triangleright$$
$$(setFormula\ auxType\ fromRef\ \ toRef)$$

Semantics - Make it expandable

- Allows some part to be added more columns/rows;
- An example:

$$b \text{ becomes } b^{\rightarrow} \text{ or } b^{\downarrow}$$

Semantics - Split

- Moves a column to a new location;
- The old value columns are substituted by references to the new position;
- Similar to create a pointer;
- The new column does not contain repeated values.

Layout - Change orientation

- It changes the entire spreadsheet orientation from vertical to horizontal;
- And vice versa.

Layout - Normalize blocks

- Some results are not well formated:
- A common example is to have as result the following type:

$A \mid B\hat{}\, C \mid D\hat{}$
$E \mid F$

Most of the times, the correct result is the following:

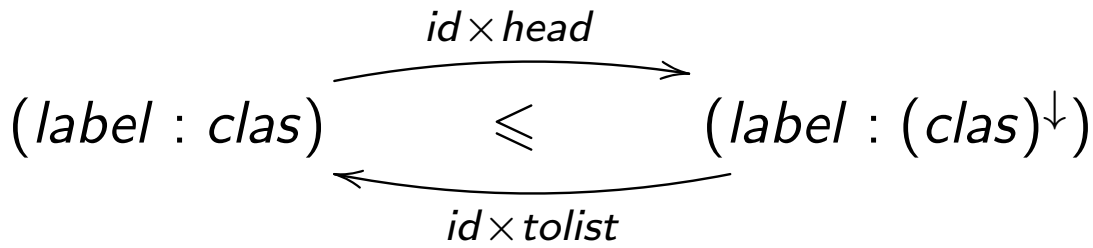
$A \mid B \mid D\hat{}$
$E \mid C \mid F$

Layout - Shift

- Shift vertically or horizontally;
- It moves the required part of the spreadsheet in the desired direction;
- It leaves the old place empty.

Make It Expandable (in Haskell)

It is possible to make a block expandable:



Its implementation is as follows:

$$
\begin{aligned}
&\mathit{expandBlock} :: \mathit{String} \to \mathit{Rule}\\
&\mathit{expandBlock}\ \mathit{str}\ (\mathit{label} : \mathit{clas}) \mid \mathit{compLabel}\ \mathit{label}\ \mathit{str} = \mathbf{do}\\
&\qquad \mathbf{let}\ \mathit{rep} = \mathit{Rep}\ \{\mathit{to} = \mathit{id} \times \mathit{tolist}, \mathit{from} = \mathit{id} \times \mathit{head}\}\\
&\qquad \mathit{return}\ (\mathit{View}\ \mathit{rep}\ (\mathit{label} : (\mathit{clas})^{\downarrow}))
\end{aligned}
$$

Conclusions

- We have created a safe representation of spreadsheet models (and instances);
- Our techniques handle references and formulas;
- We have shown rules for coupled evolution of models and instances;
- We wish to make this available for spreadsheet users;
- Thus, we are working on an embedding of our techniques in *OpenOffice.org* (submitted paper).