Calculating from Alloy relational models

J.N. Oliveira

Dept. Informática,
Universidade do Minho
Braga, Portugal

IFIP WG2.1 meeting #64
March-April 2009
Weltenburg, Germany

Model driven engineering

- **MEDEA** project — *High Assurance MDE using Alloy*
- **MDE** is a clumsy area of work, full of approaches, acronyms, notations.
- **UML** has taken the lead in *unifying* such notations, but it is too **informal** to be accepted as a reference approach.
- Model-oriented formal methods (**VDM**, **Z**) solve this informality problem at a high-cost: people find it hard to understand models written in maths (cf. maths illiteracy if not mathphobic behaviour).
- **Alloy** [2] offers a good compromise — it is formal in a light-weight manner.

Inspiration

- **BBI** project [3]: **Alloy** re-engineering of a well-tested, very well written non-trivial prototype in **Haskell** of a real-estate trading system similar to the stocks market (65 pages in lhs format) unveiled 4 bugs (2 invariant violations + 2 weak pre-conditions)
- Alloy and Haskell complementary to each other

Real Estate Exchange

Bolsa de Bens Imobiliários

PortoDigital – SEC-11

Joost Visser

Confidential

Draft of August 19, 2007

Alloy

What **Alloy** offers

- A unified approach to **modeling** based on the notion of a **relation — “everything is a relation”** in Alloy.
- A minimal syntax (centered upon relational composition) with an object-oriented flavour which captures much of what otherwise would demand for **UML+OCL**.
- A **pointfree** subset.
- A model-checker for model assertions (counter-examples within scope).

Alloy

What **Alloy does not** offer

- Complete calculus for deduction (proof theory)
- Strong type checking
- Dynamic semantics modeling features

Opportunities

- Enrich the standard Alloy *modus operandi* with relational algebra calculational proofs
- Design an Alloy-centric tool-chain for high assurance model-oriented design

Thus the **MEDEA** project (submitted).

Alloy

What **Alloy does not** offer

- Complete calculus for deduction (proof theory)
- Strong type checking
- Dynamic semantics modeling features

Opportunities

- Enrich the standard Alloy *modus operandi* with relational algebra calculational proofs
- Design an Alloy-centric tool-chain for high assurance model-oriented design

Thus the **MEDEA** project (submitted).

Relational composition

- The swiss army knife of Alloy
- It subsumes function application and "field selection"
- Encourages a navigational (point-free) style based on pattern $x.(R.S)$.
- Example:

$$
\begin{aligned}
Person &= \{(P1),(P2),(P3),(P4)\}\\
parent &= \{(P1,P2),(P1,P3),(P2,P4)\}\\
me &= \{(P1)\}\\
me.parent &= \{(P2),(P3)\}\\
me.parent.parent &= \{(P4)\}\\
Person.parent &= \{(P2),(P3),(P4)\}
\end{aligned}
$$

When "everything is a relation"

- Sets are relations of arity 1, eg. $Person = \{(P1), (P2), (P3), (P4)\}$
- Scalars are relations with size 1, eg. $me = \{(P1)\}$
- Relations are first order, but we have multi-ary relations.
- However, **Alloy** relations are not *n*-ary in the usual sense: instead of thinking of $R \in 2^{A \times B \times C}$ as a set of triples (there is no such thing as *tupling* in Alloy), think of R in terms of *currying*:

$$R \in (B \to C)^A$$

(More about this later.)

Kleene algebra flavour

Basic operators:

.	*composition*
+	*union*
^	*transitive closure*
*	*transitive-reflexive closure*

(There is no recursion is Alloy.) Other relational operators:

~	*converse*
++	*override*
&	*intersection*
-	*difference*
->	*cartesian product*
<:	*domain restriction*
:>	*range restriction*

Relational thinking

- As a rule, thinking in terms of poinfree relations (this includes **functions**, of course) pays the effort: the concepts and the reasoning become simpler.
- This includes **relational data** structuring, which is far more interesting than what can be found in SQL and relational databases.

Example — list processing

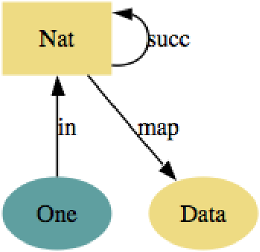
- **Lists** are traditionally viewed as recursive (linear) data structures.
- There are no lists in Alloy — they have to be modeled by **simple** relations (vulg. partial functions) between indices and elements.

Lists as relations in Alloy

```
sig List {
        map : Nat -> lone Data
}

sig Nat {
    succ: one Nat
}

one sig One in Nat {}
```



Multiplicities: `lone` (one or less), `one` (exaclty one)

Relational data structuring

Some correspondences:

list l	**relation** L
sorted	monotonic
noDuplicates	injective
$map\ f\ l$	$f \cdot L$
$zip\ l_1\ l_2$	$\langle L_1, L_2 \rangle$
$[1, \ldots]$	id

where

- id is the identity (equivalence) relation
- the "fork" (also known as "split") combinator is such that $(x, y)\langle L_1, L_2 \rangle z$ means the same as $xL_1z \wedge yL_2z$

Haskell versus Alloy

Pointwise Haskell:

```
findIndices        :: (a -> Bool) -> [a] -> [Int]
findIndices p xs = [ i | (x,i) <- zip xs [0..], p x ]
```

Pointfree (PF):

$$findIndices\ p\ L \quad \triangleq \quad \pi_2 \cdot (\Phi_p \times id) \cdot \langle L, id \rangle \tag{1}$$

where

- π_2 is the right projection of a pair
- $L \times R = \langle L \cdot \pi_1, R \cdot \pi_2 \rangle$
- $\Phi_p \subseteq id$ is the coreflexive relation (partial identity) which models predicate p (or a set)

Haskell versus Alloy

- What about Alloy? It has no pairs, therefore no forks $\langle L, R \rangle \ldots$
- Fortunately there is the relational calculus:

$$
\begin{array}{ll}
 & \pi_2 \cdot (\Phi_p \times id) \cdot \langle L, id \rangle \\
\Leftrightarrow & \quad \{ \ \times\text{-absorption} \ \} \\
 & \pi_2 \cdot \langle \Phi_p \cdot L, id \rangle \\
\Leftrightarrow & \quad \{ \ \times\text{-cancelation} \ \} \\
 & \delta\,(\Phi_p \cdot L)
\end{array}
$$

where $\delta\, R = R^\circ \cdot R \cap id$, for R° the converse of R.

Haskell versus Alloy

Two ways of writing $\delta\,(\Phi_p \cdot L)$ in Alloy, one pointwise

```
fun findIndices[s:set Data,l:List]: set Nat {
      {i: Nat | some x:s | x in i.(l.map)}
}
```

and the other pointfree,

```
fun findIndices[s:set Data,l:List]: set Nat {
      dom[l.map :> s]
}
```

the latter very close to what we've calculated.

Beyond model-checking: proofs by calculation

Suppose the following property

$$(\mathit{findIndices}\ p) \cdot r^{\star} \quad = \quad \mathit{findIndices}\ (p \cdot r) \tag{2}$$

is asserted in Alloy:

```
assert FT {
    all l,l':List, p: set Data, r: Data -> one Data |
        l'.map = l.map.r =>
            findIndices[p,l'] = findIndices[r.p,l]
}
```

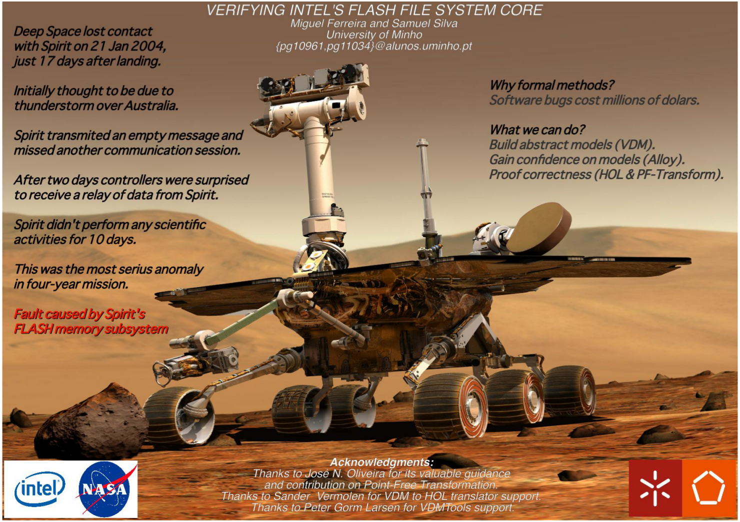
and that the model checker does not yield any counter-examples. How can we be sure of its validity?

- Free theorems — the given assertion is a corollary of the free theorem of *findIndices*, thus there is nothing to prove (model checking could be avoided!)
- Wishing to prove the assertion anyway, one calculates:

Trivial proof

$$
\begin{array}{ll}
 & (\mathit{findIndices}\ p) \cdot r^{\star} = \mathit{findIndices}\ (p \cdot r) \\
\Leftrightarrow & \quad \{ \text{ list to relation transform } \} \\
 & \delta\,(\Phi_p \cdot (r \cdot L)) = \delta\,(\Phi_{p \cdot r} \cdot L) \\
\Leftrightarrow & \quad \{ \text{ property } \Phi_{f \cdot g} = \delta\,(\Phi_f \cdot g) \ \} \\
 & \delta\,(\Phi_p \cdot (r \cdot L)) = \delta\,(\delta\,(\Phi_p \cdot r) \cdot L) \\
\Leftrightarrow & \quad \{ \text{ domain of composition } \} \\
 & \delta\,(\Phi_p \cdot (r \cdot L)) = \delta\,((\Phi_p \cdot r) \cdot L) \\
\Leftrightarrow & \quad \{ \text{ associativity } \} \\
 & \textsc{True}
\end{array}
$$

Realistic example — Verified FSystem (VFS)



VFS in Alloy (simplified)

The system:

```
sig System {
  fileStore: Path -> lone File,
  table: FileHandle -> lone OpenFileInfo
}
```

Paths:

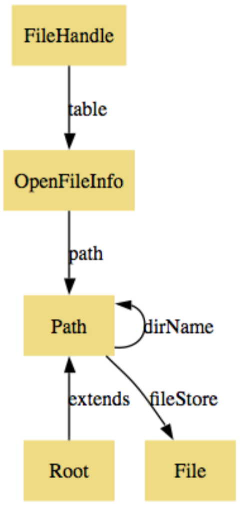
```
sig Path {
  dirName: one Path
}
```

The root is a path:

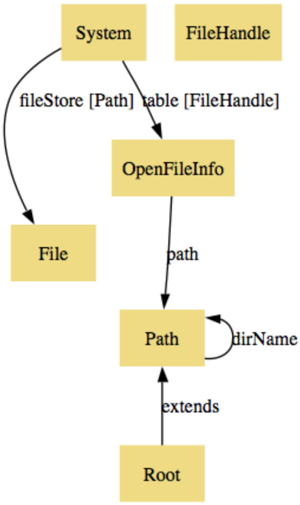
```
one sig Root extends Path {
}
```

Alloy diagrams for FSystem

Simplified:



Complete:



Binary relation semantics

Meaning of signatures:

```
sig Path {
  dirName: one Path
}
```

declares function $Path \xrightarrow{dirName} Path$.

```
sig System {
  fileStore: Path -> lone File,
}
```

declares simple relation $System \times Path \xrightarrow{fileStore} File$.
(**NB**: a relation S is **simple**, or *functional*, wherever its **image** $S \cdot S^\circ$ is coreflexive. Using harpoon arrows $\rightharpoonup$ for these.)

Binary relation semantics

- Since

$$(A \times B) \rightharpoonup C \quad \cong \quad (B \rightharpoonup C)^A$$

$\mathit{fileStore}$ can be alternatively regarded as a function in $(\mathit{Path} \rightharpoonup \mathit{File})^{\mathit{System}}$, that is, for $s : \mathit{System}$,

$$\mathit{Path} \xrightharpoonup{(\mathit{fileStore}\ \ s)} \mathit{File}$$

- Thus the "navigation-styled" notation of Alloy: $p.(s.\mathit{fileStore})$ means the file accessible from path p in file system s.
- Similarly, line `table: FileHandle -> lone OpenFileInfo` in the model declares

$$\mathit{FileHandle} \xrightharpoonup{(\mathit{table}\ \ s)} \mathit{OpenFileInfo}$$

Multiplicities in Alloy + taxonomy

A lone -> B	A -> some B	A -> lone B	A some -> B
injective	entire	simple	surjective

A lone -> some B	A -> one B	A some -> lone B
representation	function	abstraction

A lone -> one B	A some -> one B
injection	surjection

A one -> one B
bijection

(courtesy of Alcino Cunha, the Alloy expert at Minho)

Terminology reminder

Topmost criteria:

binary relation

injective entire simple surjective

Definitions:

	Reflexive	*Coreflexive*
$\ker R$	entire R	injective R
$\mathrm{img}\, R$	surjective R	simple R

$$\ker R = R^{\circ} \cdot R$$
$$\mathrm{img}\, R = R \cdot R^{\circ}$$

From Alloy to relational diagrams

$$
\begin{tikzcd}
\color{red}{OpenFileDescriptor} \arrow[d, "path"'] & & \color{red}{FileHandle} \arrow[ll, "\color{red}{table\ \ s}"'] \\
\color{red}{Path} \arrow[rr, "\color{red}{fileStore\ \ s}"'] & & \color{red}{File} \\
Path \arrow[u, "dirName"] & &
\end{tikzcd}
$$

where

- *table s*, *fileStore s* are simple relations
- the other arrows depict functions

(diagram in the **Rel** allegory to be completed)

Model constraints

Referential integrity:

Non-existing files cannot be opened:

```
pred ri[s: System]{
    all h: FileHandle, o: h.(s.table) |
                      some (o.path).(s.fileStore)
}
```

Paths closure:

Mother directories exist and are indeed directories:

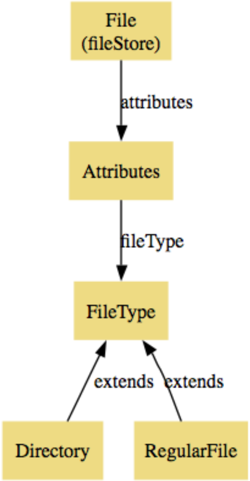
```
pred pc[s: System]{
  all p: Path |
        some p.(s.fileStore) =>
        (some d: (p.dirName).(s.fileStore) |
                                 d.fileType=Directory)
}
```

2nd part of Alloy FSystem model

```
sig File {
    attributes: one Attributes
}

sig Attributes{
    fileType: one FileType
}

abstract sig FileType {}
one sig RegularFile extends FileTy
one sig Directory extends FileType
```



Updated binary relational diagram

$$
\begin{tikzcd}
OpenFileDescriptor \arrow[d, "path"'] & & FileHandle \arrow[ll, "table\ \ s"'] & \\
Path \arrow[rr, "fileStore\ \ s"'] & & File \arrow[r, "attributes"] & Attributes \arrow[d, "fileType"] \\
Path \arrow[u, "dirName"] & & & FileType
\end{tikzcd}
$$

where

- *table s*, *fileStore s* are simple relations
- all the other arrows depict functions

Constraints: still missing

Updating diagram with constraints

Complete diagram, where M abbreviates *table s*, N abbreviates *fileStore s* and $\underline{k}$ is the "everywhere-k" function:

$$
\begin{tikzcd}
OpenFileDescriptor \arrow[d, "path"'] \arrow[dr, phantom, "\subseteq"] & FileHandle \arrow[l, "M"'] \arrow[d, "\top"] & \\
Path \arrow[r, shift right, "N"'] & File \arrow[l, shift right, "N^{\circ}"'] \arrow[r, "attributes"] \arrow[d, phantom, "\rotatebox{90}{$\subseteq$}"] & Attributes \arrow[d, "fileType"] \\
Path \arrow[u, "dirName"] \arrow[r, "N"'] & File \arrow[r, "\underline{Directory}"'] & FileType
\end{tikzcd}
$$

Constraints:

- Top rectangle is the PF-transform of *ri* (referential integrity)
- Bottom rectangle is the PF-transform of *pc* (path closure)

PF-constraints in symbols

Referential integrity:

$$ri(M, N) \quad \triangleq \quad path \cdot M \subseteq N^{\circ} \cdot \top \tag{3}$$

which is equivalent to

$$ri(M, N) \quad \triangleq \quad \rho\,(path \cdot M) \subseteq \delta\, N$$

where $\rho\, R = \delta\, R^{\circ}$. PF version (3) also easy to encode in Alloy

```
pred riPF[s: System]{
    s.table.path in (FileHandle->File).~(s.fileStore)
}
```

thanks to its emphasis on **composition**.

PF-constraints in symbols

Referential integrity:

$$ri(M, N) \triangleq path \cdot M \subseteq N^{\circ} \cdot \top \quad (3)$$

which is equivalent to

$$ri(M, N) \triangleq \rho\,(path \cdot M) \subseteq \delta\, N$$

where $\rho\, R = \delta\, R^{\circ}$. PF version (3) also easy to encode in Alloy

```
pred riPF[s: System]{
    s.table.path in (FileHandle->File).~(s.fileStore)
}
```

thanks to its emphasis on **composition**.

PF-constraints in symbols

Paths closure:

$$pc\ N \triangleq \underline{Directory} \cdot N \subseteq fileType \cdot attributes \cdot N \cdot dirName \qquad (4)$$

recall diagram:

$$\begin{tikzcd}
\color{red}{Path} \arrow[r, "\color{red}{N}"'] & \color{red}{File} \arrow[r, "attributes"] & Attributes \arrow[d, "fileType"] \\
Path \arrow[u, "dirName"] \arrow[r, "N"'] & File \arrow[u, phantom, "\cup\!|"] \arrow[r, "\underline{Directory}"'] & FileType
\end{tikzcd}$$

Again thanks to emphasis on **composition**, this is easily encoded in PF-Alloy:

```
pred pcPF[s: System]{
   s.fileStore.(File->Directory) in
      dirName.(s.fileStore).attributes.fileType
}
```

PF-constraints in symbols

Paths closure:

$$pc\ N \triangleq \underline{Directory} \cdot N \subseteq fileType \cdot attributes \cdot N \cdot dirName \qquad (4)$$

recall diagram:

$$\begin{tikzcd}
Path \arrow[r, "N"'] & File \arrow[r, "attributes"] & Attributes \arrow[d, "fileType"] \\
Path \arrow[u, "dirName"] \arrow[r, "N"'] & File \arrow[u, phantom, "\rotatebox{90}{$\subseteq$}"] \arrow[r, "\underline{Directory}"'] & FileType
\end{tikzcd}$$

Again thanks to emphasis on **composition**, this is easily encoded in PF-Alloy:

```
pred pcPF[s: System]{
   s.fileStore.(File->Directory) in
       dirName.(s.fileStore).attributes.fileType
}
```

PF-ESC by calculation

- Models with constraints put the burden on the designer to ensure that operations **type-check** (read this in **extended**-mode), that is, constraints are preserved across the models operations.
- Typical approach in MDE: **model-checking**
- Automatic **theorem proving** also considered in safety-critical systems
- However: convoluted pointwise formulæ often lead to failure.

How about doing these as "pen & paper" exercises?

- **PF-formulæ** are manageable, this is the difference.

Example of PF-ESC by calculation

Consider the operation which removes file system objects, as modeled in Alloy:

```
pred delete[s',s: System, sp: set Path]{
        s'.table = s.table
        s'.fileStore = (univ-sp) <: s.fileStore
}
```

that is,

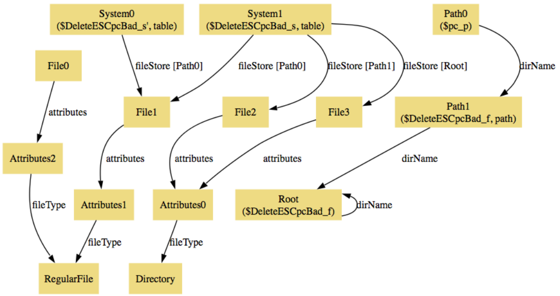
$$delete\ S\ (M, N) \quad \triangleq \quad (M, N \cdot \Phi_{(\notin S)}) \tag{5}$$

where $\Phi_{(\notin S)}$ is the coreflexive associated to the complement of S.

Intuitive steps

Intuitively, *delete* will put the

- *ri* constraint at risk once we decide to delete file system objects which are open
- *pc* constraint at risk once we decide to delete directories with children.



(Model-checking in **Alloy** will easily spot these flaws, as checked above by a counter-example for the latter situation.)

Intuitive steps

We have to guess a **pre-conditions** for *delete*. However,

- How can we be sure that such (guessed) pre-condition is *good enough*?
- The best way is to calculate the weakest pre-condition for each constraint to be maintained.
- In doing this, mind the following properties of relational algebra:

$$h \cdot R \subseteq S \quad \Leftrightarrow \quad R \subseteq h^{\circ} \cdot S \tag{6}$$

$$R \cdot \Phi \quad = \quad R \cap \top \cdot \Phi \tag{7}$$

$$f \cdot R \subseteq \top \cdot S \quad \Leftrightarrow \quad R \subseteq \top \cdot S \tag{8}$$

For improved readability, we introduce abbreviations $ft := fileType \cdot attributes$ and $d := \underline{Directory}$, and **calculate**:

Calculational steps

$$
\begin{array}{ll}
 & pc(delete\ S\ (M, N)) \\
\Leftrightarrow & \quad \{\ (5) \text{ and } (4)\ \} \\
 & d \cdot (N \cdot \Phi_{(\notin S)}) \subseteq ft \cdot (N \cdot \Phi_{(\notin S)}) \cdot dirName \\
\Leftrightarrow & \quad \{\ \text{shunting } (6)\ \} \\
 & d \cdot N \cdot \Phi_{(\notin S)} \cdot dirName^{\circ} \subseteq ft \cdot N \cdot \Phi_{(\notin S)} \\
\Leftrightarrow & \quad \{\ (7)\ \} \\
 & d \cdot N \cdot \Phi_{(\notin S)} \cdot dirName^{\circ} \subseteq ft \cdot N \cap \top \cdot \Phi_{(\notin S)} \\
\Leftrightarrow & \quad \{\ \cap\text{-universal} ; \text{shunting}\ \}
\end{array}
$$

Ensuring paths closure

$$\left\{ \begin{array}{l} d \cdot N \cdot \Phi_{(\notin S)} \subseteq ft \cdot N \cdot dirName \\ d \cdot N \cdot \Phi_{(\notin S)} \subseteq \top \cdot \Phi_{(\notin S)} \cdot dirName \end{array} \right.$$

$$\Leftrightarrow \qquad \{ \ \top \text{ absorbs } d \ (8) \ \}$$

$$\left\{ \begin{array}{l} \underbrace{d \cdot N \cdot \Phi_{(\notin S)} \subseteq ft \cdot N \cdot dirName}_{\text{weaker than } pc(N)} \\ \underbrace{N \cdot \Phi_{(\notin S)} \subseteq \top \cdot \Phi_{(\notin S)} \cdot dirName}_{wp} \end{array} \right.$$

Back to points, *wp* is:

$$\langle \forall\ q \ :\ q \in dom\ N \wedge q \notin S :\ dirName\ q \notin S \rangle$$

$$\Leftrightarrow \qquad \{ \ \text{predicate logic} \ \}$$

$$\langle \forall\ q \ :\ q \in dom\ N \wedge (dirName\ q) \in S :\ q \in S \rangle$$

Ensuring paths closure

In words:

> *if parent directory of existing path q is marked for deletion than so must be q.*

Translating calculated weakest precondition back to Alloy:

```
pred pre_delete[s: System, sp: set Path]{
     all q: Path |
        some q.(s.fileStore) &&
             q.dirName in sp => q in sp
}
```

Back to the diagram

PF-encoding of model constraints in terms of relational composition has at least the following advantages:

- it makes **calculations** easier (rich algebra of $R \cdot S$)
- it makes it possible to **draw** constraints as rectangles in diagrams, recall

$$
\begin{tikzcd}
OpenFileDescriptor \arrow[d, "path"'] & & FileHandle \arrow[ll, "M"'] \arrow[d, "\top"] \arrow[dll, phantom, "\subseteq"] & \\
Path \arrow[rr, shift right, "N"'] & & File \arrow[ll, shift right, "N^\circ"'] \arrow[r, "attributes"] \arrow[d, phantom, "\cup\!|"] & Attributes \arrow[d, "fileType"] \\
Path \arrow[u, "dirName"] \arrow[rr, "N"'] & & File \arrow[r, "\underline{Directory}"'] & FileType
\end{tikzcd}
$$

- it enables the "navigation-styled" notation of Alloy

Constraint bestiary

- Experience in formal modeling tells that designs are **repetitive** in the sense that they instantiate (**generic**) constraints whose ubiquitous nature calls for classification
- Such "**constraint patterns**" are rectangles, thus easy to draw and recall
- In the next slides we browse a little "constraint bestiary" capturing some typical samples.

Constraints are Rectangles

- All of shape

$$R \cdot I \quad \subseteq \quad O \cdot R$$

- Example: **referential integrity** in general, where N is the *offer* and M is the *demand* :

$$\rho\,(\in_{\mathsf{F}} \cdot M) \subseteq \delta\, N \quad \Leftrightarrow \quad \begin{tikzcd} \mathsf{F}\,B \arrow[d, "\in_{\mathsf{F}}"'] \arrow[dr, phantom, "\subseteq"] & A \arrow[l, "M"'] \arrow[d, "\top"] \\ B \arrow[r, shift right, "N"'] & C \arrow[l, shift right, "N^{\circ}"'] \end{tikzcd}$$

$$\Leftrightarrow \quad \in_{\mathsf{F}} \cdot M \subseteq N^{\circ} \cdot \top$$

M, N simple. $\in_{\mathsf{F}}$ is a membership relation.

Constraints are Rectangles

- **Example:** M, N domain-disjoint

$$M \cdot N^{\circ} \quad \subseteq \quad \bot$$

- **Example:** simple M, N domain-coherent

$$M \cdot N^{\circ} \quad \subseteq \quad id$$

- **Example:** M domain-closed by R:

$$M \cdot R^{\circ} \subseteq \top \cdot M$$

(path-closure constraint instance of this)

- **Example:** range of R in Φ

$$R \subseteq \Phi \cdot R$$

Experience and Current work

- Defining a simple pointfree **binary** relational semantics for **Alloy** [1]
- Studying the translation to/from Haskell and, in particular, how to port counterexamples to QuickCheck.
- Designing an Alloy-centric **tool-chain** including a (pointfree) extended static checker, translators to Haskell, UML and SQL.

Closing

Why the **UML+OCL**? Why **ERDs**?

- What one draws in UML and ERDs can be captured by binary relational diagrams — not only the class/entity attributive structure + relationships but also the constraints which one normally can't depict at all
- Drawing a constraint as a rectangle means it's well understood, and that calculations will be easier to carry out (run away from logical $\wedge$ if you can!)
- Rectangles nicely encoded in plain PF-Alloy or hybrid navigation-styled Alloy

As Alan Perlis once wrote down:

"Simplicity does not precede complexity, but follows it."

Marcelo F. Frias, Carlos G. Lopez Pombo, Gabriel A. Baum, Nazareno M. Aguirre, and Thomas S.E. Maibaum.
Reasoning about static and dynamic properties in alloy: A purely relational approach.
ACM Trans. Softw. Eng. Methodol., 14(4):478–526, 2005.

D. Jackson.
Software abstractions: logic, language, and analysis.
The MIT Press, Cambridge Mass., 2006.
ISBN 0-262-10114-9.

Joost Visser.
Real estate exchange.
Technical report, DI/UM , Braga, Jan 2007.
PortoDigital – SEC-11. Confidential.