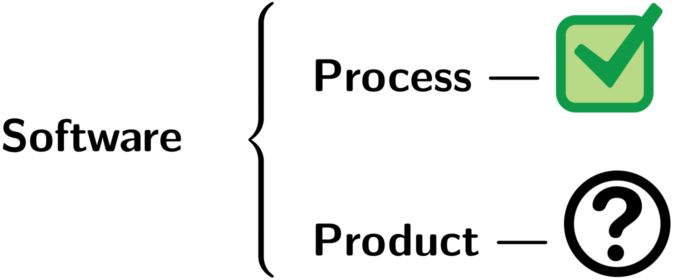
CSI - A Calculus for Information Systems (2025/26)

Class 1

Global picture

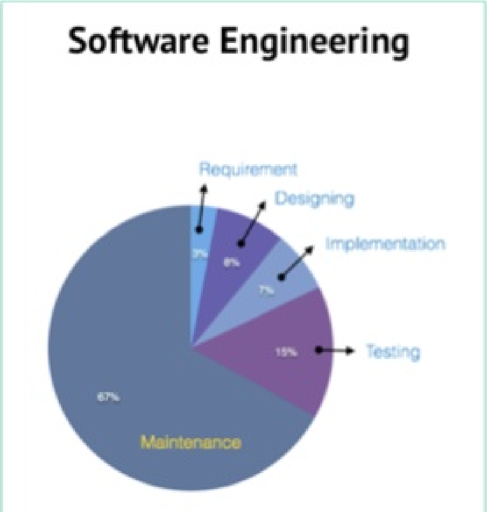
Concerning software 'engineering':

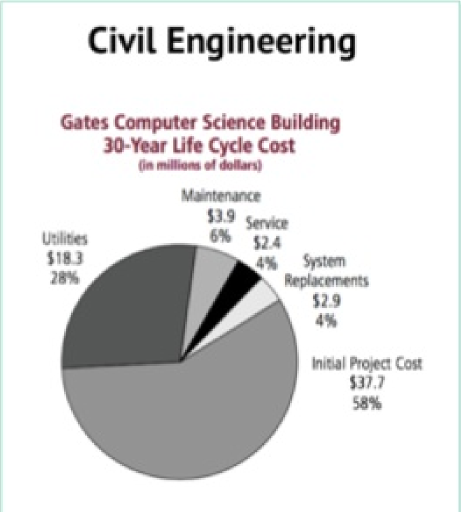


Formal methods provide an answer to the question mark above.

Global picture

Concerning software 'engineering':





Have you ever used a FM?

Of course you have! Check this:

A problem

My three children were born at a 3 year interval rate. Altogether, they are as old as me. I am 48. How old are they?

A model

$$x + (x + 3) + (x + 6) = 48$$

— maths description of the problem.

Some calculations

$$
\begin{array}{l}
3x + 9 = 48 \\
\equiv \quad \{ \text{ "al-djabr" rule } \} \\
3x = 48 - 9 \\
\equiv \quad \{ \text{ "al-hatt" rule } \} \\
x = 16 - 3
\end{array}
$$

The solution

$$
\begin{array}{rcl}
x & = & 13 \\
x + 3 & = & 16 \\
x + 6 & = & 19
\end{array}
$$

Have you ever used a FM?

Of course you have! Check this:

A problem

My three children were born at a 3 year interval rate. Altogether, they are as old as me. I am 48. How old are they?

A model

$$x + (x + 3) + (x + 6) = 48$$

— maths description of the problem.

Some calculations

$$
\begin{aligned}
& 3x + 9 = 48 \\
\equiv & \quad \{ \text{"al-djabr" rule} \} \\
& 3x = 48 - 9 \\
\equiv & \quad \{ \text{"al-hatt" rule} \} \\
& x = 16 - 3
\end{aligned}
$$

The solution

$$
\begin{aligned}
x &= 13 \\
x + 3 &= 16 \\
x + 6 &= 19
\end{aligned}
$$

Have you ever used a FM?

Of course you have! Check this:

A problem

My three children were born at a 3 year interval rate. Altogether, they are as old as me. I am 48. How old are they?

A model

$$x + (x + 3) + (x + 6) = 48$$

— maths description of the problem.

Some calculations

$$3x + 9 = 48$$

$\equiv$ { "al-djabr" rule }

$$3x = 48 - 9$$

$\equiv$ { "al-hatt" rule }

$$x = 16 - 3$$

The solution

$$\begin{aligned} x &= 13 \\ x + 3 &= 16 \\ x + 6 &= 19 \end{aligned}$$

Have you ever used a FM?

Of course you have! Check this:

A problem

My three children were born at a 3 year interval rate. Altogether, they are as old as me. I am 48. How old are they?

A model

$$x + (x+3) + (x+6) = 48$$

— maths description of the problem.

Some calculations

$$\begin{array}{cl} & 3x + 9 = 48 \\ \equiv & \{\ \text{"al-djabr" rule}\ \} \\ & 3x = 48 - 9 \\ \equiv & \{\ \text{"al-hatt" rule}\ \} \\ & x = 16 - 3 \end{array}$$

The solution

$$\begin{array}{rcl} x & = & 13 \\ x+3 & = & 16 \\ x+6 & = & 19 \end{array}$$

Have you ever used a FM?

"Al-djabr" rule ? "al-hatt" rule ?

al-djabr

$$x - z \leq y \equiv x \leq y + z$$

all-hatt

$$x * z \leq y \equiv x \leq y * z^{-1} \qquad (z > 0)$$

These rules that you have used so many times were discovered by Persian mathematicians, notably by Al-Huwarizmi (9c AD).

NB: "**algebra**" stems from "**al-djabr**" and "**algarismo**" from **Al-Huwarizmi**.

Software problems

Now, suppose the **problem** was

Please write a program to list the students of my class ordered by their marks.

Is there a mathematical **model** for this problem?

Yes, of course there is — see aside:

$$sort \subseteq \frac{bag}{bag} \cap \frac{true}{sorted}$$

where

$sorted = \ldots marks \ldots$

$bag = \ldots.$

But,

- what do $X \cap Y$, $\frac{f}{g}$ ... mean here?
- Is there an "**algebra**" for such symbols?

Yes — Wait and see :-)

Software problems

Now, suppose the **problem** was

Please write a program to list the students of my class ordered by their marks.

Is there a mathematical **model** for this problem?

Yes, of course there is — see aside:

$$sort \subseteq \frac{bag}{bag} \cap \frac{true}{sorted}$$

where

$sorted = \ldots marks \ldots$

$bag = \ldots.$

But,

- what do $X \cap Y$, $\frac{f}{g}$ ... mean here?
- Is there an "**algebra**" for such symbols?

Yes — Wait and see :-)

Software problems

Now, suppose the **problem** was

> *Please write a program to list the students of my class ordered by their marks.*

Is there a mathematical **model** for this problem?

Yes, of course there is — see aside:

$$sort \subseteq \frac{bag}{bag} \cap \frac{true}{sorted}$$

where

$$sorted = \ldots marks \ldots$$

$$bag = \ldots.$$

But,

- what do $X \cap Y$, $\frac{f}{g}$ ... mean here?
- Is there an "**algebra**" for such symbols?

Yes — Wait and see :-)

Software problems

Now, suppose the **problem** was

Please write a program to list the students of my class ordered by their marks.

Is there a mathematical **model** for this problem?

Yes, of course there is — see aside:

$$sort \subseteq \frac{bag}{bag} \cap \frac{true}{sorted}$$

where

$$sorted = \ldots marks \ldots$$

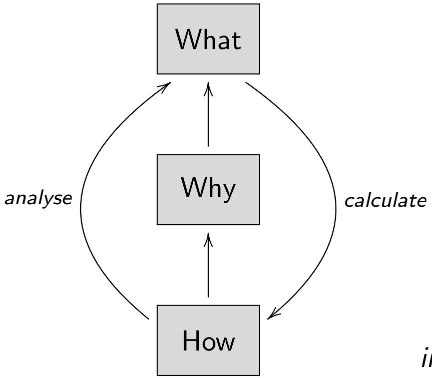
$$bag = \ldots.$$

But,

- what do $X \cap Y$, $\frac{f}{g}$ ... mean here?
- Is there an "**algebra**" for such symbols?

Yes — Wait and see :-)

FM — scientific software design

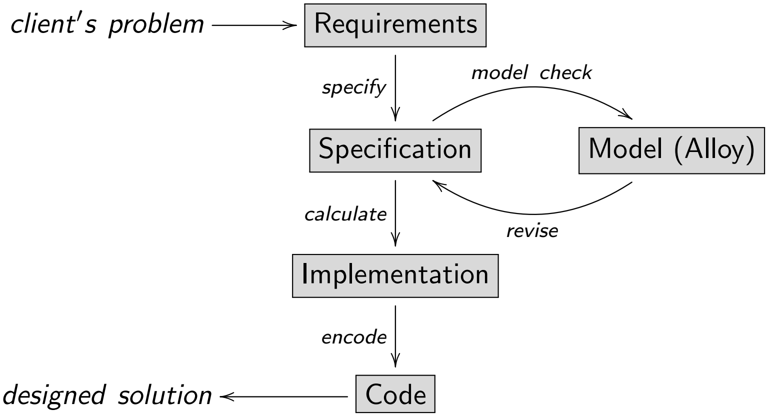


specification (*model*)

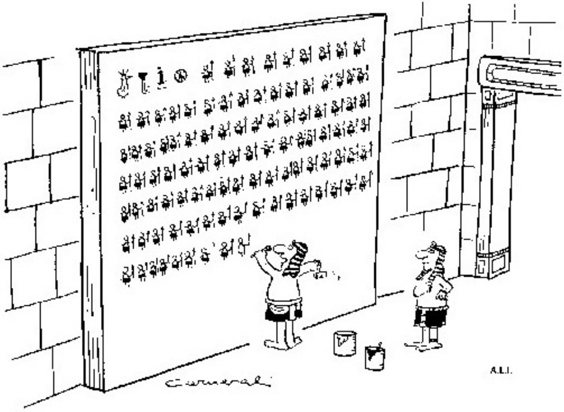
justification

implementation (*program*)

FM — simplified life-cycle



Notation matters!



Are you sure there isn't a simpler means of writing 'The Pharaoh had 10,000 soldiers?'

Credits: Cliff B. Jones 1980 [2]

Well-known FM notations / tools / resources

Just a sample, as there are many — follow the links (in alphabetic order):

Notations:

- Alloy
- B-Method
- Dafny
- mCRL2
- SPARK-Ada
- TLA+
- VDM
- Z

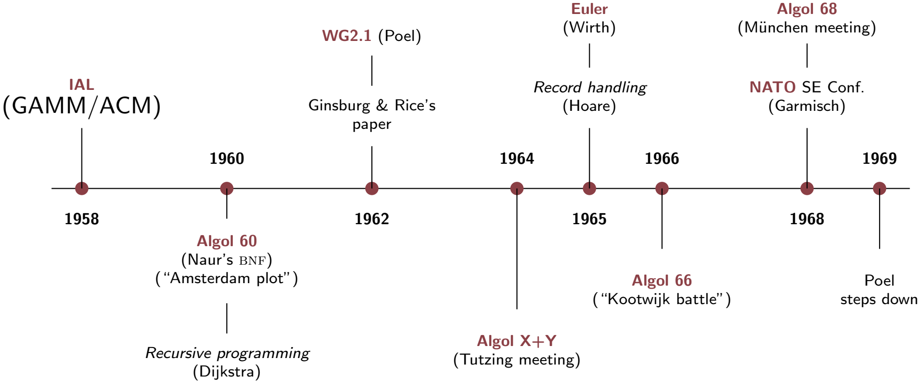
Tools:

- Alloy 6
- Coq
- Frama-C
- NuSMV
- Overture

Resources:

- Formal Methods Europe
- Formal Methods wiki (Oxford)

60+ years ago (1958-)



Hoare Logic — "turning point" (1968)

Floyd-Hoare logic for **program correctness** dates back to 1968:

Summary.

This paper illustrates the manner in which the axiomatic method may be applied to the rigorous definition of a programming language. It deals with the dynamic aspects of the behaviour of a program, which is an aspect considered to be most far removed from traditional mathematics. However, it appears that the axiomatic method not only shows how programming is closely related to traditional branches of logic and mathematics, but also formalises the techniques which may be used to prove the correctness of a program over its intended area of application.

(ADB/IFIP/1164;1456)

Inv/pre/post

Starting where (pure) **functions** stop:

```
Prelude> :{
Prelude| get :: [a] -> (a, [a])
Prelude| get x = (head x, tail x)
Prelude| :}
Prelude>
Prelude> get [1..10]
(1,[2,3,4,5,6,7,8,9,10])
Prelude> get [1]
(1,[])
Prelude> get []
(*** Exception: Prelude.head: empty list
```

Inv/pre/post

Error handling...

```
Prelude> get [] = Nothing ; get x = Just (head x, tail x)
Prelude> get []
Nothing
Prelude> get [1]
Just (1,[])
Prelude> :t get
get :: [a] -> Maybe (a, [a])
Prelude>
```

Inv/pre/post

Pre-conditions?

```haskell
get :: [a] -> (a, [a])
pre x = x /= []
get x = (head x, tail x)
```

Not everything is a **list**, a **tree** or a **stream**...

```haskell
get :: {a} -> (a, {a})
pre x = x /= {}
get x = let a = choice x
        in (a, x - {a})
```

Inv/pre/post

pre...? choice...?

- Non-determinism
- Parallelism
- Abstraction

Inv/pre/post

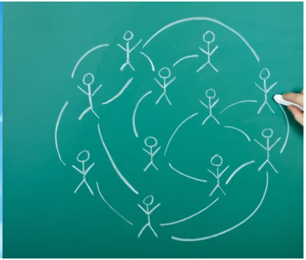
`pre`...? `choice`...?

- Non-determinism
- Parallelism
- Abstraction

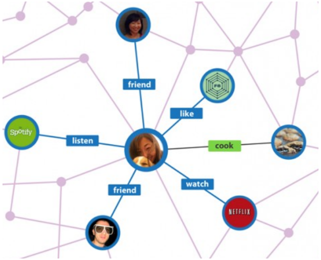
Functions not enough!

Solution?

Relations *(which extend functions)*



Is “everything” a relation?



How to "dematerialize" them?

Software is pre-science — **formal** but not fully **calculational**

Software is too **diverse** — many approaches, lack of unity

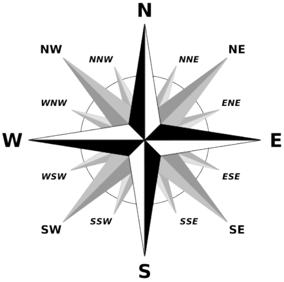
Software is too **wide** — from assembly to quantum programming

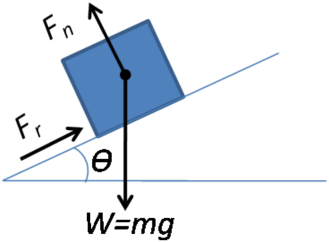
Can you think of a **unified** theory able to express and reason about software **in general**?

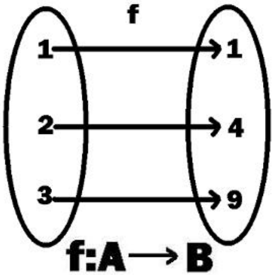
Put in another way:

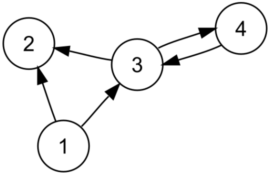
> *Is there a "lingua franca" for the software sciences?*

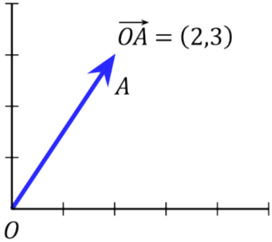
Check the pictures...



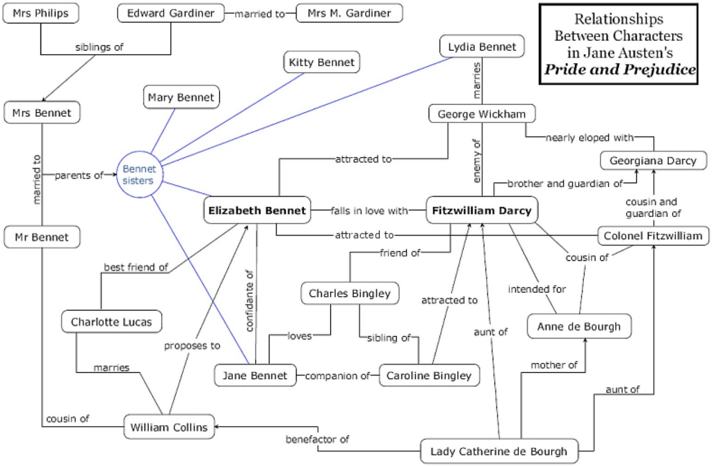






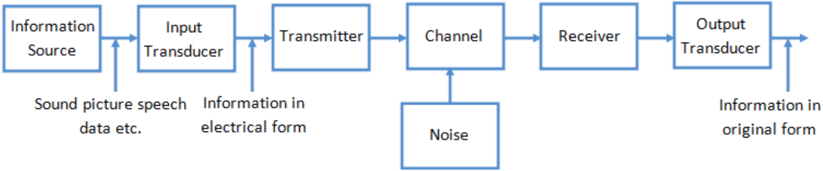


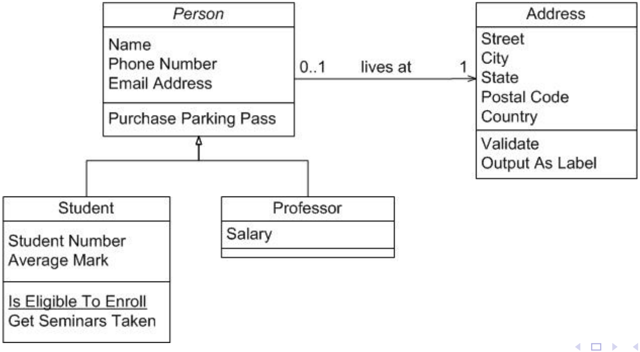
Check the pictures



(Wikipedia: **Pride and Prejudice**, by Jane Austin, 1813.)

Check the pictures





Check the pictures

Which **graphical** device have you found **common** to **all** pictures?

Arrows everywhere

Arrows! A (graphical) device **common** to describing (many) **different** fields of human activity.

For this ingredient to be able to support a **generic** theory of systems, mind the remarks:

- We need a **generic** notation able to cope with very distinct problem domains, e.g. **process** theory versus **database** theory, for instance.
- **Notation** is not enough — we need to **reason** and **calculate** about software.
- Semantics-rich **diagram** representations are welcome.
- System descriptions may have a **quantitative** side too.

Class 2

Relation algebra

In previous courses you may have used **predicate logic**, **finite automata**, **grammars** and so on to capture the meaning of real-life problems.

Question:

Is there a unified formalism for **formal modelling***?*

Relation algebra

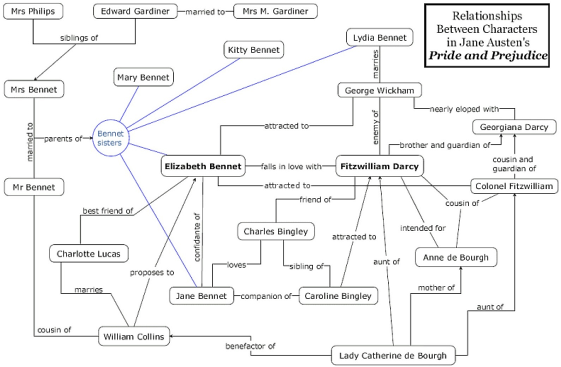
Historically, predicate logic was **not** the first one proposed:

- Augustus de Morgan (1806-71) — recall *de Morgan* laws — proposed a **Logic of Relations** as early as 1867.
- Predicate logic appeared later.

Perhaps de Morgan was right in the first place: in real life, “everything is a **relation**”...

Everything is a relation...

... as diagram



shows. (Wikipedia: **Pride and Prejudice**, by Jane Austin, 1813.)

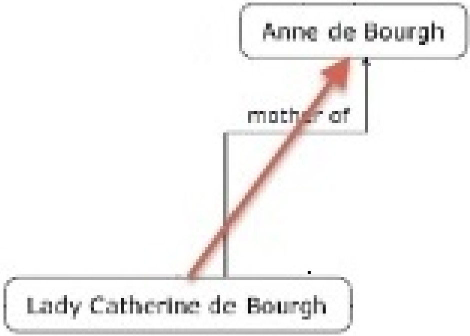
Arrow notation for relations

The picture is a collection of **relations** — vulg. a **semantic network** — elsewhere known as a (binary) **relational system**.

However, in spite of the use of **arrows** in the picture (aside) not many people would write

$$mother\_of : People \to People$$

as the **type** of **relation** *mother_of* .



Pairs

Consider assertions

$$
\begin{array}{rcl}
0 & \leqslant & \pi \\
\texttt{Catherine} & \mathit{isMotherOf} & \texttt{Anne} \\
3 & = (1+) & 2
\end{array}
$$

They are statements of fact concerning various kinds of object — real numbers, people, natural numbers, etc

They involve **two** such objects, that is, **pairs**

$$
\begin{array}{c}
(0, \pi) \\
(\texttt{Catherine}, \texttt{Anne}) \\
(3, 2)
\end{array}
$$

respectively.

Sets of pairs

So, one might have written instead:

$$\begin{array}{rcl} (0,\pi) & \in & (\leqslant) \\ (\texttt{Catherine},\texttt{Anne}) & \in & \mathit{isMotherOf} \\ (3,2) & \in & (1+) \end{array}$$

What are $(\leqslant)$, $\mathit{isMotherOf}$, $(1+)$?

- They can be regarded as **sets of pairs**
- Better: they should be regarded as **binary relations**.

Therefore,

- **orders** — eg. $(\leqslant)$ — are special cases of relations
- **functions** — eg. $\mathit{succ} = (1+)$ — are special cases of relations.

Binary Relations

Binary relations are typed:

Arrow notation. *Arrow* $A \xrightarrow{R} B$ *denotes a binary relation from* A *(source) to* B *(target).*

A, B are types.

Writing

$$B \xleftarrow{R} A$$

means the same as

$$A \xrightarrow{R} B \ .$$

Notation

Infix notation

The usual infix notation used in natural language — eg. $\mathit{Catherine}$ *isMotherOf* Anne *— and in maths — eg.* $0 \leqslant \pi$ *— extends to arbitrary* $B \xleftarrow{R} A$ *: we write*

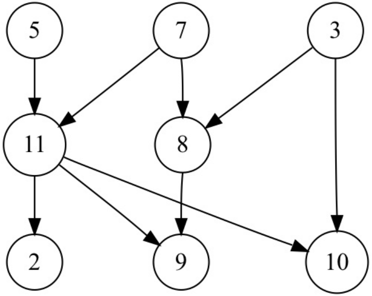
$$b \; R \; a$$

to denote that $(b, a) \in R$ *holds.*

Binary relations are matrices

Binary relations can be regarded as Boolean **matrices**, eg.

Relation R:



Matrix M:

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	1	0	0	0	1	0	0	0	0
9	0	0	0	0	0	0	0	1	0	0	1
10	0	0	1	0	0	0	0	0	0	0	1
11	0	0	0	0	1	0	1	0	0	0	0

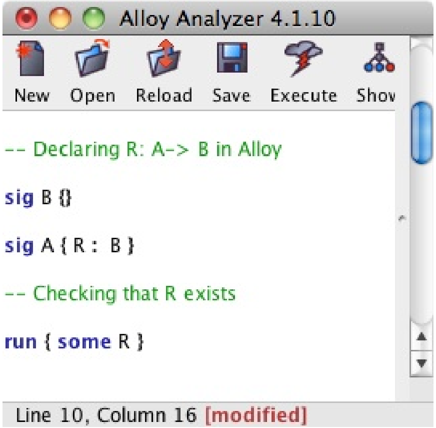
In this case $A = B = \{1..11\}$. Relations $A \xleftarrow{R} A$ over a single type A are also referred to as (directed) **graphs**.

Alloy: where "everything is a relation"

Declaring binary relation $A \xrightarrow{R} B$ in **Alloy** (aside).

Alloy is a tool designed at MIT (`http://alloy.mit.edu/alloy`)

We shall be using **Alloy** [1] in this course.



Functions are relations

Lowercase letters (or identifiers starting by one such letter) will denote special relations known as **functions**, eg. f, g, $succ$, etc.

We regard **function** $f : A \longrightarrow B$ as the binary relation which relates b to a iff $b = f\ a$. So,

$$b\ f\ a \ \text{ literally means } \ b = f\ a \tag{1}$$

Therefore, we generalize

$$\boxed{\begin{array}{c} B \xleftarrow{\ f\ } A \\ b = f\ a \end{array}} \quad \text{to} \quad \boxed{\begin{array}{c} B \xleftarrow{\ R\ } A \\ b\ R\ a \end{array}}$$

Exercise

Taken from PROPOSITIONES AD ACUENDOS IUUENES ("Problems to Sharpen the Young"), by abbot Alcuin of York († 804):

> XVIII. PROPOSITIO DE HOMINE ET CAPRA ET LVPO. *Homo quidam debebat ultra fluuium transferre lupum, capram, et fasciculum cauli. Et non potuit aliam nauem inuenire, nisi quae duos tantum ex ipsis ferre ualebat. Praeceptum itaque ei fuerat, ut omnia haec ultra illaesa omnino transferret. Dicat, qui potest, quomodo eis illaesis transire potuit?*

Exercise

XVIII. FOX, GOOSE AND BAG OF BEANS PUZZLE. *A farmer goes to market and purchases a fox, a goose, and a bag of beans. On his way home, the farmer comes to a river bank and hires a boat. But in crossing the river by boat, the farmer could carry only himself and a single one of his purchases - the fox, the goose or the bag of beans. (If left alone, the fox would eat the goose, and the goose would eat the beans.) Can the farmer carry himself and his purchases to the far bank of the river, leaving each purchase intact?*

Identify the main **types** and **relations** involved in the puzzle and draw them in a diagram.

Home work

- How would you address this problem?
- Try an write an Alloy for it (sig's only)

NB: You can seek help from ChatGPT — but please be critical...

```
abstract sig Item {}
one sig Fox, Goose, Beans extends Item {}

abstract sig Location {}
one sig InitialBank, FarBank extends Location {}

sig Boat {
    passengers: set Item
}

// Predicates to define the constraints
pred farmerCanCross[boat: Boat] {
    // Farmer must be on the boat
    Fox in boat.passengers or Goose in boat.passengers or Beans in b
}

pred foxAndGooseSafe[boat: Boat] {
    // Fox and Goose cannot be left alone together
    Fox in boat.passengers implies not (Goose in boat.passengers)
}
```

Propositio de homine et capra et lvpo

Data types:

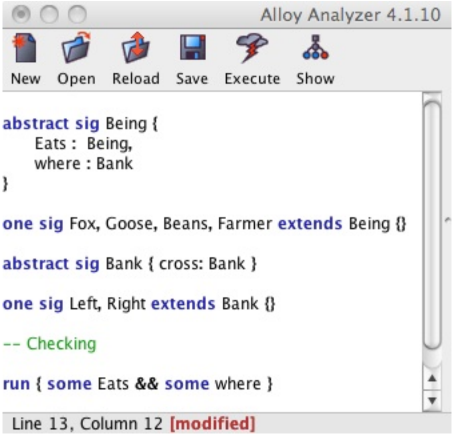
$$\begin{aligned} \mathit{Being} &= \{\mathit{Farmer}, \mathit{Fox}, \mathit{Goose}, \mathit{Beans}\} && (2) \\ \mathit{Bank} &= \{\mathit{Left}, \mathit{Right}\} && (3) \end{aligned}$$

Relations:

$$\begin{array}{ccccc} \mathit{Being} & \xrightarrow{\mathit{Eats}} & \mathit{Being} & & \\ & & \Big\downarrow{\scriptstyle \mathit{where}} & & \\ & & \mathit{Bank} & \xrightarrow{\mathit{cross}} & \mathit{Bank} \end{array} \qquad (4)$$

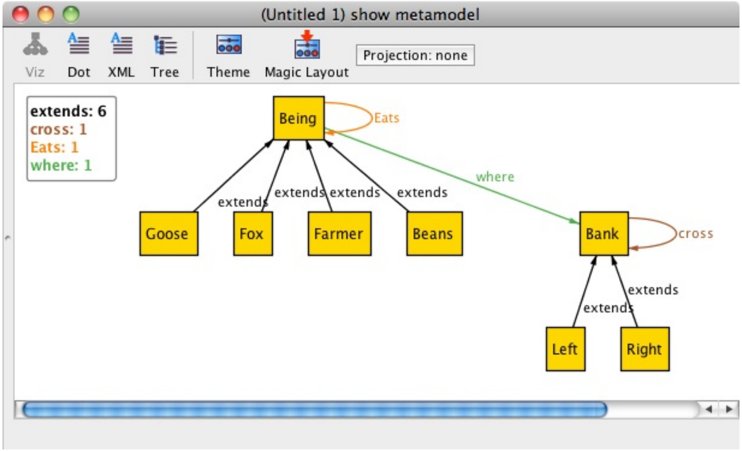
PROPOSITIO DE HOMINE ET CAPRA ET LVPO

Specification source written in Alloy:



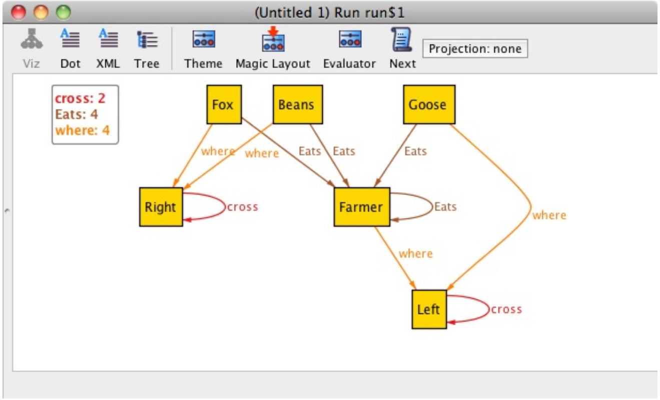
Propositio de homine et capra et lvpo

Diagram of specification (model) given by Alloy:



Propositio de homine et capra et lvpo

Diagram of instance of the model given by Alloy:



Silly instance, why? — specification too **loose**...

Propositio de homine et capra et lvpo

- How do we "fine tune" it?
- We need to be able to think in terms of **relations**

> *"Relational thinking"*

- Is there a "**language**" for relations ?

Yes! And a formal one 🙂

Propositio de homine et capra et lvpo

- How do we "fine tune" it?
- We need to be able to think in terms of **relations**

> *"Relational thinking"*

- Is there a "**language**" for relations ?

Yes! And a formal one

PROPOSITIO DE HOMINE ET CAPRA ET LVPO

- How do we “fine tune” it?
- We need to be able to think in terms of **relations**

“Relational thinking”

- Is there a “**language**” for relations ?

Yes! And a formal one

Propositio de homine et capra et lvpo

- How do we "fine tune" it?
- We need to be able to think in terms of **relations**

"Relational thinking"

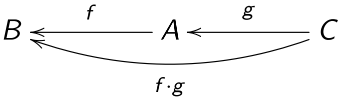
- Is there a "**language**" for relations ?

Yes! And a formal one

Composition

Recall **function composition** (aside).

We extend $f \cdot g$ to relational composition $R \cdot S$ in the obvious way:



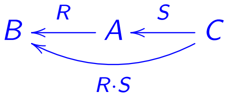
(5)

$$b = f(g\ c)$$

$$b(R \cdot S)c \quad \equiv \quad \langle \exists\ a\ ::\ b\ R\ a\ \wedge\ a\ S\ c \rangle$$

Composition

That is:



$$b(R \cdot S)c \;\equiv\; \langle \exists\, a \,::\, b\, R\, a \,\wedge\, a\, S\, c \rangle \tag{6}$$

Example: $Uncle = Brother \cdot Parent$, *that expands to*

$$u\ Uncle\ c \;\equiv\; \langle \exists\, p \,::\, u\ Brother\ p \wedge p\ Parent\ c \rangle$$

Note how this rule *removes* $\exists$ when applied from right to left.

Notation $R \cdot S$ is said to be **point-free** (no variables, or points).

Check generalization

Back to functions, (6) becomes[1]

$$
\begin{aligned}
b(f \cdot g)c &\equiv \langle \exists\, a :: \; b \; f \; a \wedge a \; g \; c \rangle \\
&\equiv \qquad \{ \; a \; g \; c \text{ means } a = g \; c \; (1) \; \} \\
& \quad \langle \exists\, a :: \; a = g \; c \wedge b \; f \; a \rangle \\
&\equiv \qquad \{ \; \exists\text{-trading } (41) \; ; \; b \; f \; a \text{ means } b = f \; a \; (1) \; \} \\
& \quad \langle \exists\, a : \; a = g \; c : \; b = f \; a \rangle \\
&\equiv \qquad \{ \; \exists\text{-one point rule } (45) \; \} \\
& \quad b = f(g \; c)
\end{aligned}
$$

So, we easily recover what we had before (5).

[1] Check the appendix on predicate calculus.

Relation inclusion

Relation inclusion generalizes function equality:

Equality *on functions*

$$f = g \quad \equiv \quad \langle \forall\ a\ ::\ f\ a = g\ a \rangle \tag{7}$$

generalizes to **inclusion** *on relations:*

$$R \subseteq S \quad \equiv \quad \langle \forall\ b, a\ :\ b\ R\ a :\ b\ S\ a \rangle \tag{8}$$

(read $R \subseteq S$ *as "*R *is at most* S*").*

Inclusion is **typed**:

For $R \subseteq S$ *to hold both* R *and* S *need to be of the same* **type**, *say* $B \xleftarrow{R,S} A$.

Relation inclusion

$R \subseteq S$ is a partial order, that is, it is

reflexive,

$$R \subseteq R \tag{9}$$

transitive

$$R \subseteq S \wedge S \subseteq Q \Rightarrow R \subseteq Q \tag{10}$$

and **antisymmetric**:

$$R \subseteq S \;\wedge\; S \subseteq R \quad\equiv\quad R = S \tag{11}$$

Therefore:

$$R = S \;\equiv\; \langle \forall\, b, a \;::\; b\, R\, a \;\equiv\; b\, S\, a \rangle \tag{12}$$

Special relations

Every type $B \longleftarrow A$ has its

- **bottom** relation $B \xleftarrow{\perp} A$, which is such that, for all b, a, $b \perp a \equiv \textsc{False}$
- **topmost** relation $B \xleftarrow{\top} A$, which is such that, for all b, a, $b \top a \equiv \textsc{True}$

Every type $A \longleftarrow A$ has the

- **identity** relation $A \xleftarrow{id} A$ which is nothing but function

$$id\ a = a \tag{13}$$

Clearly, for every R,

$$\perp \subseteq R \subseteq \top \tag{14}$$

Relational equality

Both (12) and (11) establish **relation equality**, resp. in PW/PF fashion.

Rule (11) is also called “ping-pong” or **cyclic inclusion**, often taking the format

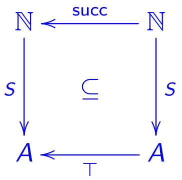
$$
\begin{array}{ll}
 & R \\
\subseteq & \{ \;....\; \} \\
 & S \\
\subseteq & \{ \;....\; \} \\
 & R \\
:: & \{ \;\text{“ping-pong” (11)}\; \} \\
 & R = S
\end{array}
$$

Diagrams

Assertions of the form $X \subseteq Y$ where X and Y are relation compositions can be represented graphically by **square**-shaped **diagrams**, see the following exercise.

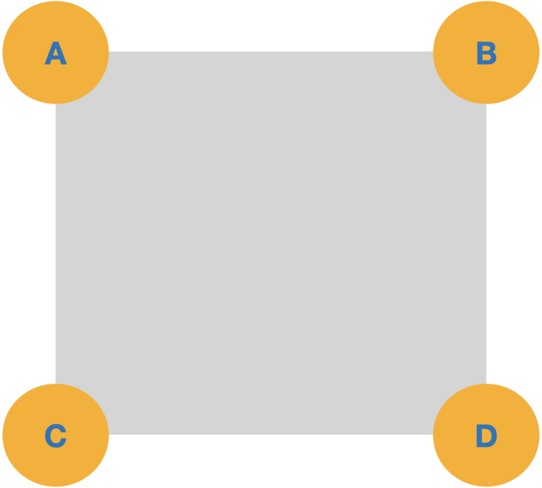
Exercise 1: Let $a\ S\ n$ mean: *"student a is assigned number n"*. Using (6) and (8), check that assertion

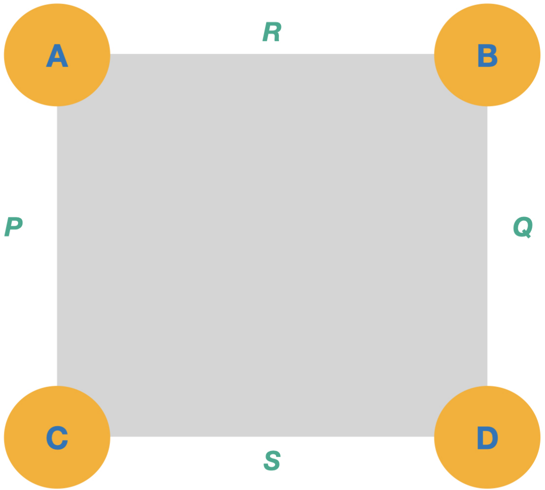
$S \cdot \mathsf{succ} \subseteq \top \cdot S$ depicted by diagram

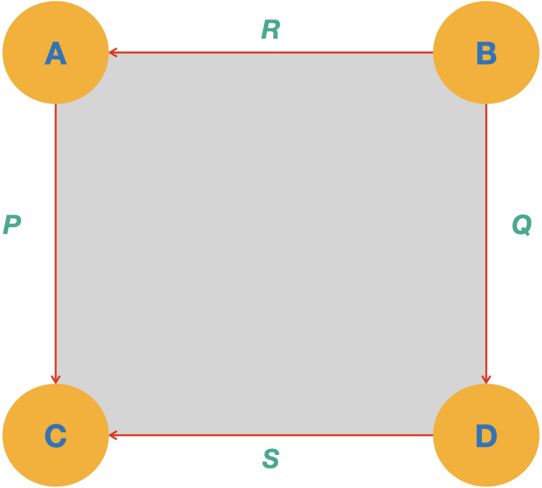


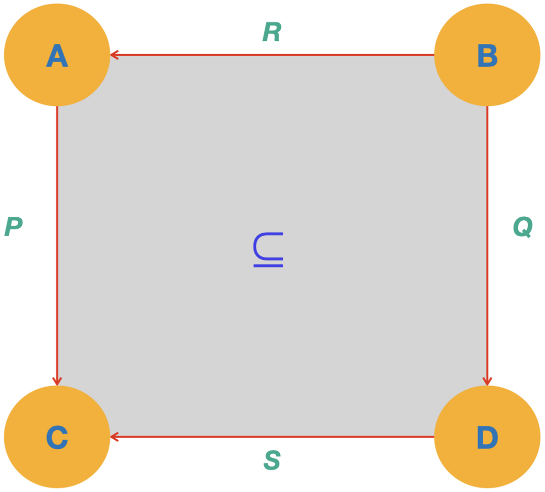
(where $\mathsf{succ}\ n = n + 1$) means that numbers are assigned to students sequentially. □

Squares

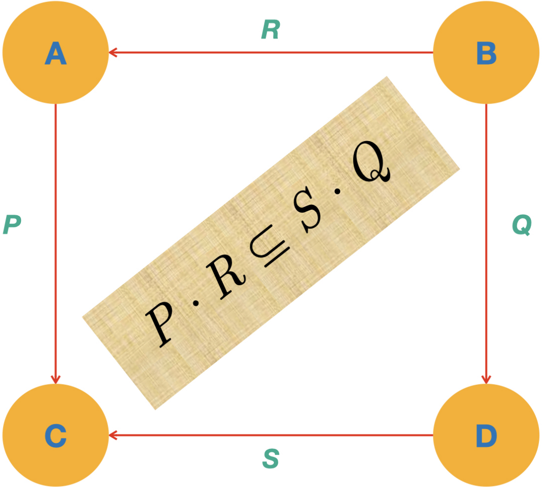

A
B
C
D


A
R
B
P
Q
C
S
D


A
R
B
P
Q
C
S
D


A
R
B
P
⊆
Q
C
S
D

"Magic" square



"Magic" square

Four binary relations:

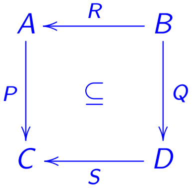
$$\begin{array}{ccc} A & \xleftarrow{\;R\;} & B \\ {\scriptstyle P}\downarrow & \subseteq & \downarrow{\scriptstyle Q} \\ C & \xleftarrow[\;S\;]{} & D \end{array} \qquad\qquad P \cdot R \subseteq S \cdot Q \qquad (15)$$

Terminology:

R — *pre-producer*
P — *pre-consumer*
Q — *post-producer*
S — *post-consumer*

“Magic” square

Pointfree:



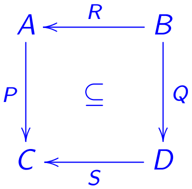
$$P \cdot R \subseteq S \cdot Q$$

Pointwise:

$$\langle \forall\ c, b\ ::\ \langle \exists\ a\ ::\ c\ P\ a \wedge a\ R\ b \rangle \Rightarrow \langle \exists\ d\ ::\ c\ S\ d \wedge d\ Q\ b \rangle \rangle \qquad (16)$$

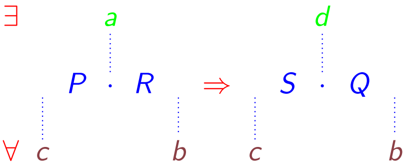
"Magic" square

Pointfree:



$$P \cdot R \subseteq S \cdot Q$$

Pointwise:



"Magic" square

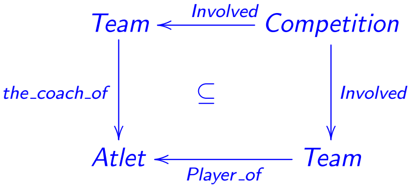
Perhaps not what you were expecting...

... but we can do **a lot** with

$$\begin{array}{ccc} A & \xleftarrow{\;R\;} & B \\ {\scriptstyle P}\big\downarrow & \subseteq & \big\downarrow{\scriptstyle Q} \\ C & \xleftarrow[\;S\;]{} & D \end{array}$$

Exercises

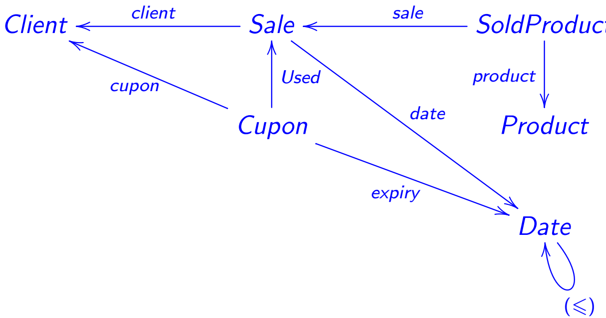
Exercise 2: Consider sports **competitions** involving **teams** which have **atlets** (players) and **coaches**. Follow rule (16) to spell out the logical meaning of the following *magic square*:



Then express this meaning in natural language, avoiding reading completely through the logic obtained via (16). □

Exercises

Exercise 3: ("D. Acácia grocery")

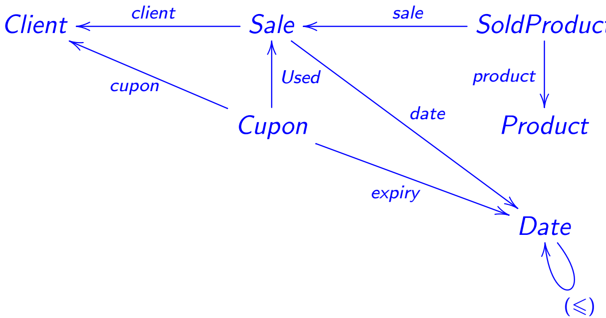


Find "magic square" for property:

Coupons cannot be used beyond their expiry date.

Exercises

Exercise 4: ("D. Acácia grocery")



Find "magic square" for property:

Coupons can only be used by clients who own them.

Exercises

Exercise 5: Use (6) and (8) and predicate calculus to show that

$$R \cdot id \;=\; R \;=\; id \cdot R \qquad (17)$$

$$R \cdot \bot \;=\; \bot \;=\; \bot \cdot R \qquad (18)$$

hold and that composition is associative:

$$R \cdot (S \cdot T) = (R \cdot S) \cdot T \qquad (19)$$

□

(**NB**: see the appendix for a compact set of rules of the predicate calculus.)

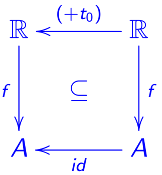
Exercises

Exercise 6: Use (7), (8) and predicate calculus to show that

$$f \subseteq g \quad \equiv \quad f = g$$

holds (moral: for functions, inclusion and equality coincide). □

Exercise 7: Let t_0 be a real number. Show that the "magic square"



tells that f is a periodic function (on $\mathbb{R}$) with period t_0. □

Converses

Every relation $B \xleftarrow{R} A$ has a **converse** $B \xrightarrow{R^\circ} A$ which is such that, for all a, b,

$$a(R^\circ)b \quad \equiv \quad b\ R\ a \tag{20}$$

Note that converse commutes with composition

$$(R \cdot S)^\circ = S^\circ \cdot R^\circ \tag{21}$$

and with itself:

$$(R^\circ)^\circ = R \tag{22}$$

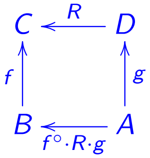
Converse captures the **passive voice**: *Catherine eats the apple* — $R = (eats)$ — is the same as *the apple is eaten by Catherine* — $R^\circ = (is\ eaten\ by)$.

Function converses

Function converses f°, g° etc. **always** exist (as **relations**) and enjoy the following (very useful!) property,

$$(f\ b)R(g\ a) \quad \equiv \quad b(f^\circ \cdot R \cdot g)a \tag{23}$$

cf. diagram:



Therefore (tell why):

$$b(f^\circ \cdot g)a \quad \equiv \quad f\ b = g\ a \tag{24}$$

Let us see an example of using these rules.

Class 3

PF-transform at work

Transforming a well-known PW-formula into PF notation:

f is **injective**

$\equiv$ { recall definition from discrete maths }

$\langle \forall\ y, x\ :\ (f\ y) = (f\ x) :\ y = x \rangle$

$\equiv$ { (24) for $f = g$ }

$\langle \forall\ y, x\ :\ y(f^\circ \cdot f)x :\ y = x \rangle$

$\equiv$ { (23) for $R = f = g = id$ }

$\langle \forall\ y, x\ :\ y(f^\circ \cdot f)x :\ y(id)x \rangle$

$\equiv$ { go pointfree (8) i.e. drop y, x }

$f^\circ \cdot f \subseteq id$

The other way round

Now check what $id \subseteq f \cdot f^\circ$ means:

$$
\begin{array}{ll}
& id \subseteq f \cdot f^\circ \\
\equiv & \quad \{ \text{ relational inclusion (8) } \} \\
& \langle \forall\ y, x\ :\ y(id)x :\ y(f \cdot f^\circ)x \rangle \\
\equiv & \quad \{ \text{ identity relation ; composition (6) } \} \\
& \langle \forall\ y, x\ :\ y = x :\ \langle \exists\ z\ ::\ y\ f\ z \wedge z\ f^\circ x \rangle \rangle \\
\equiv & \quad \{ \text{ } \forall\text{-one point (44) ; converse (20) } \} \\
& \langle \forall\ x\ ::\ \langle \exists\ z\ ::\ x\ f\ z \wedge x\ f\ z \rangle \rangle \\
\equiv & \quad \{ \text{ trivia ; function } f \text{ } \} \\
& \langle \forall\ x\ ::\ \langle \exists\ z\ ::\ x = f\ z \rangle \rangle \\
\equiv & \quad \{ \text{ recalling definition from maths } \} \\
& f \text{ is } \textbf{surjective}
\end{array}
$$

Why *id* (really) matters

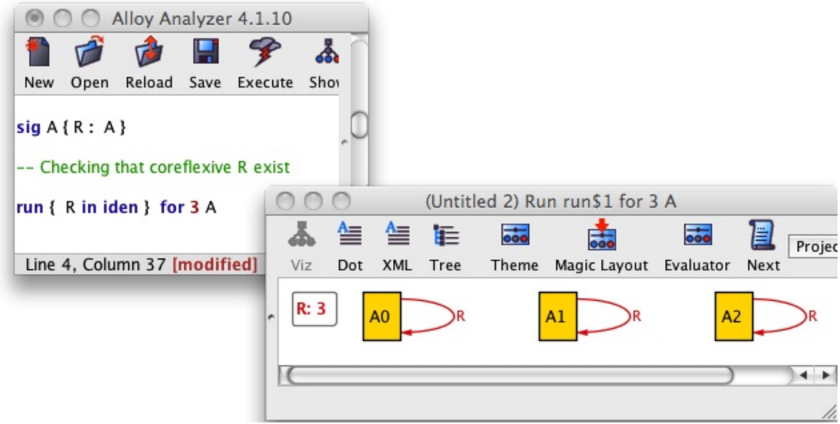
Terminology:

- Say R *is reflexive* iff $id \subseteq R$
  pointwise: $\langle \forall\ a\ ::\ a\ R\ a \rangle$ (check as homework);
- Say R *is coreflexive* (or *diagonal*) iff $R \subseteq id$
  pointwise: $\langle \forall\ b, a\ :\ b\ R\ a :\ b = a \rangle$ (check as homework).

Define, for $B \xleftarrow{R} A$:

Kernel of R	**Image** of R
$A \xleftarrow{\ker R} A$	$B \xleftarrow{\mathrm{img}\ R} B$
$\ker R \stackrel{\mathrm{def}}{=} R^\circ \cdot R$	$\mathrm{img}\ R \stackrel{\mathrm{def}}{=} R \cdot R^\circ$

Alloy: checking for coreflexive relations



Kernels of functions

Meaning of $\ker f$:

$$a'(\ker f)a$$

$$\equiv \quad \{ \text{ substitution } \}$$

$$a'(f^\circ \cdot f)a$$

$$\equiv \quad \{ \text{ rule (24) } \}$$

$$f\ a' = f\ a$$

In words: $a'(\ker f)a$ means a' *and* a *"have the same* f*-image"*.

Exercise 8: Let K be a nonempty data domain, $k \in K$ and $\underline{k}$ be the *"everywhere* k*"* function:

$$\begin{array}{rcl} \underline{k} & : & A \rightarrow K \\ \underline{k}\,a & = & k \end{array} \qquad (25)$$

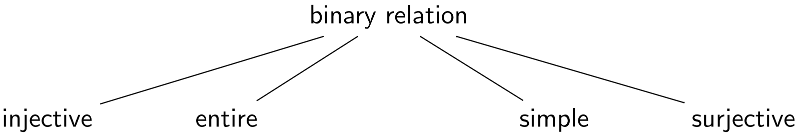
Compute which relations are defined by the following expressions:

$$\ker \underline{k}, \quad \underline{b} \cdot \underline{c}^\circ, \quad \text{img}\ \underline{k} \qquad (26)$$

□

Binary relation taxonomy

Topmost criteria:



Definitions:

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

(27)

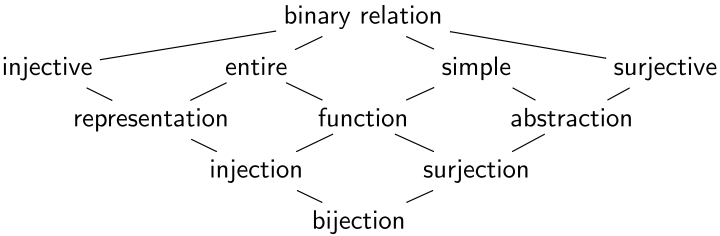
Facts:

$$\ker(R^\circ) = \operatorname{img} R \tag{28}$$

$$\operatorname{img}(R^\circ) = \ker R \tag{29}$$

Binary relation taxonomy

The whole picture:



(30)

Exercise 9: Resort to (28,29) and (27) to prove the following rules of thumb:

- converse of **injective** is **simple** (and vice-versa)
- converse of **entire** is **surjective** (and vice-versa)

□

The same in Alloy

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

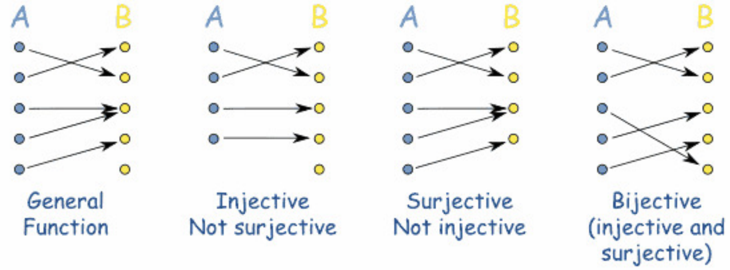
A lone -> one B	A some -> one B
injection	surjection

A one -> one B
bijection

(Courtesy of Alcino Cunha.)

Exercises

Exercise 10: Label the items (uniquely) in these drawings[2]



and compute, in each case, the **kernel** and the **image** of each relation. Why are all these relations **functions**? □

[2]Credits: `http://www.matematikaria.com/unit/injective-surjective-bijective.html`.

Exercises

Exercise 11: Prove the following fact

$$\textit{A function } f \textit{ is a bijection } \textbf{iff} \textit{ its converse } f^\circ \textit{ is a function} \tag{31}$$

by completing:

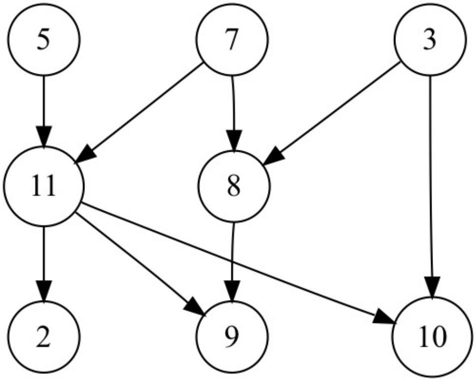
$$
\begin{array}{ll}
 & f \text{ and } f^\circ \text{ are functions} \\
\equiv & \qquad \{ \ \ldots \ \} \\
 & (id \subseteq \ker f \wedge \operatorname{img} f \subseteq id) \wedge (id \subseteq \ker (f^\circ) \wedge \operatorname{img} (f^\circ) \subseteq id) \\
\equiv & \qquad \{ \ \ldots \ \} \\
 & \vdots \\
\equiv & \qquad \{ \ \ldots \ \} \\
 & f \text{ is a bijection}
\end{array}
$$

□

Taxonomy using matrices

Recall that binary relations can be regarded as Boolean **matrices**, eg.

Relation *R*:



Matrix *M*:

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	1	0	0	0	1	0	0	0	0
9	0	0	0	0	0	0	0	1	0	0	1
10	0	0	1	0	0	0	0	0	0	0	1
11	0	0	0	0	1	0	1	0	0	0	0

Taxonomy using matrices

- **entire** — at least one 1 in every column (32)
- **surjective** — at least one 1 in every row (33)
- **simple** — at most one 1 in every column (34)
- **injective** — at most one 1 in every row (35)
- **bijective** — exactly one 1 in evey column and every row. (36)

Propositio de homine et capra et lvpo

Exercise 12: Let relation $Bank \xrightarrow{cross} Bank$ (4) be defined by:

$$Left \quad cross \quad Right$$
$$Right \quad cross \quad Left$$

It therefore is a bijection. Why? □

Exercise 13: Check which of the following properties,

simple, *entire*,
injective, *surjective*,
reflexive, *coreflexive*

Eats	*Fox*	*Goose*	*Beans*	*Farmer*
Fox	0	1	0	0
Goose	0	0	1	0
Beans	0	0	0	0
Farmer	0	0	0	0

hold for relation *Eats* (4) above ("food chain" $Fox > Goose > Beans$). □

PROPOSITIO DE HOMINE ET CAPRA ET LVPO

Exercise 14: Relation $where : Being \to Bank$ should obey the following constraints:

- *everyone is somewhere in a bank*
- *no one can be in both banks at the same time.*

Express such constraints in relational terms. Conclude that *where* should be a **function**. □

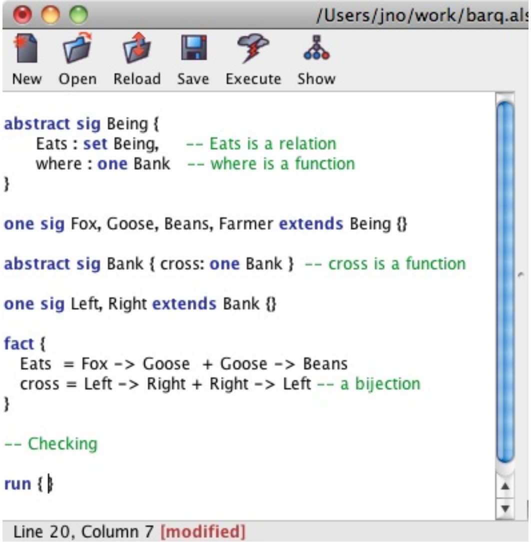
Exercise 15: There are only two **constant** functions (25) in the type $Being \longrightarrow Bank$ of *where*. Identify them and explain their role in the puzzle. □

Exercise 16: Two functions f and g are bijections iff $f^\circ = g$, recall (31). Convert $f^\circ = g$ to point-wise notation and check its meaning. □

Propositio de homine et capra et lvpo

Adding detail to the previous **Alloy** model (aside)

(More about Alloy syntax and semantics later.)



Functions in one slide

As seen before, a **function** f is a binary relation such that

Pointwise	Pointfree	
"Left" Uniqueness		
$b\ f\ a \wedge b'\ f\ a \Rightarrow b = b'$	$\mathrm{img}\ f \subseteq id$	(f is simple)
Leibniz principle		
$a = a' \Rightarrow f\ a = f\ a'$	$id \subseteq \ker f$	(f is entire)

NB: Following a widespread convention, functions will be denoted by lowercase characters (eg. f, g, ϕ) or identifiers starting with lowercase characters, and function application will be denoted by juxtaposition, eg. $f\ a$ instead of $f(a)$.

Functions, relationally

(The following properties of any function f are **extremely** useful.)

Shunting rules:

$$f \cdot R \subseteq S \quad \equiv \quad R \subseteq f^{\circ} \cdot S \qquad (37)$$

$$R \cdot f^{\circ} \subseteq S \quad \equiv \quad R \subseteq S \cdot f \qquad (38)$$

Equality rule:

$$f \subseteq g \;\equiv\; f = g \;\equiv\; f \supseteq g \qquad (39)$$

Rule (39) follows from (37,38) by "cyclic inclusion" (next slide).

Proof of functional equality rule (39)

$$
\begin{array}{ll}
 & f \subseteq g \\
\equiv & \quad \{ \text{ identity } \} \\
 & f \cdot id \subseteq g \\
\equiv & \quad \{ \text{ shunting on } f \text{ } \} \\
 & id \subseteq f^{\circ} \cdot g \\
\equiv & \quad \{ \text{ shunting on } g \text{ } \} \\
 & id \cdot g^{\circ} \subseteq f^{\circ} \\
\equiv & \quad \{ \text{ converses; identity } \} \\
 & g \subseteq f
\end{array}
$$

Then:

$$
\begin{array}{ll}
 & f = g \\
\equiv & \quad \{ \text{ cyclic inclusion (11) } \} \\
 & f \subseteq g \wedge g \subseteq f \\
\equiv & \quad \{ \text{ aside } \} \\
 & f \subseteq g \\
\equiv & \quad \{ \text{ aside } \} \\
 & g \subseteq f
\end{array}
$$

$\Box$

TBC...

Annex

Background — Eindhoven quantifier calculus

Trading:

$$\langle \forall\, k \;:\; \phi \wedge \varphi :\; \gamma \rangle \;=\; \langle \forall\, k \;:\; \phi :\; \varphi \Rightarrow \gamma \rangle \tag{40}$$

$$\langle \exists\, k \;:\; \phi \wedge \varphi :\; \gamma \rangle \;=\; \langle \exists\, k \;:\; \phi :\; \varphi \wedge \gamma \rangle \tag{41}$$

de Morgan:

$$\neg \langle \forall\, k \;:\; \phi :\; \gamma \rangle \;=\; \langle \exists\, k \;:\; \phi :\; \neg\, \gamma \rangle \tag{42}$$

$$\neg \langle \exists\, k \;:\; \phi :\; \gamma \rangle \;=\; \langle \forall\, k \;:\; \phi :\; \neg\, \gamma \rangle \tag{43}$$

One-point:

$$\langle \forall\, k \;:\; k = e :\; \gamma \rangle \;=\; \gamma[k := e] \tag{44}$$

$$\langle \exists\, k \;:\; k = e :\; \gamma \rangle \;=\; \gamma[k := e] \tag{45}$$

Background — Eindhoven quantifier calculus

Nesting:

$$\langle \forall\ a, b : \phi \wedge \varphi : \gamma \rangle = \langle \forall\ a : \phi : \langle \forall\ b : \varphi : \gamma \rangle \rangle \tag{46}$$

$$\langle \exists\ a, b : \phi \wedge \varphi : \gamma \rangle = \langle \exists\ a : \phi : \langle \exists\ b : \varphi : \gamma \rangle \rangle \tag{47}$$

Rearranging-∀:

$$\langle \forall\ k : \phi \vee \varphi : \gamma \rangle = \langle \forall\ k : \phi : \gamma \rangle \wedge \langle \forall\ k : \varphi : \gamma \rangle \tag{48}$$

$$\langle \forall\ k : \phi : \gamma \wedge \varphi \rangle = \langle \forall\ k : \phi : \gamma \rangle \wedge \langle \forall\ k : \phi : \varphi \rangle \tag{49}$$

Rearranging-∃:

$$\langle \exists\ k : \phi : \gamma \vee \varphi \rangle = \langle \exists\ k : \phi : \gamma \rangle \vee \langle \exists\ k : \phi : \varphi \rangle \tag{50}$$

$$\langle \exists\ k : \phi \vee \varphi : \gamma \rangle = \langle \exists\ k : \phi : \gamma \rangle \vee \langle \exists\ k : \varphi : \gamma \rangle \tag{51}$$

Splitting:

$$\langle \forall\ j : \phi : \langle \forall\ k : \varphi : \gamma \rangle \rangle = \langle \forall\ k : \langle \exists\ j : \phi : \varphi \rangle : \gamma \rangle \tag{52}$$

$$\langle \exists\ j : \phi : \langle \exists\ k : \varphi : \gamma \rangle \rangle = \langle \exists\ k : \langle \exists\ j : \phi : \varphi \rangle : \gamma \rangle \tag{53}$$

References

D. Jackson.
Software Abstractions: Logic, Language, and Analysis.
The MIT Press, Cambridge Mass., 2012.
Revised edition, ISBN 0-262-01715-2.

C.B. Jones.
Software Development — A Rigorous Approach.
Prentice-Hall, 1980.
IBSN 0138218846.