Lecture 1

Algebraic and Coalgebraic Methods in Software Development

October 30
Manuel A. Martins
martins@ua.pt
MAP-i

General information

- Introduction to the algebraic approach to formal specification of software systems (property – oriented)

Plan

Signatures, models

Equational logic

Signature morphisms

Refinements

Introduction to the theory of institutions

Behavioural specifications

- **Grading**

• Project (in cafeOBJ https://cafeobj.org/ or other specification language)

• Presentation of a paper with discussion

Main references

[AKK99] E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner, editors. *Algebraic foundations of systems specification.* IFIP State-of-the-Art Reports. Springer-Verlag, Berlin, 1999.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of algebraic specification 1. Equations and initial semantics.* EATCS. Monographs on Theoretical Computer Science, 6. Berlin: Springer-Verlag, 1985.

[GM96] J. Goguen and G. Malcolm. *Algebraic semantics of imperative programs.* MIT Press Series in the Foundations of Computing. Cambridge, 1996.

[LEW00] J. Loeckx, H.-D. Ehrich, and M. Wolf. Algebraic specification of abstract data types. In *Handbook of logic in computer science, Vol. 5*, Oxford Sci. Publ., pages 217–316. Oxford Univ. Press, New York, 2000.

[MT92] K. Meinke and J. V. Tucker. Universal algebra. In *Handbook of logic in computer science*, volume 1 of *Handb. Log. Comput. Sci.*, pages 189–411. Oxford Univ. Press, New York, 1992.

[ST??] D. Sannella and A. Tarlecki. *Foundations of Algebraic Specifications and Formal Program Development.* Springer, ~~to appear~~ . LINK

[DW02] K. Denecke and S. L. Wismath. *Universal algebra and applications in theoretical computer science.* Boca Raton, Florida: Chapman & Hall/CRC, 2002.

[GB92] J. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *J. Assoc. Comput. Mach.*, 39(1):95–146, 1992.

[GM85] J. Goguen and J. Meseguer. Completeness of many-sorted equational logic. *Houston J. Math.*, 11:307–334, 1985.

[Grä79] G. Grätzer. *Universal algebra. 2nd ed.* New York, Heidelberg, Berlin: Springer-Verlag, 1979.

[Hen97] R. Hennicker. Structural specifications with behavioural operators: semantics, proof methods and applications, 1997. Habilitationsschrift.

[JR97] B. Jacobs and J. Rutten. A tutorial on (co)algebra and (co)induction. *EATCS Bulletin*, 62:222–259, June 1997.

[MG96] G. Malcolm and J. Goguen. Proving correctness of refinement and implementation. Technical report, Oxford University Computing Laboratory, Technical Monograph PRG-114, 1996.

[ST88] D. Sannella and A. Tarlecki. Toward formal development of programs from algebraic specifications: Implementations revisited. *Acta Inf.*, 25(3):233–281, 1988.

[ST97] D. Sannella and A. Tarlecki. Essential concepts of algebraic specification and program development. *Formal Asp. Comput.*, 9(3):229–269, 1997.

[Wec92] W. Wechler. *Universal algebra for computer scientists.* EATCS Monographs on Theoretical Computer Science. 25. Berlin: Springer-Verlag, 1992.

Formal Methods, what are?

- Rigorous techniques and tools, mathematically developed, for **specifying** and **verifying** software and hardware systems:
- The techniques are supported by precise mathematical powerful analysis tools (including reasoning and formal verification)
- Constitute a rigorous and effective mechanism for modeling and systems analysis
- Help engineers construct trusty systems
  - **Languages**
  - **Logics**
  - **Models**
  - **Theorem provers**

Two interesting papers on FM

Edmund M. Clarke, Jeannette M. Wing. *Formal Methods: State of the Art and Future Directions,* **ACM Computing Surveys**, Volume 28 (4), pag. 626-643, 1996 (Cited by 1405).

…

Jonathan Bowen, and Michael G. Hinchey. *Ten commandments of formal methods... ten years later*. **Computer** 39(1): pag. 40-48, 2006.

Why?

- Some software systems are critical – in the sense that the failure or lack of availability has a serious human, environmental or economic effect.

- Examples:
  - – Control systems for complex equipment, such as an aircraft flight control system
  - – Infrastructure systems that manage national infrastructure (power, water, telecommunications, railways, etc.)
  - – Healthcare systems that manage patient information

(Ian Sommerville
http://www.software-engin.com/)

Importance

- "A major goal of software engineering is to enable developers to construct systems that operate reliably despite its complexity. One way of achieving this goal is by using *formal methods*. (…) Use of formal methods does not *a priori* guarantee correctness. However, they can greatly increase our understanding of a system by revealing inconsistencies, ambiguities and incompletenesses that might otherwise go undetected."

  (*Edmund M. Clarke, Jeannette M. Wing*)

FM in nowadays

Formal Methods Europe -http://www.fmeurope.org

FM2015 - 20th **International Symposium on Formal Methods** (FM'2015)
http://fm2015.ifi.uio.no

IEEE International Workshop on Formal Methods Integration - IEEE FMi 2015 http://eventegg.com/fmi-2015/

SEFM15- 13th International Conference on Software Engineering and Formal Methods – September 7-11, 2015, Grenoble, France. https://www.cs.york.ac.uk/sefm2015/

NFM 2016 - **8th NASA FORMAL METHODS SYMPOSIUM,** he Univeristy of Minnesota, Minneapolis, MN, June 7-9, 2016.
http://shemesh.larc.nasa.gov/NFM/

WADT2014 **- 22st International Workshop on Algebraic Development Techniques, Sinaia, Romania, Sep, 2014**

FM

Past of formal methods

- Heavy and imperfect notation
- Just academic examples
- Deficient tool support
- Hard to use tools
- Few applications (case studies)

Nowadays

- Establish more rigorous notations
- Model checking and theorem proving are present in the development of hardware in industry
- More industrial case studies have been studied

FM within the development process

Specification: description of the system and its properties

- Using a language with syntax and a semantics
- Integrate different specification languages to handle different aspects of a system
- Systems properties
    - Functional behavior
    - Timing behavior
    - Funcionality
    - evolution

Verification: Prove or disprove the correctness of a system with respect to the formal specification or property.Two well established approaches:

- Model cheking
    - Built a finite model and perform an exhaustive search
- Theorem proving
    - A formal proof in a logic proof system

Two perspectives

- **Model-Oriented**

Software systems are specified by a model of data/state model built by mathematical constructors using simpler structures as sets and strings and operations on the data and states.

- **Property-Oriented**

Software systems are specified as algebras and the properties of their algebraic operations are expressed as axioms in an appropriated logic.

Case studies: Intel

Intel uses formal verification quite extensively

Formal Methods at Intel — An Overview

John Harrison, Intel Corporation, Second NASA Formal Methods Symposium, NASA HQ, Washington DC, 14th April 2010

http://www.cl.cam.ac.uk/~jrh13/slides/nasa-14apr10/slides.pdf

Roope Kaivola: Intel CoreTM i7 Processor Execution Engine Validation in a Functional Language Based Formal Framework. PADL 2011: 1

Roope Kaivola, Rajnish Ghughal, Naren Narasimhan, Amber Telfer, Jesse Whittemore, Sudhindra Pandav, Anna Slobodová, Christopher Taylor, Vladimir Frolov, Erik Reeber, Armaghan Naik: Replacing Testing with Formal Verification in Intel CoreTM i7 Processor Execution Engine Validation. CAV 2009: 414-429

Roope Kaivola: Formal Verification of Pentium® 4 Components with Symbolic Simulation and Inductive Invariants. CAV 2005: 170-184

Case studies: Intel

Verification of Intel Pentium 4 floating-point unit with a mixture of STE and theorem proving

Verification of bus protocols using pure temporal logic model checking

Verification of microcode and software for many Intel Itanium floating-point operations, using pure theorem proving

Formal Verification found many high-quality bugs in P4 and verified “20%” of design

Formal Verification is now standard practice in the floating-point domain

Case studies: NASA SATS

Small Aircraft Transportation System (SATS)

Goal: Develop a software system that will sequence aircraft into the SATS airspace.

These software systems are critical and so it is mandatory strong guarantees of the safety be developed for them.

NASA Langley researchers (http://shemesh.larc.nasa.gov/fm/index.html) are currently investigating rigorous verification of these software system using formal methods :

- Modeling and Verification of Air Traffic
- Conflict Detection and Alerting …

César Muñoz, Víctor Carreño, and Gilles Dowek, Formal Analysis of the Operational Concept for the Small Aircraft Transportation System. M. Butler et al. (Eds.): REFT 2005, LNCS 4157, pp. 306–325, 2006.

Demands for a software system

- Efficient
- Robust
- Secure
- User-friendly
- Well documented

  …
- **CORRECT**

The major goal of software engineers is to develop reliable (trusty) systems

Software systems as algebras

- Emphasis on

  input/output behaviour

- "Do not care" about the

  concrete details of code and algorithms

- Model software systems using mathematical structures:

  ALGEBRAS

to model the relationship between inputs and outputs.

```
fun f1(n) = if n<=1 then 1
            else f1(n-1)+f1(n-2)

fun f2(n) = if n<=1 then 1
            else let fun g(n) = if n=1 then (1,1)
                                else let val (u,v) = g(n-1) in (u+v,u) end
                 in  let val (u,v) = g(n-1) in u+v end
                 end

fun f3(n) = if n<=1 then 1
            else let val muv = ref (1,1,1) in
                     (while let val (m,u,v) = !muv in m < n end do
                        muv := let val (m,u,v) = !muv in (m+1,u+v,u) end;
                        let val (m,u,v) = !muv in u end) end
```

```
public static nat f4(nat n) {
      nat u = 1, v = 1;
      for (nat m = 1; m < n; m++) {
          u = u + v;
          v = u - v;
      }
      return u;
}
```

- In all these examples we have the definition of the Fibonacci function.
- They differ only by the language and / or algorithm used.
- f1, f2 and f3 are in Standard ML while f4 is in Java.

- Moreover,

  In f1 and f2 is used recursion and purely functional while f3 and f4 are iterative and use "Assignment".

Exercise: Check the differences in the algorithm used.

However, the most important common feature is that all of them encode the Fibonacci function

fib: Nat $\rightarrow$ Nat defined by:

fib (0) = 1
fib (1) = 1
fib (n + 2) = fib (n + 1) + fib (n)

Formal methods

- Formal methods can be applied at various stages through the program development process
    - ✓ Specification
    - ✓ Verification
- „Specification: Give a description of the „system to be developed, and its properties
- Verification: Prove or disprove the correctness of a system with respect to the formal specification or property

Property Oriented

The abstract data types or object classes are specified by setting properties of the behavior of the associated operations.

We use methods and arguments of mathematical logic to model, analyze, design, build and improve existing software, i.e. **formal methods**. They are used to ensure correcting the current solution, and constitute an area on the border between mathematics and Software Engineering.

Algebraic specification is a branch of Formal Methods, constituting a solid foundational basis for the systematic development of programs, checking its accuracy with compliance requirements and allowing a building step by step, based on correct refinement procedures.

Example of a specification in Cafeobj

Spec: Stack;
Extend Nat **by**
Sorts: Stack;
Operations:
    newstack: $\to$ Stack
    push: Nat $\times$ Stack $\to$ Stack
    pop: Stack $\to$ Stack
    top: Stack $\to$ Nat
Variables:
    s: Stack; n: Nat
Axioms:
    pop(newstack) = newstack;
    top(newstack) = zero;
    pop(push(n, s)) = s;
    top(push(n, s)) = n;

Cafeobj Home page: https://cafeobj.org/
The Maude system:
http://maude.cs.illinois.edu/w/index.php?title=The_Maude_System

References for today

[MT92] K. Meinke and J. V. Tucker. Universal algebra. In *Handbook of logic in computer science*, volume 1 of *Handb. Log. Comput. Sci.*, pages 189–411. Oxford Univ. Press, New York, 1992.

[ST??] D. Sannella and A. Tarlecki. *Foundations of Algebraic Specifications and Formal Program Development*. Springer, to appear .

[LEW00] J. Loeckx, H.-D. Ehrich, and M. Wolf. Algebraic specification of abstract data types. In *Handbook of logic in computer science, Vol. 5*, Oxford Sci. Publ., pages 217–316. Oxford Univ. Press, New York, 2000.

Mathematical background

Relations:

- *n-ary relation* on A
- *Binary relation* if n=2
- *Inverse* of a binary relation
(a R^{-1} b iff b R a)
- *Relational product* $R \circ S$
(a $R \circ S$ b iff there is c st a R c and c S b)

Mathematical background

- **Functions**

a *function* f from a set A to a set B, written $f : A \to B$, is a subset of $A \times B$ such that for each $a \in A$ there is exactly one $b \in B$ with $\langle a, b \rangle \in f$; in this case we write $f(a) = b$ or $f : a \mapsto b$;

the set of all functions from A to B is denoted by B^A;

the function $f \in B^A$ is *injective* (or *one-to-one*) if $f(a_1) = f(a_2) \Rightarrow a_1 = a_2$;

the function $f \in B^A$ is *surjective* (or *onto*) if for every $b \in B$ there is an $a \in A$ with $f(a) = b$;

the function $f \in B^A$ is *bijective* if it is both injective and surjective;

for $f \in B^A$ and $X \subseteq A$, $f(X) = \{b \in B : f(a) = b \text{ for some } a \in X\}$;

for $f \in B^A$ and $Y \subseteq B$, $f^{-1}(Y) = \{a \in A : f(a) \in Y\}$;

Mathematical background

Definition. A binary relation $\leq$ on A is a *partial order* if

- $a \leq a$;
- $a \leq b$ and $b \leq a$ implies $a = b$
- $a \leq b$ and $b \leq c$ implies $a \leq c$

It is a *total order* if (additionally)
$a \leq b$ or $b \leq a$

Partially ordered set --- poset

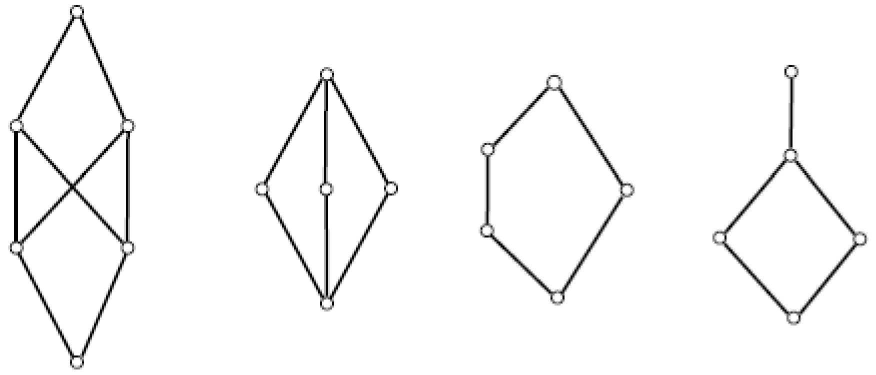
EXAMPLES

Mathematical background

Some more notions:

Upper bound and supremum
lower bound and infimum
Cover relation
Interval

Hasse diagrams



Mathematical background

Definition. A poset $(P, \leq)$ is *complete* if for any subset A of P $infA$ and $supA$ exist.

Theorem. If any subset A of a poset P has supremum then any subset has infimum.

EXAMPLES

Mathematical background

Definition. A binary relation R on A is an *equivalence relation on* A if

- aRa;
- aRb implies bRa
- aRb and bRc implies aRc

$Eq(A)$ is the set of all equivalence relations on A.

Theorem.
$(Eq(A), \subset)$ is a complete poset

Theorem. $\theta_1, \theta_2 \in Eq(A)$,

$$\theta_1 \vee \theta_2 = \theta_1 \cup (\theta_1 \circ \theta_2) \cup (\theta_1 \circ \theta_2 \circ \theta_1) \cup (\theta_1 \circ \theta_2 \circ \theta_1 \circ \theta_2) \cup \cdots ,$$

Moreover:
Theorem. $\theta_i \in Eq(A), i \in I$,

$$\bigvee_{i \in I} \theta_i = \bigcup \{\theta_{i_0} \circ \theta_{i_1} \circ \cdots \circ \theta_{i_k} : i_0, \ldots, i_k \in I,\ k < \infty\}.$$

Mathematical background

Definition.
Equivalence class: $a/\theta = \{b \in A : a\theta b\}$

Quotient: $A/\theta = \{a/\theta : a \in A\}$

Fact

$A = \bigcup_{a \in A} a/\theta.$
$a/\theta \neq b/\theta$ *implies* $a/\theta \cap b/\theta = \varnothing.$

Definition. A partition of a set A is a nonempty set π of pairwise disjoint subsets of A such that $A = \bigcup \pi$.

Isomorphism $\pi \mapsto \theta(\pi)$???

Moreover, by defining

$\pi_1 \leq \pi_2$ iff each block of π_1 is contained in some block of π_2.

we have a isomorphism between the corresponding posets

Mathematical background

Closure operator

$X \subseteq C(X)$	(extensive)
$C^2(X) = C(X)$	(idempotent)
$X \subseteq Y$ implies $C(X) \subseteq C(Y)$	(isotone).

closed subset if $C(X) = X$.

algebraic closure operator $C(X) = \bigcup\{C(Y) : Y \subseteq X \text{ and } Y \text{ is finite}\}$.

EXAMPLES

FACT.
Every complete poset is isomorphic to the poset of closed subsets of some A

Introduction

The term specification is used for the process of writing representations from entity that defines the class of algebras, i.e., the description of the class of admissible software systems.

Thus, a specification must define a signature and a class of algebras on that signature.

Typically, but not always, a class of algebras is defined by the properties that its members meet.

These properties are expressed by formulas (called axioms) in
some logical system, for example: equational logic, modal logic or first order logic.

Informally:

interface module ----------- signature

module --------------------- algebra

module specification --- class of algebras

Heterogeneous data

We use algebra to model programs which manipulates several sorts of data.

So, the underlying set of values in the algebra should be partitioned so that there is one set of values for each sort of data.

It is worth to manipulate such a family of sets as a unit, in such a way that operations on this unit respect the “typing” of data values.

Many-sorted sets

Let S be a set

$\boxed{\text{An } S\text{-}\textit{sorted set} \text{ is an } S\text{-indexed family of sets } X = \langle X_s \rangle_{s \in S}}$

also denoted by $\langle X_s : s \in S \rangle$

Let $X = \langle X_s \rangle_{s \in S}$ and $Y = \langle Y_s \rangle_{s \in S}$ be S-sorted sets.

$X \cup Y = \langle X_s \cup Y_s \rangle_{s \in S}$

$X \cap Y = \langle X_s \cap Y_s \rangle_{s \in S}$

$X \times Y = \langle X_s \times Y_s \rangle_{s \in S}$

$X \uplus Y = \langle X_s \uplus Y_s \rangle_{s \in S}$ (where $X_s \uplus Y_s = (\{1\} \times X_s) \cup (\{2\} \times Y_s)$)

$X \subseteq Y$ iff (if and only if) $X_s \subseteq Y_s$ for all $s \in S$

$X = Y$ iff $X \subseteq Y$ and $Y \subseteq X$ (equivalently, iff X and Y are equal as functions).

The usual set theoretical notions extend to many sorted in a natural way, for example:

An S-*sorted function* $f\colon X \to Y$ is an S-indexed family of functions $f = \langle f_s\colon X_s \to Y_s \rangle_{s\in S}$

Properties of functions, like *injectivity, surjective*, etc …
are defined **componentwisely**

Analogously, we define

S-sorted binary relation
S-sorted kernel
S-sorted equivalence
…

Example

Let $S = \{s_1, s_2\}$, and let X and Y be two S-sorted sets defined as follows:

$$X = \langle X_s \rangle_{s \in S} \text{ where } X_{s_1} = \{\square, \triangle\} \text{ and } X_{s_2} = \{\clubsuit, \heartsuit, \spadesuit\},$$
$$Y = \langle Y_s \rangle_{s \in S} \text{ where } Y_{s_1} = \{1, 2, 3\} \text{ and } Y_{s_2} = \{1, 2, 3\}.$$

Let $f \colon X \to Y$ be the S-sorted function such that

$$f_{s_1} = \{\square \mapsto 1, \triangle \mapsto 3\} \quad \text{and} \quad f_{s_2} = \{\clubsuit \mapsto 1, \heartsuit \mapsto 2, \spadesuit \mapsto 2\}.$$

(i.e., $f_{s_1}(\square) = 1$ and $f_{s_1}(\triangle) = 3$; analogously for f_{s_2}). Then the kernel of f is the S-sorted equivalence relation $K(f) = \langle K(f_s) \rangle_{s \in S}$ where

$$K(f_{s_1}) = \{\langle \square, \square \rangle, \langle \triangle, \triangle \rangle\} \quad \text{and} \quad K(f_{s_2}) = \{\langle \clubsuit, \clubsuit \rangle, \langle \heartsuit, \heartsuit \rangle, \langle \heartsuit, \spadesuit \rangle, \langle \spadesuit, \heartsuit \rangle, \langle \spadesuit, \spadesuit \rangle\}.$$

The quotient of X modulo $K(f)$ is the S-sorted set $X/K(f) = \langle X_s/K(f_s) \rangle_{s \in S}$ where

$$X_{s_1}/K(f_{s_1}) = \{\{\square\}, \{\triangle\}\} \quad \text{and} \quad X_{s_2}/K(f_{s_2}) = \{\{\clubsuit\}, \{\heartsuit, \spadesuit\}\}.$$

Donald Sannella, Andrzej Tarlecki, Foundations of Algebraic Specification and Formal Software Development . Springer, to appear-

Universal Algebra, what is?

In mathematics, universal algebra is the theory that unifies the treatment of several classes of algebraic structures like boolean algebras, groups, rings, etc…

In computer science, universal algebra is considered to be a general theory which is directly applicable to many problems that may be modelled by sets and functions, i.e., an ALGEBRA

Signature

Definition. A signature is a pair $\Sigma = (S, \Omega)$ such that

- S is a set (of sorts)
- Ω is a $S^* \times S$ sorted set of operation symbols.

The elements of $\Omega_{\lambda,s}$ are called constants of sort s

The elements of $\Omega_{\omega,s}$ are called operation symbols of type $\omega \to s$.

Subsignature as expected !

Example: Vector space

Gen

$\mathtt{vect}$

$\mathtt{esc}$

Op

$1 : \rightarrow \mathtt{esc};$

$\odot : \rightarrow \mathtt{esc};$

$\oplus : \mathtt{esc}, \mathtt{esc} \rightarrow \mathtt{esc};$

${}^{-1} : \mathtt{esc} \rightarrow \mathtt{esc};$

$\ominus : \mathtt{esc} \rightarrow \mathtt{esc};$

$* : \mathtt{esc}, \mathtt{esc} \rightarrow \mathtt{esc};$

$\times : \mathtt{esc}, \mathtt{vect} \rightarrow \mathtt{vect};$

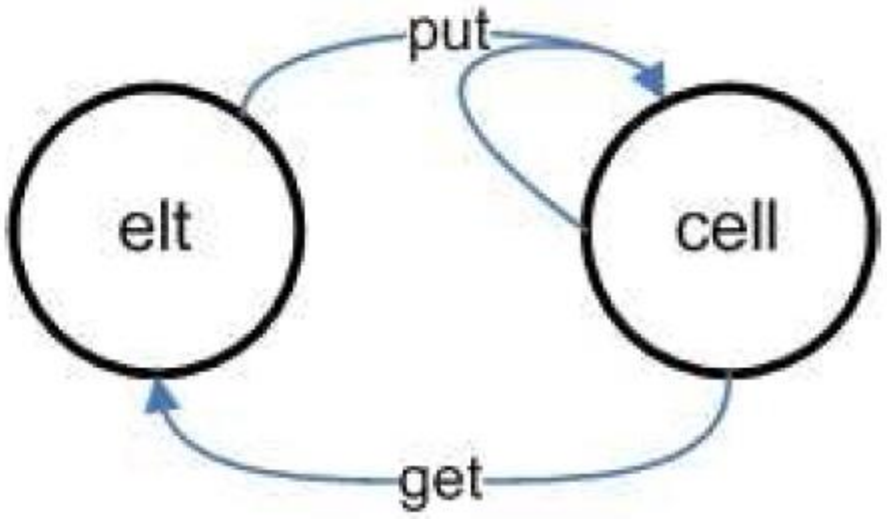
$0 : \rightarrow \mathtt{vect};$

$- : \mathtt{vect} \rightarrow \mathtt{vect};$

$+ : \mathtt{vect}, \mathtt{vect} \rightarrow \mathtt{vect};$

Example: cell of memory

```
Gen
      elt
      cell
Op
      put : elt, cell → cell;
      get : cell → elt
```



Example: automata

Gen

  Z

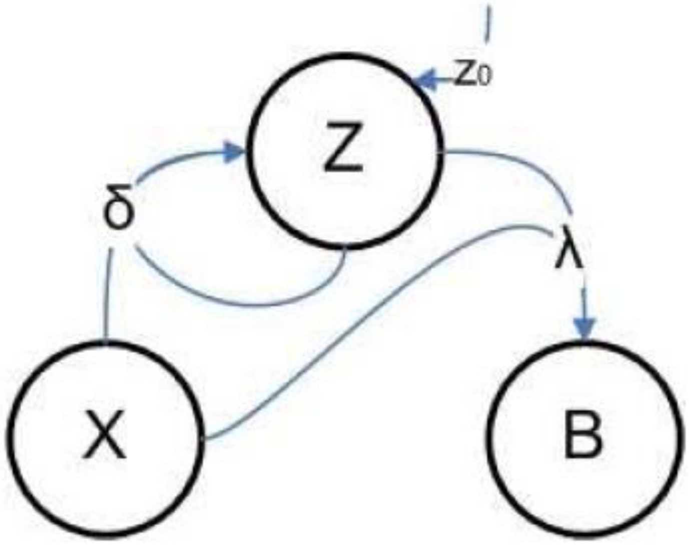
  X

Op

  $\delta : Z, X \rightarrow Z;$

  $z_0 : \quad \rightarrow Z$

Automata with output



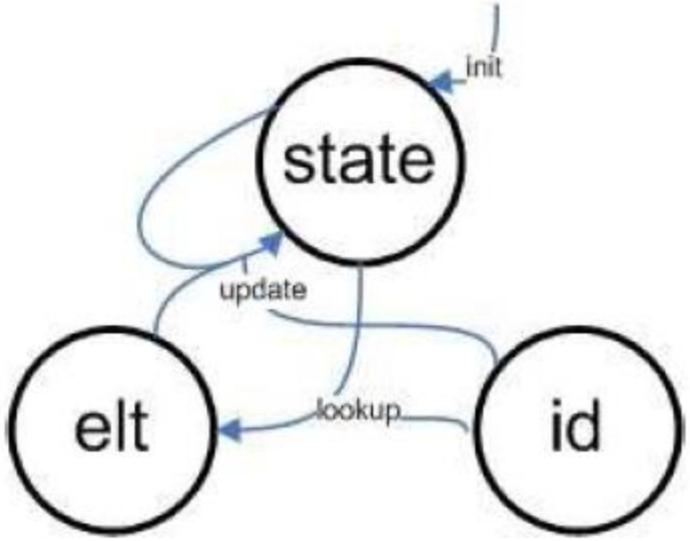
Example: Mem. with identifiers

Gen

- state
- id
- elt

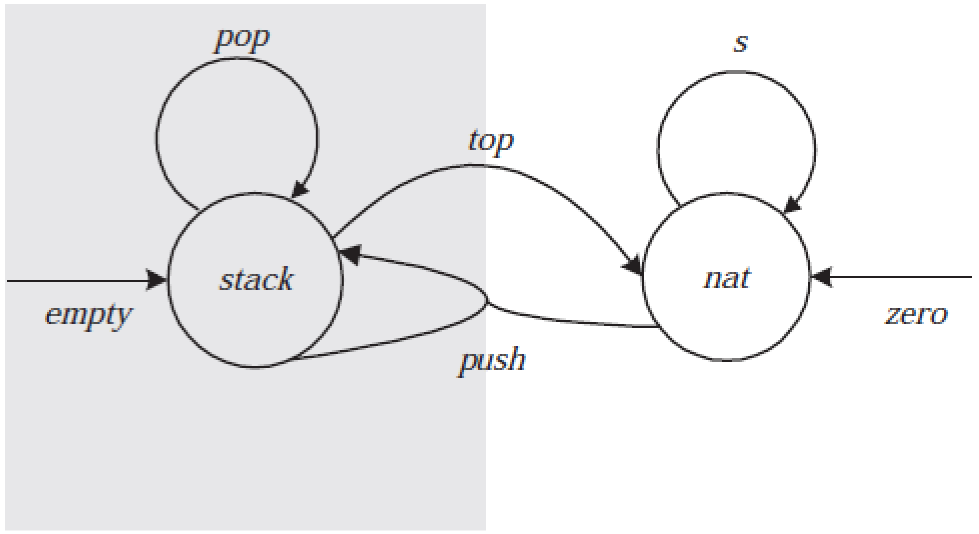
Op

- $\mathtt{init} :\to \mathtt{state};$
- $\mathtt{update} : \mathtt{id}, \mathtt{elt}, \mathtt{state} \to \mathtt{state};$
- $\mathtt{lookup} : \mathtt{id}, \mathtt{state} \to \mathtt{elt}$



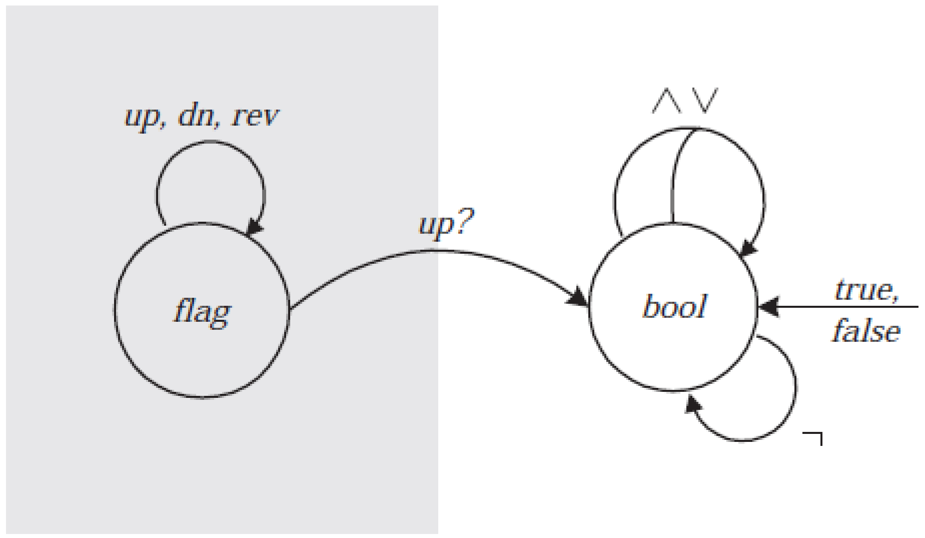
Example: Stacks

Gen	
	stack
	nat
Op	
zero :	$\to$ nat
empty :	$\to$ stack
s :	nat $\to$ nat
top :	stack $\to$ nat
pop :	stack $\to$ stack
push :	stack, nat $\to$ stack



Example: Flags

Gen	
	bool
	flag
Op	
up :	flag → flag;
rev :	flag → flag;
dn :	flag → flag;
up? :	flag → bool;
true :	→ bool;
false :	→ bool;
¬ :	bool → bool;
∧ :	bool, bool → bool;
∨ :	bool, bool → bool.



Algebras and subalgebras

Definition. Let $\Sigma = (S, \Omega)$ be a signature. A Σ-*algebra* consists on

- an S-set $A = \langle A_s \rangle_{s \in S}$, with $s \in S$ $A_s \neq \emptyset$, and
- for each $f : s_1, \ldots, s_n \to s \in \Sigma$, a function/operation

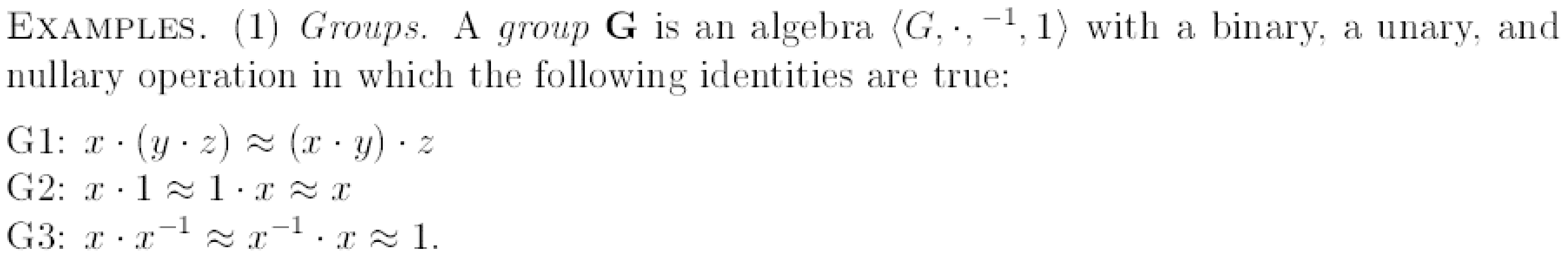
$$f^{\mathcal{A}} : A_{s_1} \times \cdots \times A_{s_n} \to A_s.$$

Definition. Let $\mathcal{A}$ and $\mathcal{B}$ be Σ-algebras.
$\mathcal{B}$ is a *subalgebra of* $\mathcal{A}$, $B \leq A$, if $B \subseteq A$ and for every $f : s_1, \ldots, s_n \to s \in \Sigma$ and all $b_1 \in B_{s_1}, \ldots, b_n \in B_{s_n}$,

$$f^{\mathcal{B}}(b_1, \ldots, b_n) = f^{\mathcal{A}}(b_1, \ldots, b_n).$$

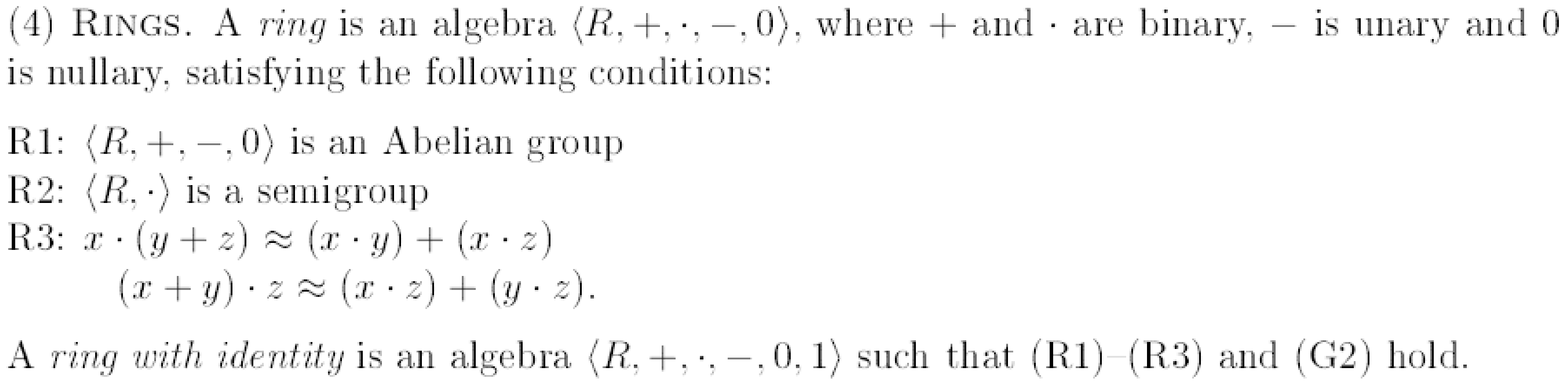
Definition. A *subuniverse of* $\mathcal{A}$ is a subset B of A closed for the operations on $\mathcal{A}$, i.e., if $f^{\mathcal{A}} : A_{s_1} \times \cdots \times A_{s_n} \to A_s$ is an operation of $\mathcal{A}$ and $b_1 \in B_{s_1}, \ldots, b_n \in B_{s_n}$ then $f^{\mathcal{A}}(b_1, \ldots, b_n) \in B_s$.

Examples in Mathematics



EXAMPLES. (1) *Groups.* A *group* **G** is an algebra $\langle G, \cdot, ^{-1}, 1\rangle$ with a binary, a unary, and nullary operation in which the following identities are true:

G1: $x \cdot (y \cdot z) \approx (x \cdot y) \cdot z$
G2: $x \cdot 1 \approx 1 \cdot x \approx x$
G3: $x \cdot x^{-1} \approx x^{-1} \cdot x \approx 1.$



(4) RINGS. A *ring* is an algebra $\langle R, +, \cdot, -, 0\rangle$, where $+$ and $\cdot$ are binary, $-$ is unary and 0 is nullary, satisfying the following conditions:

R1: $\langle R, +, -, 0\rangle$ is an Abelian group
R2: $\langle R, \cdot\rangle$ is a semigroup
R3: $x \cdot (y + z) \approx (x \cdot y) + (x \cdot z)$
$(x + y) \cdot z \approx (x \cdot z) + (y \cdot z).$

A *ring with identity* is an algebra $\langle R, +, \cdot, -, 0, 1\rangle$ such that (R1)–(R3) and (G2) hold.

Example

Algebra of natural numbers

$$Succ : \mathbf{N} \to \mathbf{N} \qquad Succ(x) = x + 1$$

$$Pred : \mathbf{N} \to \mathbf{N} \qquad Pred(x) = \begin{cases} x - 1, & \text{if } x \geq 1; \\ 0, & \text{otherwise.} \end{cases}$$

$$Add : \mathbf{N} \times \mathbf{N} \to \mathbf{N} \qquad Add(x, y) = x + y$$

$$Sub : \mathbf{N} \times \mathbf{N} \to \mathbf{N} \qquad Sub(x, y) = \begin{cases} x - y, & \text{if } x \geq y; \\ 0, & \text{otherwise.} \end{cases}$$

$$Mult : \mathbf{N} \times \mathbf{N} \to \mathbf{N} \qquad Mult(x, y) = x \cdot y$$

$$Quot : \mathbf{N} \times \mathbf{N} \to \mathbf{N} \qquad Quot(x, y) = \begin{cases} (\text{largest } k)(kx \leq y), & \text{if } x \neq 0 ; \\ 0, & \text{if } x = 0. \end{cases}$$

$$Rem : \mathbf{N} \times \mathbf{N} \to \mathbf{N} \qquad Rem(x, y) = y - Quot(x, y) \cdot x$$

$$Exp : \mathbf{N} \times \mathbf{N} \to \mathbf{N} \qquad Exp(x, y) = x^y$$

$$Log : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$$

$$Log(x, y) = \begin{cases} (\text{largest } k)(x^k \leq y), & \text{if } x \neq 0 \text{ and } x \neq 1 \text{ and } y \neq 0 ; \\ 0, & \text{if } x = 0 \text{ or } y = 0 ; \\ 1, & \text{if } x = 1 . \end{cases}$$

$$Max : \mathbf{N} \times \mathbf{N} \to \mathbf{N} \qquad Max(x, y) = \begin{cases} x, & \text{if } x \geq y; \\ y, & \text{otherwise.} \end{cases}$$

$$Min : \mathbf{N} \times \mathbf{N} \to \mathbf{N} \qquad Min(x, y) = \begin{cases} x, & \text{if } x \leq y; \\ y, & \text{otherwise.} \end{cases}$$

$$Fact : \mathbf{N} \to \mathbf{N} \qquad Fact(x) = \begin{cases} x \cdot (x - 1) \cdot \ldots \cdot 2 \cdot 1, & \text{if } x \geq 1; \\ 1, & \text{if } x = 0. \end{cases}$$

Exercise

Find examples of algebras for the previous signatures

Generated algebra

Definition. Let $\mathcal{A}$ be Σ-algebra. For each $X \subseteq A$,

$$\langle X \rangle_{\mathcal{A}} := \bigcap \{B : X \subseteq B \text{ and } B \text{ is a subuniverse of } \mathcal{A}\}.$$

$\langle X \rangle_{\mathcal{A}}$ is called *subuniverse generated by* X.

Let A be any Σ algebra and let $P(A)$ denote the power set of A. Define the map $E : P(A) \to P(A)$ by

$$E(X) = X \cup Cons^A \cup \{ \sigma_A(a_1, \ldots, a_n) \mid n \in \mathbf{N}^+, \sigma \in \Sigma_n \text{ and } a_1, \ldots, a_n \in X \}.$$

We define

$$E^0(X) = X, \; E^{n+1}(X) = E(E^n(X)). \qquad E^\infty(X) = \cup_{n \in \mathbf{N}} E^n(X).$$

Thus *If $E^\infty(X)$ is non-empty then*

$$E^\infty(X) = \langle X \rangle_A.$$

Let A be a Σ-algebra. A is *reachable* if A has no proper subalgebra

Congruences I

Definition. Let $\Sigma = \langle S, \Omega \rangle$ be a signature and $\mathcal{A}$ a Σ-algebra. A *congruence on* $\mathcal{A}$ is a S-relation of equivalence $\langle \equiv_s \rangle_{s \in S}$, such that for each $f : s_1, \ldots, s_n \to s \in \Sigma$ and every $a_1, b_1 \in A_{s_1}, \ldots, a_n, b_n \in A_{s_n}$,

$$\text{se } a_1 \equiv_{s_1} b_i, \ldots, a_n \equiv_{s_n} b_n, \text{ então } f^{\mathcal{A}}(a_1, \ldots, a_n) \equiv_s f^{\mathcal{A}}(b_1, \ldots, b_n) \qquad \text{(CP)}$$

Theorem.
$(\mathrm{Con}(\mathcal{A}), \subset)$ is a complet poset

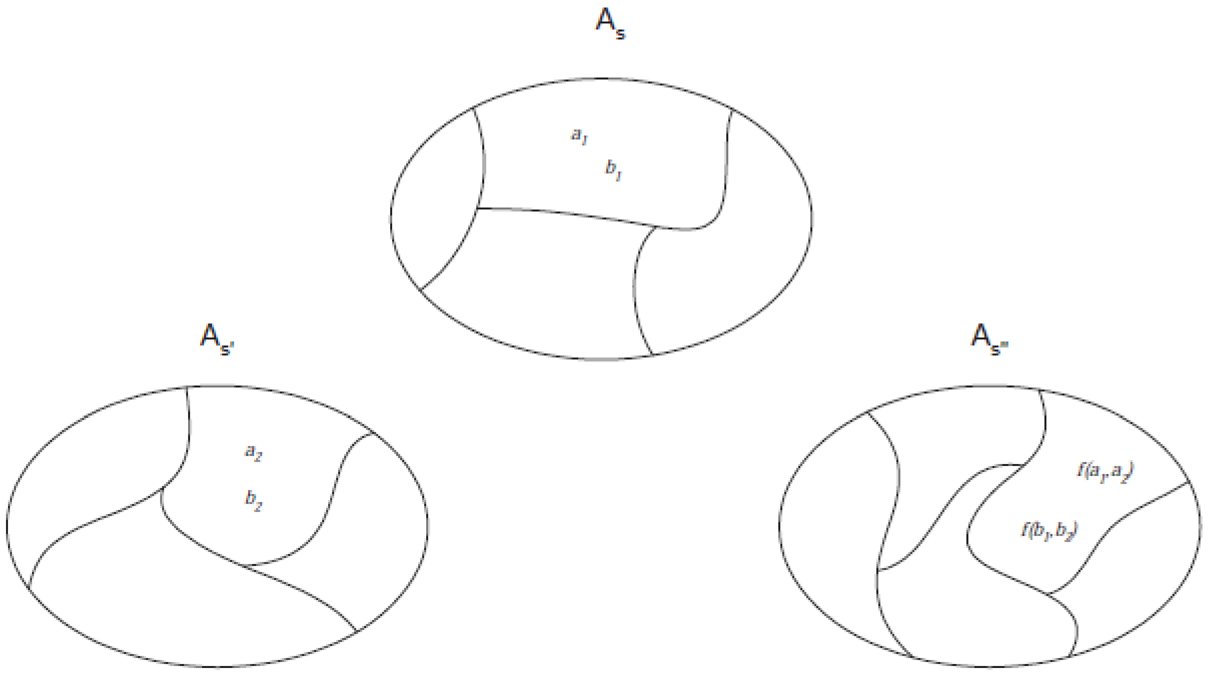
By compatible property (CP) we can define:

Definition. Let $\mathcal{A}$ a Σ-algebra and $\equiv \in \mathrm{Con}(\mathcal{A})$. The *quociente algebra of $\mathcal{A}$ by* $\equiv$ is the Σ-algebra $\mathcal{A}/\equiv$ such that

- $(A/\equiv)_s = A_s/\equiv_s$ for each $s \in S$;
- for each $f : s_1, \ldots, s_n \to s \in \Sigma$, and every $a_1 \in A_{s_1}, \ldots, a_n \in A_{s_n}$,

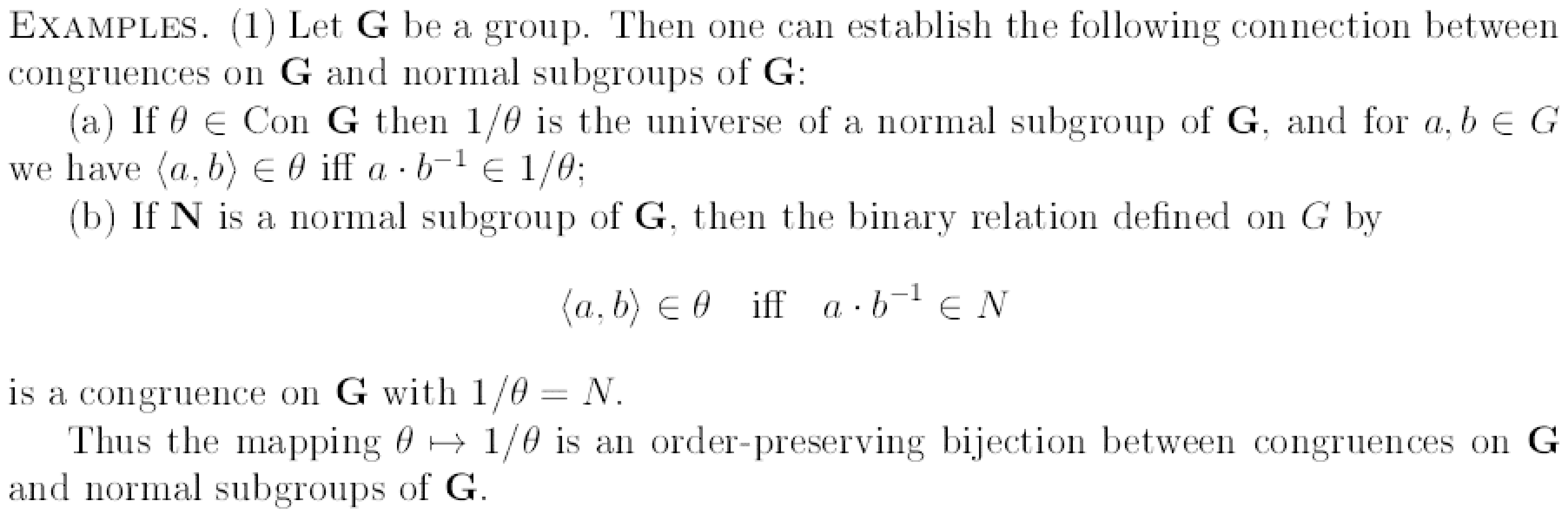
$$f^{\mathcal{A}/\equiv}(a_1/\equiv_{s_1}, \ldots, a_n/\equiv_{s_n}) = f^{\mathcal{A}}(a_1, \ldots, a_n)/\equiv_s .$$

Congruences II



f:s x s’ -> s”

Congruences in Groups



EXAMPLES. (1) Let **G** be a group. Then one can establish the following connection between congruences on **G** and normal subgroups of **G**:

(a) If $\theta \in \text{Con}\,\mathbf{G}$ then $1/\theta$ is the universe of a normal subgroup of **G**, and for $a, b \in G$ we have $\langle a, b\rangle \in \theta$ iff $a \cdot b^{-1} \in 1/\theta$;

(b) If **N** is a normal subgroup of **G**, then the binary relation defined on G by

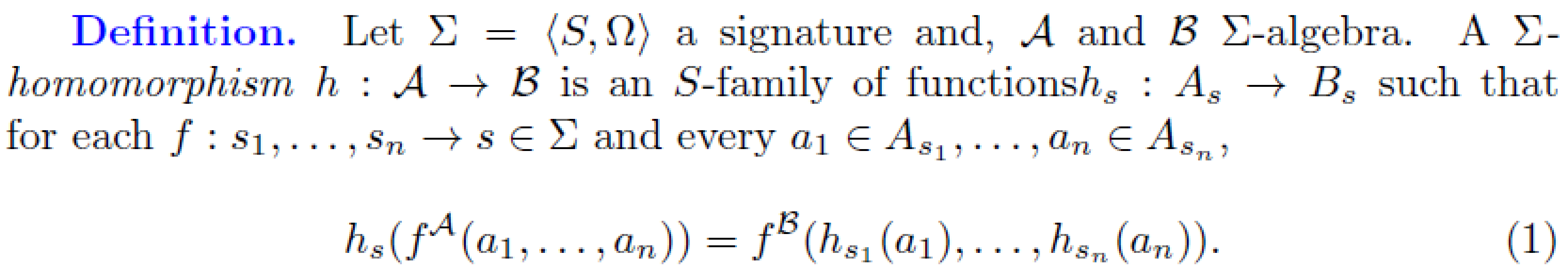
$$\langle a, b\rangle \in \theta \quad \text{iff} \quad a \cdot b^{-1} \in N$$

is a congruence on **G** with $1/\theta = N$.

Thus the mapping $\theta \mapsto 1/\theta$ is an order-preserving bijection between congruences on **G** and normal subgroups of **G**.

There is a similar phenomenon in rings with ideals.

Homomorphisms



Definition. Let $\Sigma = \langle S, \Omega \rangle$ a signature and, $\mathcal{A}$ and $\mathcal{B}$ Σ-algebra. A Σ-*homomorphism* $h : \mathcal{A} \to \mathcal{B}$ is an S-family of functions$h_s : A_s \to B_s$ such that for each $f : s_1, \ldots, s_n \to s \in \Sigma$ and every $a_1 \in A_{s_1}, \ldots, a_n \in A_{s_n}$,

$$h_s(f^{\mathcal{A}}(a_1, \ldots, a_n)) = f^{\mathcal{B}}(h_{s_1}(a_1), \ldots, h_{s_n}(a_n)). \quad (1)$$

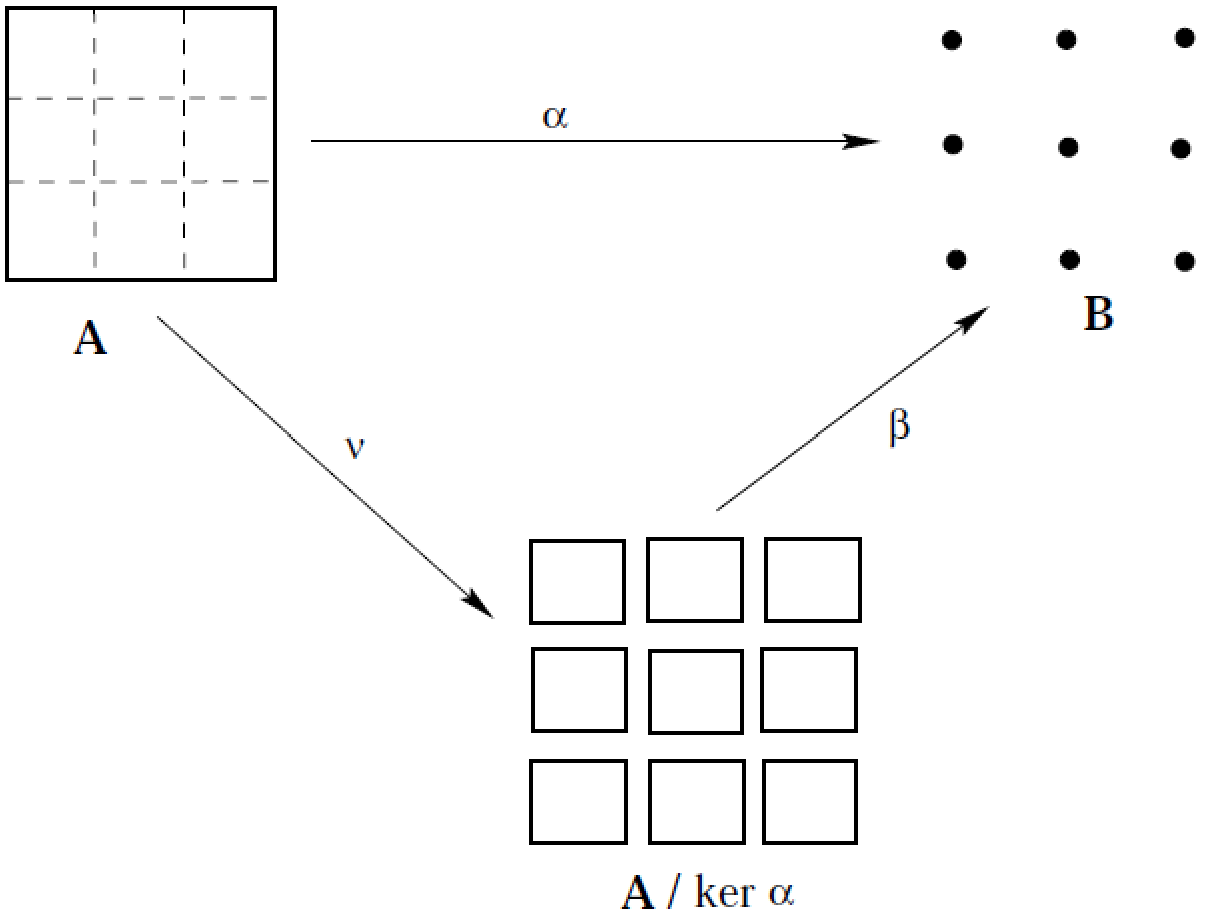
Epimorphisms, monomorphisms and isomorphisms are defined as usual

Homomorphism theorem

Theorem.[Homomorphism theorem]

Let $\mathcal{A}$ and $\mathcal{B}$ be Σ-algebras and, $\alpha : \mathcal{A} \to \mathcal{B}$ a surjective homomorphism.
Then, there is an isomorphism $\beta : \mathcal{A}/\ker(\alpha) \to \mathcal{B}$ such that $\alpha = \beta \circ \nu$, where $\nu : \mathcal{A} \to \mathcal{A}/\ker(\alpha)$ is the natural homomorphism.

Homomorphisms theorem



Corollary

The external problem of finding all homomorphic images of A reduces to the internal problem of finding all congruences on A.