Algebras

Marko Schütz-Schmuck

Department of Computer Science and Engineering
University of Puerto Rico at Mayagüez
Mayagüez, PR

September 11, 2020

what are algebras?

- principal idea: elements (or values) and operations upon them that yield such elements again
- how are they defined?
  - > what are the two key concepts of algebras?
  - > (finite or infinite) carrier set
  - > (finite) set of operations
  - > (A, Ω) : A is the carrier set, Ω is the set of operations
  - > operations ω in Ω have an arity, the number of elements the operation takes e.g. n-ary $\omega : A^n \longrightarrow A$

what are algebras?

- principal idea: elements (or values) and operations upon them that yield such elements again
- how are they defined?
  - what are the two key concepts of algebras?
  - (finite or infinite) carrier set
  - (finite) set of operations
  - (A, Ω) : A is the carrier set, Ω is the set of operations
  - operations ω in Ω have an arity, the number of elements the operation takes e.g. n-ary $\omega : \mathrm{A}^{\mathrm{n}} \rightarrow \mathrm{A}$

what are algebras?

- principal idea: elements (or values) and operations upon them that yield such elements again
- how are they defined?
    - what are the two key concepts of algebras?
    - (finite or infinite) carrier set
    - (finite) set of operations
    - (A, Ω) : A is the carrier set, Ω is the set of operations
    - operations ω in Ω have an arity, the number of elements the operation takes e.g. n-ary $\omega : A^n \to A$

what are algebras?

- principal idea: elements (or values) and operations upon them that yield such elements again
- how are they defined?
    - what are the two key concepts of algebras?
    - (finite or infinite) carrier set
    - (finite) set of operations
    - (A, Ω) : A is the carrier set, Ω is the set of operations
    - operations ω in Ω have an arity, the number of elements the operation takes e.g. n-ary $\omega : A^n \to A$

what are algebras?

- principal idea: elements (or values) and operations upon them that yield such elements again
- how are they defined?
  - what are the two key concepts of algebras?
  - (finite or infinite) carrier set
  - (finite) set of operations
  - (A, Ω) : A is the carrier set, Ω is the set of operations
  - operations ω in Ω have an arity, the number of elements the operation takes e.g. n-ary $\omega : \mathrm{A}^{\mathrm{n}} \rightarrow \mathrm{A}$

what are algebras?

- principal idea: elements (or values) and operations upon them that yield such elements again
- how are they defined?
  - what are the two key concepts of algebras?
  - (finite or infinite) carrier set
  - (finite) set of operations
  - (A, Ω) : A is the carrier set, Ω is the set of operations
  - operations ω in Ω have an arity, the number of elements the operation takes e.g. n-ary $\omega : A^n \to A$

concrete and abstract algebras

- how do concrete algebras differ from abstract algebras?
  - concrete: we know the elements of the carrier set
  - abstract: there are sets that could be used as carrier set, but the specific one is (currently) of no concern

concrete and abstract algebras

- how do concrete algebras differ from abstract algebras?
  - concrete: we know the elements of the carrier set
  - abstract: there are sets that could be used as carrier set, but the specific one is (currently) of no concern

concrete and abstract algebras

- how do concrete algebras differ from abstract algebras?
  - concrete: we know the elements of the carrier set
  - abstract: there are sets that could be used as carrier set, but the specific one is (currently) of no concern

examples of concrete and abstract algebras I

- example concrete algebra: integers with +, -, * as operations $(\mathbb{Z}, \{+, -, *\})$

examples of concrete and abstract algebras II

- example abstract algebra: stacks (without making the implementation nor the elements explicit)
  Introduce a set of stacks S containing elements from a set E, there is an operation empty() and, for any stack s an operation isEmpty(s) that yields a Boolean value. isEmpty is defined by isEmpty(empty()) = true and for all stacks s different from empty() we get isEmpty(s) = false.

examples of concrete and abstract algebras III

- example abstract algebra: stacks (without making the implementation nor the elements explicit) (cont.)
  Given any stack s and any element e other operations are push(s, e) which yields a stack, and if s isn't empty() the operations pop(s) yields a stack and top(s) yields an element. The relationship between the operations is pop(push(s, e)) = s and top(push(s,e)) = e. We have an algebra $(\mathrm{S} \cup \mathrm{E} \cup \mathbb{B}, \{\mathrm{empty}, \mathrm{isEmpty}, \mathrm{push}, \mathrm{pop}, \mathrm{top}\})$

how do they relate to software engineering?

- algebras: closed under operations
- "The Magical Number Seven, Plus or Minus Two"
- reduce conceptual dependencies
- concepts and operations easier to understand if "self-contained"
- light-weight (well-understood) concepts: pure integers, Boolean
  - > add little to complexity

how do they relate to software engineering?

- algebras: closed under operations
- "The Magical Number Seven, Plus or Minus Two"
- reduce conceptual dependencies
- concepts and operations easier to understand if "self-contained"
- light-weight (well-understood) concepts: pure integers, Boolean
  > add little to complexity

how do they relate to software engineering?

- algebras: closed under operations
- "The Magical Number Seven, Plus or Minus Two"
- reduce conceptual dependencies
- concepts and operations easier to understand if "self-contained"
- light-weight (well-understood) concepts: pure integers, Boolean
  - > add little to complexity

how do they relate to software engineering?

- algebras: closed under operations
- "The Magical Number Seven, Plus or Minus Two"
- reduce conceptual dependencies
- concepts and operations easier to understand if "self-contained"
- light-weight (well-understood) concepts: pure integers, Boolean
  - add little to complexity

some examples of algebras in software engineering

- stacks see above
- bank accounts and transfers, examples
  - > transfer : Account -> Account -> Amount -> (Account, Account)
  - > transfer : Account -> Account -> Amount -> (Account, Account, Result)
  - > transfer : (Account, Account, Amount) -> (Account, Account, Amount, Result)
  - > transfer : TransferSpecification -> (TransferSpecification, Result)

some examples of algebras in software engineering

- stacks see above
- bank accounts and transfers, examples
  - transfer : Account -> Account -> Amount -> (Account, Account)
  - transfer : Account -> Account -> Amount -> (Account, Account, Result)
  - transfer : (Account, Account, Amount) -> (Account, Account, Amount, Result)
  - transfer : TransferSpecification -> (TransferSpecification, Result)

some examples of algebras in software engineering

- stacks see above
- bank accounts and transfers, examples
  - transfer : Account -> Account -> Amount -> (Account, Account)
  - transfer : Account -> Account -> Amount -> (Account, Account, Result)
  - transfer : (Account, Account, Amount) -> (Account, Account, Amount, Result)
  - transfer : TransferSpecification -> (TransferSpecification, Result)

some examples of algebras in software engineering

- stacks see above
- bank accounts and transfers, examples
  - transfer : Account -> Account -> Amount -> (Account, Account)
  - transfer : Account -> Account -> Amount -> (Account, Account, Result)
  - transfer : (Account, Account, Amount) -> (Account, Account, Amount, Result)
  - transfer : TransferSpecification -> (TransferSpecification, Result)

some examples of algebras in software engineering

- stacks see above
- bank accounts and transfers, examples
    - transfer : Account -> Account -> Amount -> (Account, Account)
    - transfer : Account -> Account -> Amount -> (Account, Account, Result)
    - transfer : (Account, Account, Amount) -> (Account, Account, Amount, Result)
    - transfer : TransferSpecification -> (TransferSpecification, Result)