

Softwarekonstruktion

im Wintersemester 2012 / 2013

3.1 Algebraische Spezifikation

[inkl. Beiträge von Prof. Volker Gruhn und Dr. Johannes Henkel]

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

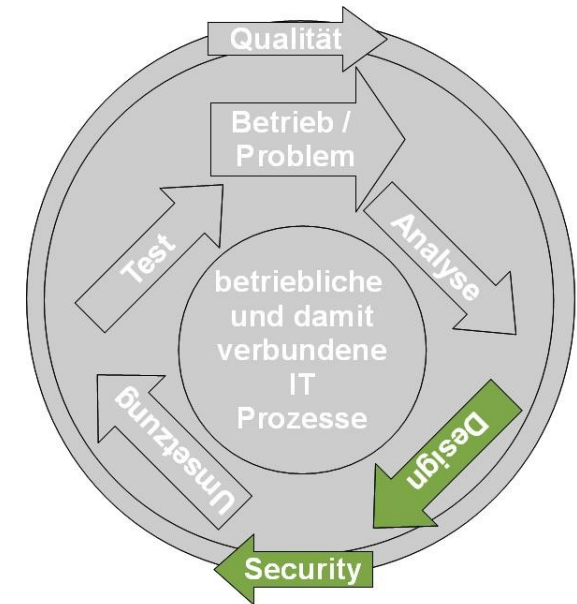
v. 15.01.2013

3.1 Algebraische Spezifikation

[inkl. Beiträge von Prof. Volker Gruhn und Dr. Johannes Henkel]

Einordnung Algebraische Spezifikation

- Qualitätsmanagement
- Testen
- **Modellgetriebene SW-Entwicklung**
 - OCL
 - **Algebraische Spezifikation**



3.1 Algebraische Spezifikation



3.1 Algebraische Spezifikation

Das Spezifikationsproblem

Vollständigkeit der Algebraischen Spezifikation

Signatur und Algebra

Terme

Algebraische Spezifikation: Diskussion

In diesem Kapitel beschäftigen wir uns mit der *algebraischen Spezifikation* als *Grundlage* für die *Spezifikation des Verhaltens einzelner Softwaremodule*.

Warum ?

- *Verhalten einzelner Softwaremodule*: Wie in Teil 1 diskutiert, sind heutige Softwaresysteme von erheblicher Komplexität, die i.A. nur durch Verwendung modularer Herangehensweisen beherrscht werden kann.
- *Spezifikation*: Wie in Teil 1 diskutiert, ist es daher wichtig, das Verhalten einzelner Softwaremodule spezifizieren zu können, um eine modulare Softwareentwicklung und Qualitätssicherung zu ermöglichen.
- *Grundlage*: In der Praxis gibt es verschiedene Ansätze, das Verhalten einzelner Softwaremodule zu spezifizieren, z.B. UML (insbes. Object Constraint Language OCL), Java / C assertions, Java Markup Language (JML). Die formalen Grundlagen dazu werden wir in diesem Abschnitt betrachten.

In einigen Anwendungen kommt algebraische Spezifikation auch direkt zum Einsatz (Beispiel Verifikation von kryptographischen Protokollen: Spezifikation krypto-graphischer Eigenschaften als algebraische Gleichungen; vgl. später).



- Algebraische Spezifikation ...
 - ... betrachtet abstrakte Datentypen als algebraische Strukturen, d.h. als Mengen mit darauf definierten Operationen, für die bestimmte Axiome gelten,
 - ... dient zur formalen Spezifikation des Verhaltens von Modulen / Modulschnittstellen (Unterschied zur Spezifikation des Verhaltens eines gesamten Software-Systems),
 - ... dient zur Verifikation der Implementierung gegen die Spezifikation und der Verifikation der Spezifikation gegenüber den zu erfüllenden Anforderungen.
 - Durch das Zusammensetzen solcher Module kommt man zu Software-Systemen, deren Verhalten durch die Spezifikation der Module und durch die Konstruktion der Module zu einem Software-System definiert ist.
 - Die Konstruktion von Software-Systemen aus Modulen ist Gegenstand des Entwurfs.

Nach [Som10] besteht der Prozess zur Erstellung einer vollständigen algebraischen Spezifikation aus den folgenden Schritten:

- 1) **Strukturieren der Spezifikation:** Zuerst müssen alle Schnittstellen eines Systems identifiziert werden. Das heißt also, es muss herausgefunden werden, welche Teile miteinander kommunizieren. Dazu wird informell beschrieben, welche Operationen notwendig sind.
- 2) **Benennen der Spezifikation:** Alle abstrakten Datentypen einer Spezifikation werden mit einem Namen versehen (z.B. Liste, Stack, Queue, Bool usw.) und es wird entschieden, ob sie generische Parameter benötigen.
- 3) **Auswahl der Operationen:** Hierbei wird die Bezeichnung aller Operationen festgelegt.
- 4) **Informelles Spezifizieren der Operation:** Die identifizierten Operationen sowie deren Auswirkungen auf das System werden beschrieben.
- 5) **Definieren der Syntax:** Für jede identifizierte Operation wird die Syntax formal beschrieben. Dazu werden alle Operationen zusammen mit ihren Parametern definiert. Die Syntax der gesamten Spezifikation ergibt sich aus der Zusammensetzung aller abstrakten Datentypen.
- 6) **Definieren der Semantik:** Die Semantik der Operationen wird definiert, indem beschrieben wird, welche Bedingungen (Axiome) für verschiedene Kombinationen von Operationen immer zutreffen müssen.

[Som10] Sommerville, Ian: Software engineering, ch. 27: Formal Specification. Addison-Wesley Longman Publishing Co., Inc., 2010. http://www.cs.st-andrews.ac.uk/~ifs/Books/SE9/WebChapters/PDF/Ch_27_Forma_spec.pdf

Algebraische Spezifikation eines abstrakten Datentyps besteht aus einem Syntax- und einem Semantikeil:

- Syntax: Menge von Vereinbarungen von Zugriffsoperationen
 - Operationsname: Definitionsbereich \rightarrow Wertebereich
- Semantik: Menge von Gleichungen, die Beziehungen zwischen den Eingabe- und Rückgabewerten der verschiedenen Operationen in Form von Axiomen beschreiben.

Bemerkung: Algebraische Spezifikation bietet zunächst kein Konzept, interne Zustandsänderungen direkt zu beschreiben, sondern allenfalls indirekt, z.B. indem für jede Operation, für die Zustandsänderungen modelliert werden soll, ein Eingabe- und Rückgabewert vom Type „State“ (o.ä.) definiert wird (1). Es gibt allerdings Erweiterungen, die internen Zustand als Modellierungskonzept beinhalten (z.B. Abstract State Machines (2) oder Algebraic State Machines (3)).

(1) vgl. http://www.ldl.jaist.ac.jp/jaist-fssv2010/studentsSlides/zhang_slides.pdf ,
<http://www-plan.cs.colorado.edu/henkel/pubs.html>

(2) http://en.wikipedia.org/wiki/Abstract_state_machines

(3) <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.140.8498&rep=rep1&type=pdf>

algebra X

introduces sorts X,Y,Z

operations

$op1 : X \rightarrow X$

$op2 : X \rightarrow Y$

constraints op1, op2 so that for all

$x, y : X, z : Y$

...

Syntax

Semantik

algebra stack-of-nat **introduces sorts** nat, bool, stack-of-nat
operations

create: \rightarrow stack-of-nat

push: stack-of-nat, nat \rightarrow stack-of-nat

pop: stack-of-nat \rightarrow stack-of-nat

top: stack-of-nat \rightarrow nat

isempty: stack-of-nat \rightarrow bool

constraints create, push, pop, top, isempty
so that for all st: stack-of-nat, n: nat

isempty (create()) = true

isempty (push (st, n)) = false

pop (create()) = create()

pop (push (st, n)) = st

top (create()) = 0

top (push (st, n)) = n

Beispiel für gesamtes Vorgehen: Spezifikation des Datentyps String

Schritt 1) - 4):

- Erzeugen (*new*). Sorte *String* wird dafür gebraucht.
- Verketteten (*append*)
- Zeichen anhängen (*add*). Sorte *Char* wird außerdem gebraucht.
- Länge ermitteln (*length*). Sorte *Nat* wird außerdem gebraucht.
- **NB: Wir definieren hier die Sorte *Nat* als die natürlichen Zahlen inklusive 0.**
- Ermitteln, ob leer (*isEmpty*). Sorte *Bool* wird außerdem gebraucht.
- Gleichheit zweier Zeichenketten (*equal*)

Schritt 4) hier implizit (es ist klar, was z.B. „verketteten“ für Auswirkungen hat).

algebra StringSpec

introduces sorts String, Char, Nat, Bool

operations

new: \rightarrow String

append: String, String \rightarrow String

add: String, Char \rightarrow String

length: String \rightarrow Nat

isEmpty: String \rightarrow Bool

equal: String, String \rightarrow Bool

Syntax

algebra `StringSpec` legt nur die Syntax fest, aber was bedeuten die Operationen?

- Die Semantik der Operationen wird durch **Gleichungen** (engl. Equations) beschreiben.
- Diese Gleichungen formulieren Eigenschaften, die von der Realisierung der Operationen nicht verletzt werden dürfen. Die Gleichungen heißen deshalb auch **Axiome**.
- Alle späteren Realisierungen der Operationen, die die Axiome nicht verletzen, sind spezifikationskonform ! „Lücken“ in den Axiomen (unvollständige Axiomatisierung) sind deshalb fast zwangsläufig die Ursache für spätere Missverständnisse.
- (Informelle) **Definition**: Wir bezeichnen eine algebraische Spezifikation als **unvollständig**, wenn sich nicht alle (intuitiv) als wahr angenommenen Aussagen auch anhand der Axiome ableiten lassen.

algebra StringSpec **introduces sorts** String, Char, Nat, Bool

operations

constraints new, append, add, length, isEmpty, equal

so that for all s,s1,s2: String, c: Char

$\text{isEmpty}(\text{new}()) = \text{true}$

$\text{isEmpty}(\text{add}(s,c)) = \text{false}$

$\text{length}(\text{new}()) = 0$

$\text{length}(\text{add}(s,c)) = \text{length}(s) + 1$

$\text{append}(s, \text{new}()) = s$

$\text{append}(s1, \text{add}(s2,c)) = \text{add}(\text{append}(s1,s2),c)$

$\text{equal}(\text{new}(), \text{new}()) = \text{true}$

$\text{equal}(\text{new}(), \text{add}(s,c)) = \text{false}$

$\text{equal}(\text{add}(s,c), \text{new}()) = \text{false}$

$\text{equal}(\text{add}(s1,c), \text{add}(s2,c)) = \text{equal}(s1,s2)$

Semantik

Frage: Sind diese Gleichungen (intuitiv) „korrekt“ ? Sind sie „vollständig“ ?

„Korrekt“ ? => Die o.g. Gleichungen sind intuitiv korrekt...

... allerdings nur, wenn bestimmte Annahmen an die Intuition erfüllt sind (die im Rahmen von Schritt 4 des Vorgehens bereits vorab informell spezifiziert worden sein sollten). Zum Beispiel:

- Das Axiom „ $\text{append}(s1, \text{add}(s2, c)) = \text{add}(\text{append}(s1, s2), c)$ “ setzt voraus, dass die intendierten Implementierungen von $\text{append}(s1, s2)$ und $\text{add}(s, c)$ konsistent bezüglich der Reihenfolgen von $s1$ und $s2$ bzw s und c sein sollen (z.B. $\text{append}(s1, s2) = [s1::s2]$ und $\text{add}(s, c) = [s::c]$ oder $\text{append}(s1, s2) = [s2::s1]$ und $\text{add}(s, c) = [c::s]$, aber nicht $\text{append}(s1, s2) = [s1::s2]$ und $\text{add}(c, s) = [c::s]$, wobei $[s1::s2]$ die Konkatenation von $s1$ gefolgt von $s2$ ist).
- Das Axiom „ $\text{isEmpty}(\text{add}(s, c)) = \text{false}$ “ setzt voraus, dass es kein „leeres“ Zeichen geben soll.

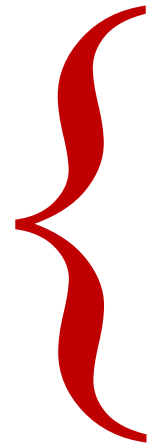
Vollständig ? Nein. Zum Beispiel: Die Gleichungen implizieren nicht:

$\text{equal}(s1, s2) = \text{false}$ (für $s1 \# s2$ mit $\text{length}(s1) = \text{length}(s2)$)
obwohl dies intuitiv gelten sollte.

Anmerkung: Kommutativität von equal folgt allerdings schon aus den Folien der vorigen Folie (Beweis per Induktion).

3.1 Algebraische Spezifikation

3.1 Algebraische Spezifikation



Das Spezifikationsproblem

Vollständigkeit der Algebraischen Spezifikation

Signatur und Algebra

Terme

Algebraische Spezifikation: Diskussion

Frage:

Gilt

(i) $\text{append}(\text{new}(), \text{add}(\text{new}(), c)) = \text{add}(\text{new}(), c)$??

Frage:

Gilt

$$(i) \text{ append (new(), add (new(), c)) = add (new(), c) \quad ??}$$

Start vom Axiom:

$$(ii) \text{ append (s1, add(s2,c)) = add (append (s1,s2),c)}$$

Frage:

Gilt

$$(i) \text{ append (new(), add (new(), c)) = add (new(), c) \quad ??}$$

Start vom Axiom:

$$(ii) \text{ append (s1, add(s2,c)) = add (append (s1,s2),c)}$$

Wir ersetzen hier s1 und s2 durch new():

$$(iii) \text{ append (new(), add (new(),c)) = add (append (new(), new()),c)}$$

Frage:

Gilt

$$(i) \text{ append (new(), add (new(), c)) = add (new(), c) \quad ??}$$

Start vom Axiom:

$$(ii) \text{ append (s1, add(s2,c)) = add (append (s1,s2),c)}$$

Wir ersetzen hier s1 und s2 durch new():

$$(iii) \text{ append (new(), add (new(),c)) = add (append (new(), new()),c)}$$

Im Axiom $\text{append (s, new()) = s}$ ersetzen wir s durch new() und erhalten:

$$(iv) \text{ append (new(), new()) = new()}$$

Frage:

Gilt

$$(i) \text{ append}(\text{new}(), \text{add}(\text{new}(), c)) = \text{add}(\text{new}(), c) \quad ??$$

Start vom Axiom:

$$(ii) \text{ append}(s1, \text{add}(s2, c)) = \text{add}(\text{append}(s1, s2), c)$$

Wir ersetzen hier $s1$ und $s2$ durch $\text{new}()$:

$$(iii) \text{ append}(\text{new}(), \text{add}(\text{new}(), c)) = \text{add}(\text{append}(\text{new}(), \text{new}()), c)$$

Im Axiom $\text{append}(s, \text{new}()) = s$ ersetzen wir s durch $\text{new}()$ und erhalten:

$$(iv) \text{ append}(\text{new}(), \text{new}()) = \text{new}()$$

In (iii) setzen wir (iv) ein und erhalten:

$$(v) \text{ append}(\text{new}(), \text{add}(\text{new}(), c)) = \text{add}(\text{new}(), c)$$

womit (i) gezeigt ist.

Spezifikation des Datentyps String: Vollständigkeit der Axiome II



Frage: Gilt (i) $\text{append}(\text{new}(), s) = s$??

Frage: Gilt (i) $\text{append}(\text{new}(), s) = s$??

- Induktionsanfang

Offensichtlich gilt (i) für $s=\text{new}()$, denn $\text{append}(\text{new}(),\text{new}()) = \text{new}()$ gilt nach Axiom $\text{append}(s,\text{new}()) = s$

Frage: Gilt (i) $\text{append}(\text{new}(), s) = s$??

- Induktionsanfang

Offensichtlich gilt (i) für $s=\text{new}()$, denn $\text{append}(\text{new}(),\text{new}()) = \text{new}()$ gilt nach Axiom $\text{append}(s,\text{new}()) = s$

- Induktionsannahme:

Wir wählen ein beliebiges s' sodass $s=\text{add}(s',c)$ und nehmen an, dass (i) für s' gilt.

Frage: Gilt (i) $\text{append}(\text{new}(), s) = s$??

- Induktionsanfang

Offensichtlich gilt (i) für $s = \text{new}()$, denn $\text{append}(\text{new}(), \text{new}()) = \text{new}()$ gilt nach Axiom $\text{append}(s, \text{new}()) = s$

- Induktionsannahme:

Wir wählen ein beliebiges s' sodass $s = \text{add}(s', c)$ und nehmen an, dass (i) für s' gilt.

- Nachweis des Axioms

Damit gilt:

$\text{append}(\text{new}(), s) =$

$\text{append}(\text{new}(), \text{add}(s', c)) =$

$\text{add}(\text{append}(\text{new}(), s'), c) =$

$\text{add}(s', c) =$

s

Induktionsannahme

Axiom $\text{append}(s1, \text{add}(s2, c))$
 $= \text{add}(\text{append}(s1, s2), c)$

Induktionsannahme

Induktionsannahme

Frage: Gilt (i) $\text{append}(\text{new}(), s) = s$??

- Induktionsanfang

Offensichtlich gilt (i) für $s=\text{new}()$, denn $\text{append}(\text{new}(),\text{new}()) = \text{new}()$ gilt nach Axiom $\text{append}(s,\text{new}()) = s$

- Induktionsannahme:

Wir wählen ein beliebiges s' sodass $s=\text{add}(s',c)$ und nehmen an, dass (i) für s' gilt.

- Nachweis des Axioms

Damit gilt:

$$\text{append}(\text{new}(),s) =$$

$$\text{append}(\text{new}(), \text{add}(s',c)) =$$

$$\text{add}(\text{append}(\text{new}(),s'),c) =$$

$$\text{add}(s',c) =$$

s

Induktionsannahme

$$\begin{aligned} \text{Axiom } \text{append}(s1, \text{add}(s2,c)) \\ = \text{add}(\text{append}(s1,s2),c) \end{aligned}$$

Induktionsannahme

Induktionsannahme

Frage: Für welche Elemente in String beweist der Induktionsbeweis die o.g. Gleichung ?

Frage: Gilt (i) $\text{append}(\text{new}(), s) = s$??

- Induktionsanfang

Offensichtlich gilt (i) für $s=\text{new}()$, denn $\text{append}(\text{new}(),\text{new}()) = \text{new}()$ gilt nach Axiom $\text{append}(s,\text{new}()) = s$

- Induktionsannahme:

Wir wählen ein beliebiges s' sodass $s=\text{add}(s',c)$ und nehmen an, dass (i) für s' gilt.

- Nachweis des Axioms

Damit gilt:

$$\text{append}(\text{new}(),s) =$$

$$\text{append}(\text{new}(), \text{add}(s',c)) =$$

$$\text{add}(\text{append}(\text{new}(),s'),c) =$$

$$\text{add}(s',c) =$$

s

Induktionsannahme

$$\begin{aligned} \text{Axiom } \text{append}(s1, \text{add}(s2,c)) \\ = \text{add}(\text{append}(s1,s2),c) \end{aligned}$$

Induktionsannahme

Induktionsannahme

Frage: Für welche Elemente in String beweist der Induktionsbeweis die o.g. Gleichung ?
Antwort: Die Strings, die durch new und add erzeugt werden.

Der letzte Beweis basiert auf der Annahme, dass alle Strings durch die Operationen `new` und `add` erzeugt werden (denn nur für diese gilt der Beweis).

Zur Ermittlung der erforderlichen Axiome und zur Sicherstellung der Beweiskraft von Induktionsbeweisen wird eine erzeugende Menge von Operationen identifiziert.

Definition: Eine Menge O von Operationen heisst „**generierend**“ für eine Menge X , wenn alle Elemente in X durch sukzessive Anwendung der Operationen erzeugt werden können.

Bemerkung: Insbesondere kann die Menge O null-stellige Operationen (= Konstanten) enthalten.

Frage:

- Was ist eine generierende Menge von Operationen der Booleschen Algebra ?
- Was ist eine generierende Menge von Operationen der Algebra der positiven, ganzen Zahlen ?

Frage:

- Was ist eine generierende Menge von Operationen der Booleschen Algebra ?

Antwort: zum Beispiel {false, not}

- Was ist eine generierende Menge von Operationen der Algebra der positiven, ganzen Zahlen ?

Frage:

- Was ist eine generierende Menge von Operationen der Booleschen Algebra ?

Antwort: zum Beispiel {false, not}

- Was ist eine generierende Menge von Operationen der Algebra der positiven, ganzen Zahlen ?

Antwort: zum Beispiel {zero, succ}

Im Rahmen der algebraischen Spezifikation können wir definieren, dass Typen von bestimmten Operationen generiert werden:

constraints *<operations>* **so that**

<Type> **generated by** [*<generating ops>*]

for all *<variables>*: *<Type>* ...

Beispiel:

constraints new, append, add, length, isEmpty, equal **so that**

String **generated by** [new,add,Char]

for all s,s1,s2: String, c: Char

Bemerkung: Hier wird Char als Menge von Atomen (generierende Konstanten) vorausgesetzt.

Erweiterung der algebraischen Spezifikation *String* um Konstanten 'a', 'b' von Typ Char (formal: nullstellige Operation $a: () \rightarrow 'a'$).

Frage:

```
isEmpty(new()) = true
isEmpty(add (s,c)) = false
length (new()) = 0
length (add(s,c)) = length(s) + 1
append (s, new()) = s
append (s1, add(s2,c)) = add (append(s1,s2),c)
equal (new(), new()) = true
equal (new(), add(s,c)) = false
equal (add(s,c), new()) = false
equal (add(s1,c),add(s2,c)) = equal (s1,s2)
```

$\text{equal} (\text{add} (s, 'a'), \text{add} (s, 'b')) = \text{false} ?$

Erweiterung der algebraischen Spezifikation *String* um Konstanten 'a', 'b' von Typ Char (formal: nullstellige Operation $a: () \rightarrow 'a'$).

Frage:

```
isEmpty(new()) = true
isEmpty(add (s,c)) = false
length (new()) = 0
length (add(s,c)) = length(s) + 1
append (s, new()) = s
append (s1, add(s2,c)) = add (append(s1,s2),c)
equal (new(), new()) = true
equal (new(), add(s,c)) = false
equal (add(s,c), new()) = false
equal (add(s1,c),add(s2,c)) = equal (s1,s2)
```

$\text{equal}(\text{add}(s, 'a'), \text{add}(s, 'b')) = \text{false} ?$

- Sollte intuitiv gelten, lässt sich aber nicht aus den Axiomen ableiten.
- => Die algebraische Spezifikation ist unvollständig, weil sich nicht alle (intuitiv) als wahr angenommenen Aussagen beweisen lassen.

Vervollständigung von *String* durch zusätzliche Operation

$\text{equalC} : \text{Char}, \text{Char} \rightarrow \text{Bool}$

mit den Axiomen:

$$\text{equalC} ('a', 'a') = \text{true}$$

$$\text{equalC} ('a', 'b') = \text{false}$$

$$\text{equalC} ('b', 'a') = \text{false}$$

$$\text{equalC} ('b', 'b') = \text{true}$$

und der Ersetzung des Axioms

$$\text{equal} (\text{add} (s1, c), \text{add} (s2, c)) = \text{equal} (s1, s2) \quad \text{durch}$$

$$\text{equal} (\text{add} (s1, c1), \text{add} (s2, c2)) = \\ \text{equal} (s1, s2) \wedge \text{equalC} (c1, c2)$$

Frage:

$\text{equal}(\text{add}(s, 'a'), \text{add}(s, 'b'))$
 $= \text{false} ?$

```
isEmpty(new()) = true
isEmpty(add(s,c)) = false
length(new()) = 0
length(add(s,c)) = length(s) + 1
append(s, new()) = s
append(s1, add(s2,c)) = add(append(s1,s2),c)
equal(new(), new()) = true
equal(new(), add(s,c)) = false
equal(add(s,c), new()) = false

equal(add(s1,c), add(s2,c)) = equal(s1,s2)

equalC('a','a') = true           (USW...)
equalC('a','b') = false
equal(add(s1,c1), add(s2,c2))
    = equal(s1,s2) ^ equalC(c1,c2)
```

Frage:

`equal (add (s,'a'), add (s,'b'))`
`= false ?`

Antwort:

`equal (add (s,'a'), add (s,'b'))`

```
isEmpty(new()) = true
isEmpty(add (s,c)) = false
length (new()) = 0
length (add(s,c)) = length(s) + 1
append (s, new()) = s
append (s1, add(s2,c)) = add (append(s1,s2),c)
equal (new(), new()) = true
equal (new(), add(s,c)) = false
equal (add(s,c), new()) = false

equal (add(s1,c), add(s2,c)) = equal (s1,s2)

equalC ('a','a') = true           (USW...)
equalC ('a','b') = false
equal (add (s1,c1), add (s2,c2))
      = equal (s1,s2) ^ equalC (c1,c2)
```

Frage:

`equal (add (s,'a'), add (s,'b'))`
`= false ?`

Antwort:

`equal (add (s,'a'), add (s,'b'))`
`= equal (s,s) \wedge equalC ('a','b')`

```
isEmpty(new()) = true
isEmpty(add (s,c)) = false
length (new()) = 0
length (add(s,c)) = length(s) + 1
append (s, new()) = s
append (s1, add(s2,c)) = add (append(s1,s2),c)
equal (new(), new()) = true
equal (new(), add(s,c)) = false
equal (add(s,c), new()) = false

equal (add(s1,c), add(s2,c)) = equal (s1,s2)
                                     (USW...)
equalC ('a','a') = true
equalC ('a','b') = false
equal (add (s1,c1), add (s2,c2))
      = equal (s1,s2)  $\wedge$  equalC (c1,c2)
```

Frage:

$\text{equal}(\text{add}(s, 'a'), \text{add}(s, 'b'))$
 $= \text{false} ?$

Antwort:

$\text{equal}(\text{add}(s, 'a'), \text{add}(s, 'b'))$
 $= \text{equal}(s, s) \wedge \text{equalC}('a', 'b')$
 $= \text{true} \wedge \text{false} = \text{false}$

```
isEmpty(new()) = true
isEmpty(add(s,c)) = false
length(new()) = 0
length(add(s,c)) = length(s) + 1
append(s, new()) = s
append(s1, add(s2,c)) = add(append(s1,s2),c)
equal(new(), new()) = true
equal(new(), add(s,c)) = false
equal(add(s,c), new()) = false

equal(add(s1,c), add(s2,c)) = equal(s1,s2)

equalC('a','a') = true           (usw...)
equalC('a','b') = false
equal(add(s1,c1), add(s2,c2))
    = equal(s1,s2) ^ equalC(c1,c2)
```

Operationen:

- Erzeugen einer neuen Datei („file“): `newF`
- Testen, ob eine Datei leer ist: `isEmptyF`
- Anfügen einer Zeichenkette an eine Datei: `addF`
- Einfügen einer Zeichenkette an einer bestimmten Position einer Datei: `insertF`
- Hintereinanderhängen zweier Dateien: `appendF`

algebra TextEditor **introduces**
sorts Text, String, Char, Bool, Nat
operations

$\text{newF}: () \rightarrow \text{Text}$

$\text{isEmptyF}: \text{Text} \rightarrow \text{Bool}$

$\text{addF}: \text{Text}, \text{String} \rightarrow \text{Text}$

$\text{insertF}: \text{Text}, \text{Nat}, \text{String} \rightarrow \text{Text}$

$\text{appendF}: \text{Text}, \text{Text} \rightarrow \text{Text}$

$\text{lenghtF}: \text{Text} \rightarrow \text{Nat}$

$\text{equalF}: \text{Text}, \text{Text} \rightarrow \text{Bool}$

$\text{addFC}: \text{Text}, \text{Char} \rightarrow \text{Text}$

Operationen aus String implizit weiter verfügbar. Mit Suffix „S“ versehen um Verwechslung zu vermeiden.

Hilfsoperation für addF.

constraints newF, isEmptyF, addF, appendF, insertF
so that for all [f, f₁, f₂: Text; s: String; c: Char; cursor: Nat]

isEmptyF (newF()) = true
isEmptyF (addFC (f,c)) = false
addF (f, newS()) = f
addF (f, addS(s,c)) = addFC (addF(f,s), c)
lengthF (newF()) = 0
lengthF (addFC(f,c)) = lengthF(f) + 1
appendF (f, newF()) = f
appendF (f₁, addFC(f₂,c)) = addFC (appendF(f₁,f₂), c)
...

...

$$\text{equalF}(\text{newF}(), \text{newF}()) = \text{true}$$
$$\text{equalF}(\text{newF}(), \text{addFC}(f, c)) = \text{false}$$
$$\text{equalF}(\text{addFC}(f, c), \text{newF}()) = \text{false}$$
$$\text{equalF}(\text{addFC}(f_1, c_1), \text{addFC}(f_2, c_2)) =$$
$$\text{equalF}(f_1, f_2) \wedge \text{equalC}(c_1, c_2)$$
$$\text{insertF}(f, \text{cursor}, \text{newS}()) = f$$
$$[(\text{equalF}(f, \text{appendF}(f_1, f_2))) \wedge (\text{lengthF}(f_1) = \text{cursor})]$$
$$\Rightarrow \text{equalF}(\text{insertF}(f, \text{cursor} + 1, s), \text{appendF}(\text{addF}(f_1, s), f_2))]$$

...

$\text{equalF}(\text{newF}(), \text{newF}()) = \text{true}$

$\text{equalF}(\text{newF}(), \text{addFC}(f, c)) = \text{false}$

$\text{equalF}(\text{addFC}(f, c), \text{newF}()) = \text{false}$

$\text{equalF}(\text{addFC}(f_1, c_1), \text{addFC}(f_2, c_2)) = \text{equalF}(f_1, f_2) \wedge \text{equalC}(c_1, c_2)$

$\text{insertF}(f, \text{cursor}, \text{newS}()) = f$ (*)

$[(\text{equalF}(f, \text{appendF}(f_1, f_2)) \wedge (\text{lengthF}(f_1) = \text{cursor}))$
 $\Rightarrow \text{equalF}(\text{insertF}(f, \text{cursor}+1, s), \text{appendF}(\text{addF}(f_1, s), f_2))]$

Frage: $\text{insertF}(\text{newF}(), 0, s) = ?$

...

$\text{equalF}(\text{newF}(), \text{newF}()) = \text{true}$

$\text{equalF}(\text{newF}(), \text{addFC}(f, c)) = \text{false}$

$\text{equalF}(\text{addFC}(f, c), \text{newF}()) = \text{false}$

$\text{equalF}(\text{addFC}(f_1, c_1), \text{addFC}(f_2, c_2)) = \text{equalF}(f_1, f_2) \wedge \text{equalC}(c_1, c_2)$

$\text{insertF}(f, \text{cursor}, \text{newS}()) = f$ (*)

$[(\text{equalF}(f, \text{appendF}(f_1, f_2)) \wedge (\text{lengthF}(f_1) = \text{cursor}))$

$\Rightarrow \text{equalF}(\text{insertF}(f, \text{cursor}+1, s), \text{appendF}(\text{addF}(f_1, s), f_2))]$

Frage: $\text{insertF}(\text{newF}(), 0, s) = ?$

Antwort:

- $\text{insertF}(\text{newF}(), 0, \text{newS}()) = \text{newF}()$

...

$\text{equalF}(\text{newF}(), \text{newF}()) = \text{true}$

$\text{equalF}(\text{newF}(), \text{addFC}(f, c)) = \text{false}$

$\text{equalF}(\text{addFC}(f, c), \text{newF}()) = \text{false}$

$\text{equalF}(\text{addFC}(f_1, c_1), \text{addFC}(f_2, c_2)) = \text{equalF}(f_1, f_2) \wedge \text{equalC}(c_1, c_2)$

$\text{insertF}(f, \text{cursor}, \text{newS}()) = f$ (*)

$[(\text{equalF}(f, \text{appendF}(f_1, f_2)) \wedge (\text{lengthF}(f_1) = \text{cursor}))$
 $\Rightarrow \text{equalF}(\text{insertF}(f, \text{cursor}+1, s), \text{appendF}(\text{addF}(f_1, s), f_2))]$

Frage: $\text{insertF}(\text{newF}(), 0, s) = ?$

Antwort:

- $\text{insertF}(\text{newF}(), 0, \text{newS}()) = \text{newF}()$
- unbestimmt für $s \neq \text{newS}()$:
für $\text{cursor} = 0$ gilt nur Gleichung (*)

...

$\text{equalF}(\text{newF}(), \text{newF}()) = \text{true}$

$\text{equalF}(\text{newF}(), \text{addFC}(f, c)) = \text{false}$

$\text{equalF}(\text{addFC}(f, c), \text{newF}()) = \text{false}$

$\text{equalF}(\text{addFC}(f_1, c_1), \text{addFC}(f_2, c_2)) = \text{equalF}(f_1, f_2) \wedge \text{equalC}(c_1, c_2)$

$\text{insertF}(f, \text{cursor}, \text{newS}()) = f$ (*)

$[(\text{equalF}(f, \text{appendF}(f_1, f_2)) \wedge (\text{lengthF}(f_1) = \text{cursor}))$
 $\Rightarrow \text{equalF}(\text{insertF}(f, \text{cursor}+1, s), \text{appendF}(\text{addF}(f_1, s), f_2))]$

Frage: $\text{insertF}(\text{newF}(), 1, s) = ?$

...

$\text{equalF}(\text{newF}(), \text{newF}()) = \text{true}$

$\text{equalF}(\text{newF}(), \text{addFC}(f, c)) = \text{false}$

$\text{equalF}(\text{addFC}(f, c), \text{newF}()) = \text{false}$

$\text{equalF}(\text{addFC}(f_1, c_1), \text{addFC}(f_2, c_2)) = \text{equalF}(f_1, f_2) \wedge \text{equalC}(c_1, c_2)$

$\text{insertF}(f, \text{cursor}, \text{newS}()) = f$ (*)

$[(\text{equalF}(f, \text{appendF}(f_1, f_2)) \wedge (\text{lengthF}(f_1) = \text{cursor}))$
 $\Rightarrow \text{equalF}(\text{insertF}(f, \text{cursor}+1, s), \text{appendF}(\text{addF}(f_1, s), f_2))]$

Antwort: $\text{insertF}(\text{newF}(), 1, s)$
 $= \text{appendF}(\text{addF}(\text{newF}(), s), \text{newF}())$

...

$\text{equalF}(\text{newF}(), \text{newF}()) = \text{true}$

$\text{equalF}(\text{newF}(), \text{addFC}(f, c)) = \text{false}$

$\text{equalF}(\text{addFC}(f, c), \text{newF}()) = \text{false}$

$\text{equalF}(\text{addFC}(f_1, c_1), \text{addFC}(f_2, c_2)) = \text{equalF}(f_1, f_2) \wedge \text{equalC}(c_1, c_2)$

$\text{insertF}(f, \text{cursor}, \text{newS}()) = f$ (*)

$[(\text{equalF}(f, \text{appendF}(f_1, f_2)) \wedge (\text{lengthF}(f_1) = \text{cursor}))$
 $\Rightarrow \text{equalF}(\text{insertF}(f, \text{cursor}+1, s), \text{appendF}(\text{addF}(f_1, s), f_2))]$

Antwort: $\text{insertF}(\text{newF}(), 1, s)$
 $= \text{appendF}(\text{addF}(\text{newF}(), s), \text{newF}())$
 $= \text{addF}(\text{newF}(), s)$ [$\text{appendF}(f, \text{newF}()) = f$]

...

$\text{equalF}(\text{newF}(), \text{newF}()) = \text{true}$

$\text{equalF}(\text{newF}(), \text{addFC}(f, c)) = \text{false}$

$\text{equalF}(\text{addFC}(f, c), \text{newF}()) = \text{false}$

$\text{equalF}(\text{addFC}(f_1, c_1), \text{addFC}(f_2, c_2)) = \text{equalF}(f_1, f_2) \wedge \text{equalC}(c_1, c_2)$

$\text{insertF}(f, \text{cursor}, \text{newS}()) = f \quad (*)$

$[(\text{equalF}(f, \text{appendF}(f_1, f_2)) \wedge (\text{lengthF}(f_1) = \text{cursor}))$
 $\Rightarrow \text{equalF}(\text{insertF}(f, \text{cursor}+1, s), \text{appendF}(\text{addF}(f_1, s), f_2))]$

Antwort: $\text{insertF}(\text{newF}(), 1, s)$

$= \text{appendF}(\text{addF}(\text{newF}(), s), \text{newF}())$

$= \text{addF}(\text{newF}(), s) \quad [\text{appendF}(f, \text{newF}()) = f]$

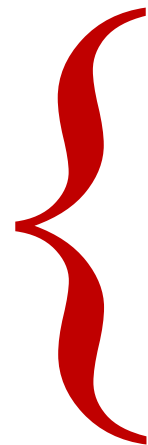
$= \text{addFC}(\text{addF}(\text{newF}(), s'), c) \quad \text{für } s = \text{addS}(s', c)$

$= \dots \quad [\text{addF}(f, \text{addS}(s, c)) = \text{addFC}(\text{addF}(f, s), c)]$

3.1 Algebraische Spezifikation



3.1 Algebraische Spezifikation



Das Spezifikationsproblem

Vollständigkeit der Algebraischen Spezifikation

Signatur und Algebra

Terme

Algebraische Spezifikation: Diskussion

$\Sigma = (S, F)$ heißt algebraische Signatur

- S eine Menge von Sorten (Typen)
- F eine Menge von Operationssymbolen
- Auf F ist eine Abbildung definiert: $type: F \rightarrow S^* \times S$
wobei $S^0 = \{\emptyset\}$ und $S^* = \bigcup_{n \in \mathbb{N}} S^n$
- Für $type(f) = (s_1, \dots, s_n, s)$ schreiben wir $f: s_1, \dots, s_n \rightarrow s$
- $type(f)$ bezeichnet man auch als *Signatur* der Operation f .
- *Konstanten* sind die Abbildungen aus F , deren Vorbereich die leere Menge ist.

$\Sigma_{\text{Test}} = (S, F)$ mit

$S = \{\text{Nat}, \text{Bool}\}$

$F = \{\text{zero}, \text{one}, \text{succ}, \text{add}, \text{equal}, \text{equalBool}, \text{T}, \text{F}\}$

$\text{zero}: \quad \quad \quad \rightarrow \text{Nat} \quad \quad \quad [\text{d.h. type}(\text{zero})=(\emptyset, \text{Nat})]$

$\text{one}: \quad \quad \quad \rightarrow \text{Nat}$

$\text{succ}: \text{Nat} \quad \quad \quad \rightarrow \text{Nat}$

$\text{add}: \quad \quad \text{Nat} \times \text{Nat} \quad \rightarrow \text{Nat}$

$\text{equal}: \quad \quad \text{Nat} \times \text{Nat} \quad \rightarrow \text{Bool} \quad [\text{d.h. type}(\text{equal})=(\text{Nat}, \text{Nat}, \text{Bool})]$

$\text{equalBool}: \text{Bool} \times \text{Bool} \rightarrow \text{Bool}$

$\text{T}: \quad \quad \quad \rightarrow \text{Bool}$

$\text{F}: \quad \quad \quad \rightarrow \text{Bool}$

Eine Algebra interpretiert eine Signatur Σ , indem sie die Symbole in Σ mit konkreten Mengen und Abbildungen „füllt“.

Σ -Algebra (Definition)

- Sei $\Sigma = (S, F)$ eine Signatur.
- Für alle $s \in S$ sei A_s eine Menge.
- Für alle $f \in F$ mit $f: s_1, \dots, s_n \rightarrow s$ sei $f_A: A_{s_1} \times \dots \times A_{s_n} \rightarrow A_s$ eine Abbildung.
- Dann ist das Paar $A = ((A_s)_{s \in S}, (f_A)_{f \in F})$ eine Σ -Algebra.
- A_s bezeichnet man auch als Trägermenge von A zu s , f_A die Interpretation der Operation f .
- Die Beschreibung der Operation f in der Algebra A umfasst ihre Signatur $\text{type}(f)$ und ihre Semantik f_A .
- Die Semantik von Operationen wird axiomatisch definiert.

Σ -Algebra: Beispiel

Signatur Σ_{Test}	Σ_{Test} -Algebra A
Nat	$A_{\text{Nat}} =_{\text{def}} \mathbb{N}$
Bool	$A_{\text{Bool}} =_{\text{def}} \{\text{true}, \text{false}\}$
zero: \rightarrow Nat	$\text{zero}_A =_{\text{def}} 0$
one: \rightarrow Nat	$\text{one}_A =_{\text{def}} 1$
succ: Nat \rightarrow Nat	$\forall n \in \mathbb{N}. \text{succ}_A(n) =_{\text{def}} n + 1$
add: Nat x Nat \rightarrow Nat	$\forall n, m \in \mathbb{N}. \text{add}_A(n, m) =_{\text{def}} n + m$
equal: Nat x Nat \rightarrow Bool	$\text{equal}_A(\text{zero}_A, \text{zero}_A) =_{\text{def}} \text{true}$ $\forall n, m \in \mathbb{N}. \text{equal}_A(\text{succ}_A(n), \text{succ}_A(m)) =_{\text{def}} \text{equal}_A(n, m)$ $\forall n, m \in \mathbb{N}. \text{equal}_A(n, m) =_{\text{def}} \text{equal}_A(m, n)$ $\forall m \in \mathbb{N}. \text{equal}_A(\text{zero}_A, \text{add}_A(\text{one}_A, m)) =_{\text{def}} \text{false}$

Anmerkung: Die Axiome definieren jeweils eindeutig eine Funktion (müsste
genaugenommen jeweils bewiesen werden).

Signatur Σ_{Test}	Σ_{Test} -Algebra A
$\text{equalBool} : \text{Bool} \times \text{Bool} \rightarrow \text{Bool}$	$\text{equalBool}_A (\text{true}, \text{true}) =_{\text{def}} \text{true}$ $\text{equalBool}_A (\text{false}, \text{false}) =_{\text{def}} \text{true}$ $\text{equalBool}_A (\text{true}, \text{false}) =_{\text{def}} \text{false}$ $\text{equalBool}_A (\text{false}, \text{true}) =_{\text{def}} \text{false}$
$T : \rightarrow \text{Bool}$	$T_A =_{\text{def}} \text{true}$
$F : \rightarrow \text{Bool}$	$F_A =_{\text{def}} \text{false}$

- Beispiele mit genau einer Trägermenge:
 - Σ_{Nat} -Algebra B (A_{Nat} mit den relevanten Operationen und Axiomen aus Σ_{Test})
 - Gruppen, Ringe, Körper, Verbände
- Beispiele mit mehreren Trägermengen:
 - Σ_{Test} -Algebra A
 - Vektorräume (Vektor und Skalare als Träger)

Wie wir in Teil 1 gesehen haben, ist ein wichtiges Ziel ist die Wiederverwendbarkeit von Software-Komponenten.

Frage: Wie können Ansätze, die auf algebraischer Spezifikation basieren, dieses Ziel unterstützen ?

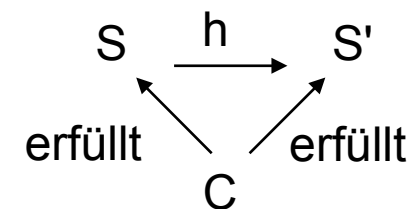
Wie wir in Teil 1 gesehen haben, ist ein wichtiges Ziel die Wiederverwendbarkeit von Software-Komponenten.

Frage: Wie können Ansätze, die auf algebraischer Spezifikation basieren, dieses Ziel unterstützen ?

Antwort: Für eine gegebene algebraische Spezifikation S einer Software-Komponente kann ich eine Alt-Komponente C wiederverwenden, sofern sie die Spezifikation S erfüllt.

Frage: Was tun, wenn die Alt-Komponente C gegen eine (etwas) abweichenden algebraischen Spezifikation S' entwickelt wurde ?

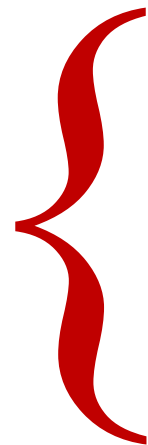
Antwort: Definiere eine Abbildung h zwischen den algebraischen Spezifikationen S und S' , die die wesentlichen gewünschten Eigenschaften bewahrt (genannt „Homomorphismus“ bzw. „Isomorphismus“).



3.1 Algebraische Spezifikation



3.1 Algebraische Spezifikation



Das Spezifikationsproblem

Vollständigkeit der Algebraischen Spezifikation

Signatur und Algebra

Terme

Algebraische Spezifikation: Diskussion

Σ -Grundterm

- Signatur: Syntax
- Algebra: Semantik
- Terme: Wörter der Sprache
 - Beschreibt syntaktische Struktur auf Ebene der Signatur
 - Interpretation durch die jeweilige Algebra
 - z.B. `succ(add(n1,n2))`
 - Grundterme: Terme ohne Variablen
 - Terme: Terme mit (oder ohne) Variablen

Definition: Σ -Grundterm

Σ -Grundterm (Definition)

Sei $\Sigma = (S, F)$ eine Signatur.

Die Familie $T_\Sigma = (T(\Sigma, s))_{s \in S}$ der Mengen von Σ -Grundtermen

besteht für jedes $s \in S$ aus der kleinsten Teilmenge $T(\Sigma, s)$ von Wörtern über dem Alphabet $(F \cup \{„(“, „)“, „,“, „{“, „}“\})$, für die gilt:

- für jede Konstante $c: \rightarrow s$ aus der Menge F gilt $c \in T(\Sigma, s)$ und
- für jede Funktion $f: s_1 \dots s_n \rightarrow s$ aus der Menge F und alle Terme $t_i \in T(\Sigma, s_i)$ (für alle $i \in \{1, \dots, n\}$) gilt $f(t_1, \dots, t_n) \in T(\Sigma, s)$.

Konstanten

Operations-
symbole +
zusammen-
gesetzte
Terme



Σ -Grundterm (Beispiel)

- Sei gegeben $\Sigma_{\text{Nat}} = (S, F)$ aus vorherigem Beispiel
- $T(\Sigma_{\text{Nat}}, \text{Nat})$
 - Stufe 0: **Konstanten**
 - Stufe 1: **Operationen angewendet auf Stufe 0**
 - Stufe 2: **etc.**

 - Stufe 3: ...
 -
- ...

Signatur Σ_{Nat}
Nat
Bool
one: $\rightarrow \text{Nat}$
succ: $\text{Nat} \rightarrow \text{Nat}$
add: $\text{Nat} \times \text{Nat} \rightarrow \text{Nat}$
T: $\rightarrow \text{Bool}$



Σ -Grundterm (Beispiel)

- Sei gegeben $\Sigma_{\text{Nat}} = (S, F)$ aus vorherigem Beispiel
- $T(\Sigma_{\text{Nat}}, \text{Nat})$
 - Stufe 0: **one**
 - Stufe 1: **Operationen angewendet auf Stufe 0**
 - Stufe 2: **etc.**

 - Stufe 3: ...
 -
- ...

Signatur Σ_{Nat}
Nat
Bool
one: $\rightarrow \text{Nat}$
succ: $\text{Nat} \rightarrow \text{Nat}$
add: $\text{Nat} \times \text{Nat} \rightarrow \text{Nat}$
T: $\rightarrow \text{Bool}$



Σ -Grundterm (Beispiel)

- Sei gegeben $\Sigma_{\text{Nat}} = (S, F)$ aus vorherigem Beispiel
- $T(\Sigma_{\text{Nat}}, \text{Nat})$
 - Stufe 0: **one**
 - Stufe 1: **succ(one), add(one,one)**
 - Stufe 2: **etc.**

 - Stufe 3: ...
 -
- ...

Signatur Σ_{Nat}
Nat
Bool
one: $\rightarrow \text{Nat}$
succ: $\text{Nat} \rightarrow \text{Nat}$
add: $\text{Nat} \times \text{Nat} \rightarrow \text{Nat}$
T: $\rightarrow \text{Bool}$

Σ -Grundterm: Beispiel

Σ -Grundterm (Beispiel)

- Sei gegeben $\Sigma_{\text{Nat}} = (S, F)$ aus vorherigem Beispiel
- $T(\Sigma_{\text{Nat}}, \text{Nat})$
 - Stufe 0: one
 - Stufe 1: succ(one), add(one,one)
 - Stufe 2: succ(succ(one)), succ(add(one,one)),
add(succ(one),one), add(one,succ(one)),
add(add(one,one),succ(one)),...
 - Stufe 3: succ(succ(succ(one))),...
 -
- Vollständige Beschreibung per Rekursion
 - $T(\Sigma_{\text{Nat}}, \text{Nat}) = \{\text{one}\} \cup \{\text{succ}(n) \mid n \in T(\Sigma_{\text{Nat}}, \text{Nat})\}$
 $\cup \{\text{add}(n_1, n_2) \mid \{n_1, n_2\} \subseteq T(\Sigma_{\text{Nat}}, \text{Nat})\}$

Signatur Σ_{Nat}
Nat
Bool
one: \rightarrow Nat
succ: Nat \rightarrow Nat
add: Nat x Nat \rightarrow Nat
T: \rightarrow Bool

Definition: Auswertung von Σ -Grundtermen

Sei $\Sigma = (S, F)$ eine Signatur und A eine Σ -Algebra.

Die **Auswertung** (Evaluation / Interpretation) $\text{eval}(A)$

der Terme zur Signatur Σ in A ist eine Familie von Abbildungen

$$\text{eval}(A) = (\text{eval}(A)_s: T(\Sigma, s) \rightarrow A_s)_{s \in S}$$

Sie ist definiert für alle $s \in S$ durch:

Konstanten

- für jede Konstante $c: \rightarrow s$ aus der Menge F gilt:
 $\text{eval}(A)_s(c) = c_A$

Operations-
symbole +
zusammen-
gesetzte
Terme

- für jede Funktion $f: s_1 \dots s_n \rightarrow s$ aus der Menge F und alle Terme $t_i \in T(\Sigma, s_i)$ (für alle $i \in \{1, \dots, n\}$) gilt:
$$\text{eval}(A)_s(f(t_1, \dots, t_n)) = f_A(\text{eval}(A)_{s_1}(t_1), \dots, \text{eval}(A)_{s_n}(t_n))$$

Beispiel für Auswertung von Σ -Grundtermen: Σ_{Nat} -Algebren A und A2

- $\text{eval}(A)_{\text{Nat}} (\text{succ}(\text{add}(\text{one}, \text{one})))$

= ?

= ?

= ?

= ?

= ?

= ?

- $\text{eval}(A2)_{\text{Nat}} (\text{succ}(\text{add}(\text{one}, \text{one})))$

= ?

= ?

= ?

= ?

= ?

= ?

$$[\text{eval}(A)_s(f(t_1, \dots, t_n)) = f_A(\text{eval}(A)_{s_1}(t_1), \dots, \text{eval}(A)_{s_n}(t_n))]$$

[„]

$$[\text{eval}(A)_s(c) = c_A]$$

Beispiel für Auswertung von Σ -Grundtermen: Σ_{Nat} -Algebren A und A2

- $\text{eval}(A)_{\text{Nat}} (\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_A(\text{eval}(A)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$
= ?
= ?
= ?
= ?
= ?

$$[\text{eval}(A)_s(f(t_1, \dots, t_n)) = f_A(\text{eval}(A)_{s_1}(t_1), \dots, \text{eval}(A)_{s_n}(t_n))]]$$

[„]

$$[\text{eval}(A)_s(c) = c_A]$$

- $\text{eval}(A2)_{\text{Nat}} (\text{succ}(\text{add}(\text{one}, \text{one})))$
= ?
= ?
= ?
= ?
= ?
= ?

Beispiel für Auswertung von Σ -Grundtermen: Σ_{Nat} -Algebren A und A2

- $\text{eval}(A)_{\text{Nat}} (\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_A(\text{eval}(A)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$ [$\text{eval}(A)_s(f(t_1, \dots, t_n)) = f_A(\text{eval}(A)_{s_1}(t_1), \dots, \text{eval}(A)_{s_n}(t_n))$]
= $\text{succ}_A(\text{add}_A(\text{eval}(A)_{\text{Nat}}(\text{one}), \text{eval}(A)_{\text{Nat}}(\text{one})))$ [„]
= ? [$\text{eval}(A)_s(c) = c_A$]
= ?
= ?
= ?
- $\text{eval}(A2)_{\text{Nat}} (\text{succ}(\text{add}(\text{one}, \text{one})))$
= ?
= ?
= ?
= ?
= ?
= ?

Beispiel für Auswertung von Σ -Grundtermen: Σ_{Nat} -Algebren A und A2

- $\text{eval}(A)_{\text{Nat}} (\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_A(\text{eval}(A)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$ [$\text{eval}(A)_s(f(t_1, \dots, t_n)) = f_A(\text{eval}(A)_{s_1}(t_1), \dots, \text{eval}(A)_{s_n}(t_n))$]
= $\text{succ}_A(\text{add}_A(\text{eval}(A)_{\text{Nat}}(\text{one}), \text{eval}(A)_{\text{Nat}}(\text{one})))$ [„]
= $\text{succ}_A(\text{add}_A(\text{one}_A, \text{one}_A))$ [$\text{eval}(A)_s(c) = c_A$]
= ?
= ?
= ?
- $\text{eval}(A2)_{\text{Nat}} (\text{succ}(\text{add}(\text{one}, \text{one})))$
= ?
= ?
= ?
= ?
= ?
= ?

Beispiel für Auswertung von Σ -Grundtermen: Σ_{Nat} -Algebren A und A2

- $\text{eval}(A)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_A(\text{eval}(A)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$ [$\text{eval}(A)_s(f(t_1, \dots, t_n)) = f_A(\text{eval}(A)_{s_1}(t_1), \dots, \text{eval}(A)_{s_n}(t_n))$]
= $\text{succ}_A(\text{add}_A(\text{eval}(A)_{\text{Nat}}(\text{one}), \text{eval}(A)_{\text{Nat}}(\text{one})))$ [„]
= $\text{succ}_A(\text{add}_A(\text{one}_A, \text{one}_A))$ [$\text{eval}(A)_s(c) = c_A$]
= $\text{succ}_A(\text{add}_A(1, 1))$
= ?
= ?
- $\text{eval}(A2)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= ?
= ?
= ?
= ?
= ?
= ?

Beispiel für Auswertung von Σ -Grundtermen: Σ_{Nat} -Algebren A und A2

- $\text{eval}(A)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_A(\text{eval}(A)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$ [$\text{eval}(A)_s(f(t_1, \dots, t_n)) = f_A(\text{eval}(A)_{s_1}(t_1), \dots, \text{eval}(A)_{s_n}(t_n))$]
= $\text{succ}_A(\text{add}_A(\text{eval}(A)_{\text{Nat}}(\text{one}), \text{eval}(A)_{\text{Nat}}(\text{one})))$ [„]
= $\text{succ}_A(\text{add}_A(\text{one}_A, \text{one}_A))$ [$\text{eval}(A)_s(c) = c_A$]
= $\text{succ}_A(\text{add}_A(1, 1))$
= $\text{succ}_A(2)$
= ?
- $\text{eval}(A2)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= ?
= ?
= ?
= ?
= ?
= ?

Beispiel für Auswertung von Σ -Grundtermen: Σ_{Nat} -Algebren A und A2

- $\text{eval}(A)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_A(\text{eval}(A)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$ [$\text{eval}(A)_s(f(t_1, \dots, t_n)) = f_A(\text{eval}(A)_{s_1}(t_1), \dots, \text{eval}(A)_{s_n}(t_n))$]
= $\text{succ}_A(\text{add}_A(\text{eval}(A)_{\text{Nat}}(\text{one}), \text{eval}(A)_{\text{Nat}}(\text{one})))$ [„]
= $\text{succ}_A(\text{add}_A(\text{one}_A, \text{one}_A))$ [$\text{eval}(A)_s(c) = c_A$]
= $\text{succ}_A(\text{add}_A(1, 1))$
= $\text{succ}_A(2)$
= 3
- $\text{eval}(A2)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= ?
= ?
= ?
= ?
= ?
= ?

Beispiel für Auswertung von Σ -Grundtermen: Σ_{Nat} -Algebren A und A2

- $\text{eval}(A)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_A(\text{eval}(A)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$ [$\text{eval}(A)_s(f(t_1, \dots, t_n)) = f_A(\text{eval}(A)_{s_1}(t_1), \dots, \text{eval}(A)_{s_n}(t_n))$]
= $\text{succ}_A(\text{add}_A(\text{eval}(A)_{\text{Nat}}(\text{one}), \text{eval}(A)_{\text{Nat}}(\text{one})))$ [„]
= $\text{succ}_A(\text{add}_A(\text{one}_A, \text{one}_A))$ [$\text{eval}(A)_s(c) = c_A$]
= $\text{succ}_A(\text{add}_A(1, 1))$
= $\text{succ}_A(2)$
= 3
- $\text{eval}(A2)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_{A2}(\text{eval}(A2)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$
= ?
= ?
= ?
= ?
= ?

Beispiel für Auswertung von Σ -Grundtermen: Σ_{Nat} -Algebren A und A2

- $\text{eval}(A)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_A(\text{eval}(A)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$ [$\text{eval}(A)_s(f(t_1, \dots, t_n)) = f_A(\text{eval}(A)_{s_1}(t_1), \dots, \text{eval}(A)_{s_n}(t_n))$]
= $\text{succ}_A(\text{add}_A(\text{eval}(A)_{\text{Nat}}(\text{one}), \text{eval}(A)_{\text{Nat}}(\text{one})))$ [„]
= $\text{succ}_A(\text{add}_A(\text{one}_A, \text{one}_A))$ [$\text{eval}(A)_s(c) = c_A$]
= $\text{succ}_A(\text{add}_A(1, 1))$
= $\text{succ}_A(2)$
= 3
- $\text{eval}(A2)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_{A2}(\text{eval}(A2)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_{A2}(\text{add}_{A2}(\text{eval}(A2)_{\text{Nat}}(\text{one}), \text{eval}(A2)_{\text{Nat}}(\text{one})))$
= ?
= ?
= ?
= ?

Beispiel für Auswertung von Σ -Grundtermen: Σ_{Nat} -Algebren A und A2

- $\text{eval}(A)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_A(\text{eval}(A)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$ [$\text{eval}(A)_s(f(t_1, \dots, t_n)) = f_A(\text{eval}(A)_{s_1}(t_1), \dots, \text{eval}(A)_{s_n}(t_n))$]
= $\text{succ}_A(\text{add}_A(\text{eval}(A)_{\text{Nat}}(\text{one}), \text{eval}(A)_{\text{Nat}}(\text{one})))$ [„]
= $\text{succ}_A(\text{add}_A(\text{one}_A, \text{one}_A))$ [$\text{eval}(A)_s(c) = c_A$]
= $\text{succ}_A(\text{add}_A(1, 1))$
= $\text{succ}_A(2)$
= 3
- $\text{eval}(A2)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_{A2}(\text{eval}(A2)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_{A2}(\text{add}_{A2}(\text{eval}(A2)_{\text{Nat}}(\text{one}), \text{eval}(A2)_{\text{Nat}}(\text{one})))$
= $\text{succ}_{A2}(\text{add}_{A2}(\text{one}_{A2}, \text{one}_{A2}))$
= ?
= ?
= ?

Beispiel für Auswertung von Σ -Grundtermen: Σ_{Nat} -Algebren A und A2

- $\text{eval}(A)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_A(\text{eval}(A)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$ [$\text{eval}(A)_s(f(t_1, \dots, t_n)) = f_A(\text{eval}(A)_{s_1}(t_1), \dots, \text{eval}(A)_{s_n}(t_n))$]
= $\text{succ}_A(\text{add}_A(\text{eval}(A)_{\text{Nat}}(\text{one}), \text{eval}(A)_{\text{Nat}}(\text{one})))$ [„]
= $\text{succ}_A(\text{add}_A(\text{one}_A, \text{one}_A))$ [$\text{eval}(A)_s(c) = c_A$]
= $\text{succ}_A(\text{add}_A(1, 1))$
= $\text{succ}_A(2)$
= 3
- $\text{eval}(A2)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_{A2}(\text{eval}(A2)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_{A2}(\text{add}_{A2}(\text{eval}(A2)_{\text{Nat}}(\text{one}), \text{eval}(A2)_{\text{Nat}}(\text{one})))$
= $\text{succ}_{A2}(\text{add}_{A2}(\text{one}_{A2}, \text{one}_{A2}))$
= $\text{succ}_{A2}(\text{add}_{A2}(2, 2))$
= ?
= ?

Beispiel für Auswertung von Σ -Grundtermen: Σ_{Nat} -Algebren A und A2

- $\text{eval}(A)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_A(\text{eval}(A)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$ [$\text{eval}(A)_s(f(t_1, \dots, t_n)) = f_A(\text{eval}(A)_{s_1}(t_1), \dots, \text{eval}(A)_{s_n}(t_n))$]
= $\text{succ}_A(\text{add}_A(\text{eval}(A)_{\text{Nat}}(\text{one}), \text{eval}(A)_{\text{Nat}}(\text{one})))$ [„]
= $\text{succ}_A(\text{add}_A(\text{one}_A, \text{one}_A))$ [$\text{eval}(A)_s(c) = c_A$]
= $\text{succ}_A(\text{add}_A(1, 1))$
= $\text{succ}_A(2)$
= 3
- $\text{eval}(A2)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_{A2}(\text{eval}(A2)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_{A2}(\text{add}_{A2}(\text{eval}(A2)_{\text{Nat}}(\text{one}), \text{eval}(A2)_{\text{Nat}}(\text{one})))$
= $\text{succ}_{A2}(\text{add}_{A2}(\text{one}_{A2}, \text{one}_{A2}))$
= $\text{succ}_{A2}(\text{add}_{A2}(2, 2))$
= $\text{succ}_{A2}(4)$
= ?

Beispiel für Auswertung von Σ -Grundtermen: Σ_{Nat} -Algebren A und A2

- $\text{eval}(A)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_A(\text{eval}(A)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$ [$\text{eval}(A)_s(f(t_1, \dots, t_n)) = f_A(\text{eval}(A)_{s_1}(t_1), \dots, \text{eval}(A)_{s_n}(t_n))$]
= $\text{succ}_A(\text{add}_A(\text{eval}(A)_{\text{Nat}}(\text{one}), \text{eval}(A)_{\text{Nat}}(\text{one})))$ [„]
= $\text{succ}_A(\text{add}_A(\text{one}_A, \text{one}_A))$ [$\text{eval}(A)_s(c) = c_A$]
= $\text{succ}_A(\text{add}_A(1, 1))$
= $\text{succ}_A(2)$
= 3
- $\text{eval}(A2)_{\text{Nat}}(\text{succ}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_{A2}(\text{eval}(A2)_{\text{Nat}}(\text{add}(\text{one}, \text{one})))$
= $\text{succ}_{A2}(\text{add}_{A2}(\text{eval}(A2)_{\text{Nat}}(\text{one}), \text{eval}(A2)_{\text{Nat}}(\text{one})))$
= $\text{succ}_{A2}(\text{add}_{A2}(\text{one}_{A2}, \text{one}_{A2}))$
= $\text{succ}_{A2}(\text{add}_{A2}(2, 2))$
= $\text{succ}_{A2}(4)$
= 6

Signatur mit Variablen (Definition):

- Sei $\Sigma = (S, F)$ eine Signatur. Sei $X = (X_s)_{s \in S}$ eine Familie von Mengen, die durch S indiziert ist (dies wird als „S-sortiert“ oder „S-indiziert“ bezeichnet) und sodass für jedes $s \in S$ gilt: $X_s \cap F = \{\}$. Die Elemente der Mengen X_s werden dann als „Variablen“ bezeichnet.
- Dann heißt $\Sigma = (S, F, X)$ **Signatur mit Variablen**.

Definition: Allgemeine Terme

Sei $\Sigma = (S, F, X)$ eine Signatur mit Variablen und $X = (X_s)_{s \in S}$ eine Familie von Mengen. Die Familie $T_\Sigma(X) = (T(\Sigma, s)(X))_{s \in S}$ der Mengen von **(allgemeinen) Σ -Termen** besteht für jedes $s \in S$ aus der kleinsten Teilmenge $T(\Sigma, s)(X)$ von Wörtern über dem Alphabet $(F \cup X \cup \{„(“, „)“, „,“, „.“, „{“, „}\})$, für die gilt:

Variablen

- für jede Variable $x \in X_s$ gilt $x \in T(\Sigma, s)(X)$,

Konstanten

- für jede Konstante $c: \rightarrow s$ aus der Menge F gilt $c \in T(\Sigma, s)(X)$ und

Operations-
symbole +
zusammen-
gesetzte
Terme

- für jede Funktion $f: s_1 \dots s_n \rightarrow s$ aus der Menge F und alle Terme $t_i \in T(\Sigma, s_i)(X)$ (für alle $i \in \{1, \dots, n\}$) gilt $f(t_1, \dots, t_n) \in T(\Sigma, s)(X)$.

Variablen spielen dieselbe syntaktische Rolle wie Konstantensymbole, unterscheiden sich jedoch in der Semantik:

- Konstanten haben in Σ -Algebra eine feste Bedeutung.
- Variablen müssen für eine Termauswertung erst belegt werden !

Definition: Auswertung von allgemeinen Σ -Termen:

Sei $\Sigma = (S, F, X)$ eine Signatur mit Variablen, A eine Σ -Algebra und $\text{ass}: X \rightarrow A$ eine Variablenbelegung („assignment“).

Die erweiterte **Auswertung** $\text{xeval}(\text{ass}): T_{\Sigma}(X) \rightarrow A$ **der Terme zur Signatur Σ in A (bzgl. ass)** ist eine Familie von Abbildungen

- $(\text{xeval}(\text{ass})_s: T(\Sigma, s)(X) \rightarrow A_s)_{s \in S}$

Sie ist definiert für alle $s \in S$ durch:

Variablen

- für jede Variable $x \in X_s$ sei $\text{xeval}(\text{ass})_s(x) =_{\text{def}} (\text{ass})_s(x)$,

Konstanten

- für jede Konstante $c: \rightarrow s$ aus der Menge F sei $\text{xeval}(\text{ass})_s(c) =_{\text{def}} c_A$ und

Operations-
symbole +
zusammen-
gesetzte
Terme

- für jede Funktion $f: s_1 \dots s_n \rightarrow s$ aus der Menge F und alle Terme $t_i \in T(\Sigma, s_i)(X)$ (für alle $i \in \{1, \dots, n\}$) sei $\text{xeval}(\text{ass})_s(f(t_1, \dots, t_n)) =_{\text{def}} f_A(\text{xeval}(\text{ass})_{s_1}(t_1), \dots, \text{xeval}(\text{ass})_{s_n}(t_n))$

Beispiel: Auswertung von allgemeinen Σ -Termen:

- Signatur Σ -Nat, Algebra A
- $X_{\text{Nat}} = \{n_1, n_2\}$
- Variablenbelegung $v: X \rightarrow A$
 - $v_{\text{Nat}}(n_1) = 1909$
 - $v_{\text{Nat}}(n_2) = 5$
- $\text{xeval}(v)_{\text{Nat}}(\text{add}(\text{succ}(\text{zero}()), \text{succ}(n_2)))$
= ?

Beispiel: Auswertung von allgemeinen Σ -Termen:

- Signatur Σ -Nat, Algebra A
- $X_{\text{Nat}} = \{n_1, n_2\}$
- Variablenbelegung $v: X \rightarrow A$
 - $v_{\text{Nat}}(n_1) = 1909$
 - $v_{\text{Nat}}(n_2) = 5$
- $\text{xeval}(v)_{\text{Nat}}(\text{add}(\text{succ}(\text{zero}()), \text{succ}(n_2)))$ [$\text{xeval}(\text{ass})_s(f(t_1, \dots, t_n)) =_{\text{def}} f_A(\text{xeval}(\text{ass})_{s_1}(t_1), \dots, \text{eval}(\text{ass})_{s_n}(t_n))$]
= $\text{add}_A(\text{xeval}(v)_{\text{Nat}}(\text{succ}(\text{zero}())), \text{xeval}(v)_{\text{Nat}}(\text{succ}(n_2)))$
= ?

Beispiel: Auswertung von allgemeinen Σ -Termen:

- Signatur Σ -Nat, Algebra A
- $X_{\text{Nat}} = \{n_1, n_2\}$
- Variablenbelegung $v: X \rightarrow A$
 - $v_{\text{Nat}}(n_1) = 1909$
 - $v_{\text{Nat}}(n_2) = 5$
- $\text{xeval}(v)_{\text{Nat}}(\text{add}(\text{succ}(\text{zero}()), \text{succ}(n_2)))$ [$\text{xeval}(\text{ass})_s(f(t_1, \dots, t_n)) =_{\text{def}} f_A(\text{xeval}(\text{ass})_{s_1}(t_1), \dots, \text{eval}(\text{ass})_{s_n}(t_n))$]
= $\text{add}_A(\text{xeval}(v)_{\text{Nat}}(\text{succ}(\text{zero}())), \text{xeval}(v)_{\text{Nat}}(\text{succ}(n_2)))$
= $\text{add}_A(\text{succ}_A(\text{xeval}(v)_{\text{Nat}}(\text{zero}())), \text{succ}_A(\text{xeval}(v)_{\text{Nat}}(n_2)))$
= ?

Beispiel: Auswertung von allgemeinen Σ -Termen:

- Signatur Σ -Nat, Algebra A
- $X_{\text{Nat}} = \{n_1, n_2\}$
- Variablenbelegung $v: X \rightarrow A$
 - $v_{\text{Nat}}(n_1) = 1909$
 - $v_{\text{Nat}}(n_2) = 5$
- $\text{xeval}(v)_{\text{Nat}}(\text{add}(\text{succ}(\text{zero}()), \text{succ}(n_2)))$ [$\text{xeval}(\text{ass})_s(f(t_1, \dots, t_n)) =_{\text{def}} f_A(\text{xeval}(\text{ass})_{s_1}(t_1), \dots, \text{eval}(\text{ass})_{s_n}(t_n))$]
= $\text{add}_A(\text{xeval}(v)_{\text{Nat}}(\text{succ}(\text{zero}())), \text{xeval}(v)_{\text{Nat}}(\text{succ}(n_2)))$
= $\text{add}_A(\text{succ}_A(\text{xeval}(v)_{\text{Nat}}(\text{zero}())), \text{succ}_A(\text{xeval}(v)_{\text{Nat}}(n_2)))$
= $\text{add}_A(\text{succ}_A(\text{zero}()), \text{succ}_A(v_{\text{Nat}}(n_2)))$
= ?

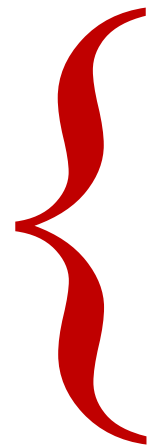
Beispiel: Auswertung von allgemeinen Σ -Termen:

- Signatur Σ -Nat, Algebra A
- $X_{\text{Nat}} = \{n_1, n_2\}$
- Variablenbelegung $v: X \rightarrow A$
 - $v_{\text{Nat}}(n_1) = 1909$
 - $v_{\text{Nat}}(n_2) = 5$
- $\text{xeval}(v)_{\text{Nat}}(\text{add}(\text{succ}(\text{zero}()), \text{succ}(n_2)))$ [$\text{xeval}(\text{ass})_s(f(t_1, \dots, t_n)) =_{\text{def}} f_A(\text{xeval}(\text{ass})_{s_1}(t_1), \dots, \text{eval}(\text{ass})_{s_n}(t_n))$]
 - $= \text{add}_A(\text{xeval}(v)_{\text{Nat}}(\text{succ}(\text{zero}())), \text{xeval}(v)_{\text{Nat}}(\text{succ}(n_2)))$
 - $= \text{add}_A(\text{succ}_A(\text{xeval}(v)_{\text{Nat}}(\text{zero}())), \text{succ}_A(\text{xeval}(v)_{\text{Nat}}(n_2)))$
 - $= \text{add}_A(\text{succ}_A(\text{zero}()), \text{succ}_A(v_{\text{Nat}}(n_2)))$
 - $= \text{add}_A(\text{succ}_A(\text{zero}()), \text{succ}_A(5))$ [Def. succ_A]
 - $= \text{add}_A(1, 6)$ [Def. add_A]
 - $= 7$

3.1 Algebraische Spezifikation



3.1 Algebraische Spezifikation



Das Spezifikationsproblem

Vollständigkeit der Algebraischen Spezifikation

Signatur und Algebra

Terme

Algebraische Spezifikation: Diskussion

Algebraische Signatur Σ

- Sorten S
- Operationssymbole F
 - Operationssymbole bilden abstrakt von Sorten auf Sorten ab
 - Konstanten bilden auf Sorten ab.
- ggf. Variablen X

Term t

- Verknüpfungen von Konstanten und Operationen (\rightarrow Grundterme) sowie ggf. Variablen (\rightarrow Allgemeine Terme)
- Entsprechend den Regeln der Algebra

algebraische Spezifikation SP

- Signatur Σ und wohldefinierte Formeln E (Gleichungen und ggf. logische Verknüpfungen)

Algebra

- Signatur Σ
- Mengen A
 - Konkrete Ausprägungen der Sorten
- Operationen f
 - konkrete Definition der Operationssymbole

Gleichung e

- Verknüpfung von zwei Termen
- Gültig, wenn beide Terme (ggf. für alle Variablenbelegungen) identisch ausgewertet werden

Modell

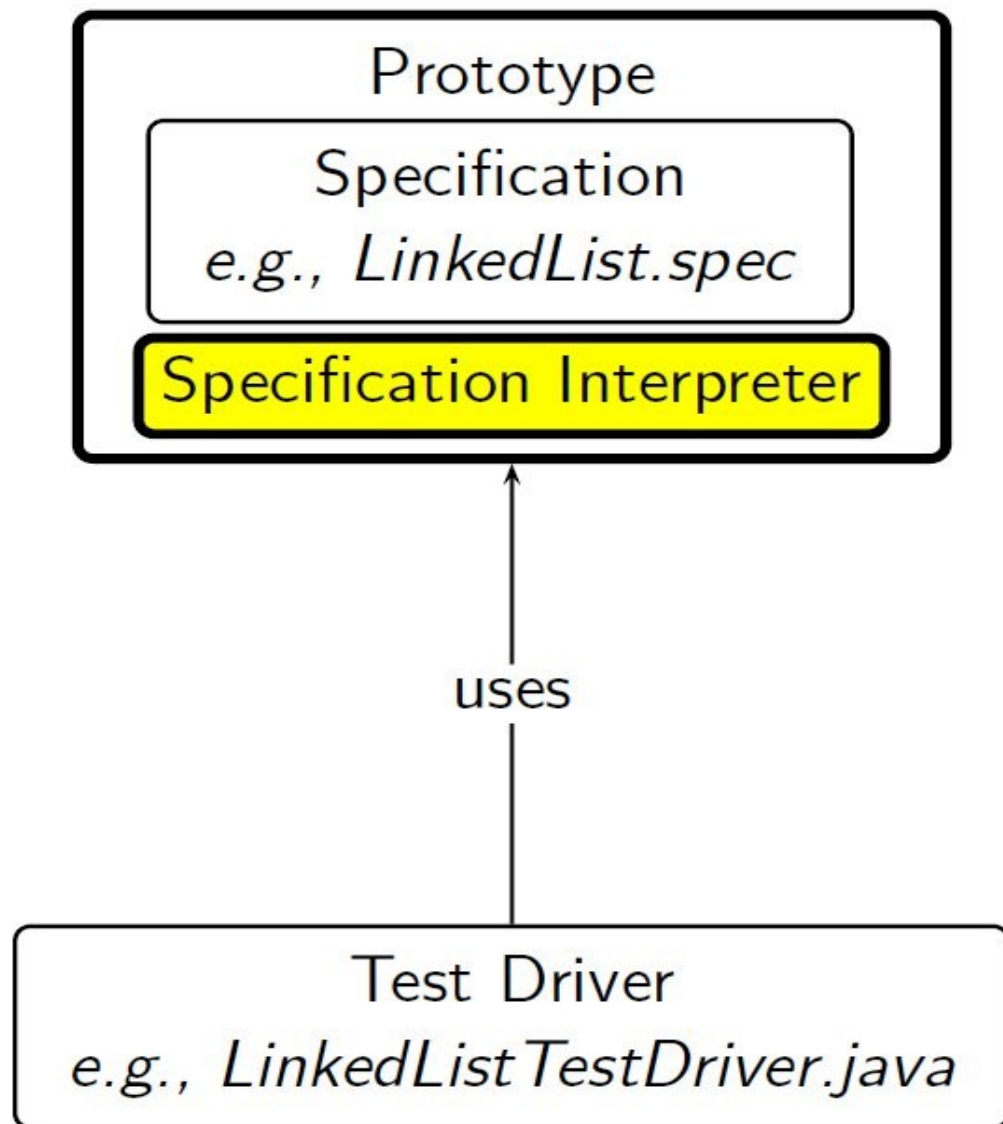
- algebraische Spezifikation SP und Algebra A, mit „gültigen“ Gleichungen.

Algebraische Spezifikation...

- ... wird in speziellen Einsatzgebieten (z.B. Einsatz von Kryptographie in Software) verwendet, wo sich der Aufwand lohnt.
- ... bildet Grundlage für „benutzerfreundlichere“ Ansätze, die allgemein verwendet werden (z.B. Object Constraint Language (OCL) im Rahmen der UML, oder Java / C assertions). [OCL: Teil 3.0 dieser Vorlesung]
- ... bildet Grundlagen für allgemein verwendete Software-Entwicklungs-Prinzipien wie „Design-by-contract“ (vgl. Programmiersprache Eiffel oder Java Markup Language (JML)) oder Software-Verifikationswerkzeuge wie VCC [<http://research.microsoft.com/en-us/projects/vcc>].

Johannes Henkel and Amer Diwan: A Tool for Writing and Debugging Algebraic Specifications, International Conference on Software Engineering (ICSE) 2004 (<http://www-plan.cs.colorado.edu/henkel/pubs.html>)

Zum Beispiel für spezifikations-basiertes Testen.



Beispiel: Von Java zu algebraischer Spezifikation

```
class LinkedList {  
    Object get(int index){...}  
}
```

[HenDiw04]

Algebraic Signature of get?

$LinkedList \times int \rightarrow LinkedList \times Object$



Retrieving the 5th element of LinkedList l :

$get(l,5).state$ — updated state of l after executing $l.get(5)$

$get(l,5).retval$ — return value of $l.get(5)$

Beispiel: Von Java zu algebraischer Spezifikation

[HenDiw04]

original client

algebraic term

```
s = new IntStack();
```

state : *NewIntStack().state*

retval :

```
s.push(5);
```

state : *push(NewIntStack().state, 5).state*

retval :

```
s.push(7);
```

state : *push(push(..., 5).state, 7).state*

retval :

```
i = s.pop();
```

state : *pop(push(..., 7).state).state*

retval : *pop(push(..., 7).state).retval*



ASTOOT [Doong, Frankl 1994]:

<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.23.2625>

Black/White [Chen et al. 1998]:

<http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.541>

TACCLE [Chen, Tse, Chen 2001]:

<http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.2484>

Korat [Boyapati, Kurshid, Marinov 2002]:

<http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.122.9788>

In diesem Kapitel haben wir folgende Themen behandelt:

- Das Spezifikationsproblem
- Vollständigkeit der Algebra
- Signatur und Algebra
- Terme
- Algebraische Spezifikation

Wir haben somit algebraische Spezifikation als Grundlage für Constraint-Spezifikationsansätze wie OCL (Teil 3.0) betrachtet, das insbesondere grundlegende Konzepte für die Spezifikation des Verhaltens einzelner Softwaremodule bereitstellt. Wir haben außerdem kurz einige Beispiele für praktische Anwendungen dieses Ansatz betrachtet.

- Ehrig, Mahr: Fundamentals of Algebraic Specification 1: Equations and Initial Semantics, Springer Verlag, 1985
- Ehrig, Mahr: Fundamentals of Algebraic Specification 2: Module Specifications and Constraints, Springer Verlag, 1990
- Ehrig, Mahr, Cornelius et.al.: Mathematisch-strukturelle Grundlagen der Informatik, Springer Verlag, 1999
- Ehrich, Gogolla, Lipeck: Algebraische Spezifikation abstrakter Datentypen, Teubner Verlag, 1989
- Sommerville, Ian: Software Engineering, Addison-Wesley Longman, 2010. Kapitel 27: Formal Specification.
http://www.cs.st-andrews.ac.uk/~ifs/Books/SE9/WebChapters/PDF/Ch_27_Formal_spec.pdf