Grundlagen der Systementwicklung

Martin Wirsing

in Zusammenarbeit mit
Axel Rauschmayer

WS 05/06

Ziele

- Umbenennung von Spezifikationen und Theoriemorphismen verstehen
- Parametrisierung von Spezifikationen kennen lernen

Umbenennung von Spezifikationen

Zur Definition der Umbenennung von Symbolen einer Spezifikation benötigen wir die Begriffe

- Signaturmorphismus und
- σ-Redukt.

Ein *Signaturmorphismus* ist eine Abbildung zwischen Signaturen, bei der die Funktionalität der abgebildeten Funktionssymbole mit der Abbildung der Sorten verträglich ist.

Definition Signaturmorphismus

1. *Seien* $\Sigma = (S,F)$ *und* $\Sigma' = (S',F')$ *Signaturen. Eine Abbildung* $\sigma = (\sigma_{sort}, \sigma_{op})$ *mit*

   $$\sigma_{sort} : S \to S' \qquad \sigma_{op} : F \to F'$$

   heißt Signaturmorphismus *von* Σ *nach* Σ'*, geschrieben* $\sigma : \Sigma \to \Sigma'$*, wenn für alle* $f \in F_{\langle\langle s_1,\ldots,s_n\rangle,s\rangle}$ *gilt*

   $$\sigma_{op}(f) : \sigma_{sort}(s_1),\ldots,\sigma_{sort}(s_n) \to \sigma_{sort}(s)$$

   das heißt, wenn die Funktionalität von $\sigma_{op}(f)$ *mit der Abbildung der Sorten verträglich ist.*

2. *Sei* $\sigma : \Sigma \to \Sigma'$ *ein* injektiver *Signaturmorphismus,* $A \in \mathrm{Alg}(\Sigma)$*. Die* σ-Übersetzung $\sigma(A)$ *von* A *ist die Algebra mit*

   $$\sigma(A)_{\sigma_{sort}(s)} =_{\text{def}} A_s \qquad \textit{und} \qquad \sigma_{op}(f)^{\sigma(A)} =_{\text{def}} f^A$$

3. *Sei* $\sigma : \Sigma \to \Sigma'$ *Signaturmorphismus,* $B \in \mathrm{Alg}(\Sigma')$*. Das* σ-Redukt $B|_\sigma$ *ist die* Σ*-Algebra mit*

   $$(B|_\sigma)_s =_{\text{def}} B_{\sigma_{sort}(s)} \qquad \textit{und} \qquad f^{B|_\sigma} =_{\text{def}} (\sigma_{op}(f))^B$$

Signaturmorphismus

Beispiel Verträglichkeit von σ_{sort} und σ_{op}.

Ist etwa

- σ_{sort}(Natural) = Int und σ_{op}(succ) = succ und gilt
- succ : Natural -> Natural in der Signatur Σ,

dann muss in der Bildsignatur gelten

- succ : Int -> Int

Signaturmorphismus

Für die Definition der **σ-Übersetzung** ist die **Injektivität** von σ notwendig.

Beispiel

σ : Sig(MONOID) $\cup$ {f : Mon} –> Sig(NAT)

σ_{sort}(Mon) = Nat

$\sigma_{op}(e) = \sigma_{op}(f)$ = zero

$\sigma_{op}(^\circ)$ = +

ein wohldefinierter Signaturmorphismus.

Betrachte den Monoid M mit

$M_{\text{Mon}} = \text{N}$ $\quad e^M = 0$ $\quad f^M = 1$

Dann wäre $\sigma(M)$ nicht wohldefiniert, da sowohl

$\text{zero}^{\sigma(M)} = (\sigma_{op}(e))^{\sigma(M)} = e^M = 0$

als auch

$\text{zero}^{\sigma(M)} = (\sigma_{op}(f))^{\sigma(M)} = f^M = 1$

gelten müsste.

Signaturmorphismus

Dagegen ist die Reduktbildung für beliebige Signaturmorphismen möglich:

Das Redukt des Standardmodells $\mathcal{N}$ der natürlichen Zahlen bzgl. σ

ist die folgende Monoid-Algebra

$$
\begin{array}{llll}
Mon^{\mathcal{N}|\sigma} & = \sigma_{sort}(Mon)_{\mathcal{N}} = Nat_{\mathcal{N}} & & = N \\
e^{\mathcal{N}|\sigma} & = \sigma_{op}(e)^{\mathcal{N}} & = zero^{\mathcal{N}} & = 0 \\
f^{\mathcal{N}|\sigma} & = \sigma_{op}(f)^{\mathcal{N}} & = zero^{\mathcal{N}} & = 0 \\
\circ^{\mathcal{N}|\sigma} & = \sigma_{op}(\circ)^{\mathcal{N}} & = +^{\mathcal{N}} & = +
\end{array}
$$

bei der e und f beide durch 0 interpretiert werden.

Signaturmorphismus

Seien Signaturen Σ und Σ' gegeben.
Eine endliche Abbildung σ (zwischen Zeichen) der Form

sort s_1 to q1 .
…
sort s_k to q_k .
op f_1 to g_1 .
…
op f_1 to g_1 .

wobei

Sorten von Σ auf Sorten von Σ' und
Funktionszeichen von Σ auf Funktionszeichen von Σ'
(ohne Angabe der Funktionalität)

so abgebildet werden, dass sie verträglich mit der Abbildung der Sorten sind,

ist ein Signaturmorphismus $\sigma : \Sigma \text{ -> } \Sigma'$.

Signaturmorphismus

Bemerkung:

- Sorten oder Funktionszeichen von Σ, die in der Zeichenabbildung nicht erwähnt werden, werden identisch nach Σ' abgebildet.
- Die Umbenennung von Definitions- und Wertebereich der Funktionszeichen ergibt sich aus der Umbenennung der Sorten.
- Ist nur Σ gegeben und σ injektiv, so ist σ ein Signaturmorphismus von Σ nach $\sigma(\Sigma)$.

Theoriemorphismus

Ein Theoriemorphismus α: SP1 $\rightarrow$ SP2 erhält die Theorie von SP1; d.h.

SP2 erfüllt alle Axiome von SP1 (modulo Umbenennung).

Definition (Theorie Morphismus):

Ein **Theoriemorphismus** α: SP1 $\rightarrow$ SP2 ist ein

Signaturmorphismus α: sig(SP1) $\rightarrow$ sig(SP2),

so dass für jedes Modell M $\in$ Mod(SP2) gilt:

$M|_\alpha \in$ Mod(SP1)

Sicht (View)

Jeder Theoriemorphismus α: SP1 $\rightarrow$ SP2 induziert eine Sicht von SP1 nach SP2:

Definition (Sicht in Maude)

Sei α: SP1 $\rightarrow$ SP2 ein **Theoriemorphismus**.

Dann ist

```
view SM from SP1 to SP2 is α endv
```

eine **Sicht** in Maude.

Sicht (View)

Beispiel: Eine Sicht von TRIV nach NAT0

```
view NATVIEW from TRIV to NAT0 is
    sort Elt to Natural .
endv
```

wobei

```
fth TRIV is sort Elt . endfth
fmod NAT0 is
   sort Natural . op 0 : -> Natural [ctor] .
   op s : Natural -> Natural [ctor] .
   op _+_ : Natural Natural -> Natural .
   vars N M : Natural .
   eq N + 0 = N .  eq N + s(M) = s(N + M) .
 endfm
```

Sicht von geordneter Struktur auf Listen

Beispiel: Eine Sicht von TAO-SET nach SEQ

```
view TAO2SEQ from TAO-SET to SEQ is
    sort Elt to SEQ .
endv
```

Zu zeigen: SEQ erfüllt die Eigenschaften von TAO-SET;
$Mod(SEQ|_{TAOSEQ}) \subset Mod(TAO\text{-}SET)$ bzw. äquivalent
$SEQ \models trans \wedge antisymmetric$

```
fth TAO-SET is
  protecting BOOL .
  including TRIV .
  op _<_ : Elt Elt -> Bool .
  vars X Y Z : Elt .
  ceq X < Z = true if X < Y and Y < Z [nonexec label trans] .
  ceq X = Y if X < Y /\ Y < X [nonexec label antisymmetric] .
endfth
```

"<" transitiv & antisymmetrisch: Kann instantiiert werden mit totaler Ordnung, partieller Ordnung, "kleiner-gleich" und "kleiner"

Sicht von geordneter Struktur auf Listen

```
fmod SEQ is
   protecting BOOL .
   including NATURAL .
   sort Seq .
   subsort Natural < Seq .

   op empty : -> Seq [ctor] .
   op _;_ : Seq Seq -> Seq [ctor assoc id: empty prec 33] .

   vars N M : Natural .
   vars L L1 : Seq .

   op _<_ : Seq Seq -> Bool [prec 37] .
   eq empty < (N ; L) = true .
   eq L < empty = false .
   eq (N ; L) < (M ; L1) = L < L1 .
endfm
```

`L < L1` gdw.
Länge von L < Länge von L1

Sicht von geordneter Struktur auf Listen

Schöner wäre „<length" statt „<„ :

```
fmod SEQ1 is
     . . .
  op _<length_ : List List -> Bool .
  vars N M : Natural . vars L L1 : List .
  eq empty <length (N ; L) = true .
  eq L <length empty = false .
  eq (N ; L) <length (M ; L1) = L <length L1
  .
endfm

view TAO2SEQ1 from TAO-SET to SEQ1 is
```


Parametrisierte Spezifikationen

Eine **parametrisierte Spezifikation** (oder generische Spezifikation) hat die Form

```
fmod SN{SP1, …, SPk} is
     Body
endfm
```

wobei

SN der Name der parametrisierten Spezifikation,

SP1, … , SPk die Namen der formalen Parameter**theorien** und

Body der Rumpf der Spezifikation ist.

SN ist wohldefiniert, wenn der Rumpf die formalen Parameter erweitert, d.h. wenn

including SP1 . … including SPk . Body

wohldefiniert ist .

Parametrisierte Listen

```
fmod LIST{X :: TRIV} is
  protecting NAT .
  sorts NeList{X} List{X} .
  subsort X$Elt < NeList{X} < List{X} .

  op nil : -> List{X} [ctor] .
  op __ : List{X} List{X} -> List{X}
                    [ctor assoc id: nil prec 25] .
  op __ : NeList{X} List{X} -> NeList{X}    [ctor ditto] .
  op __ : List{X} NeList{X} -> NeList{X}    [ctor ditto] .
  . . .

  var E E' : X$Elt .
  vars A L : List{X} .
  var C : Nat .
```

Parametrisierte Sorten

Qualifizierte Sorte des formalen Parameters

Parametrisierte Listen

```
  op append : List{X} List{X} -> List{X} .
  op append : NeList{X} List{X} -> NeList{X} .
  op append : List{X} NeList{X} -> NeList{X} .
  eq append(A, L) = A L .

  op head : NeList{X} -> X$Elt .
  eq head(E L) = E .

  op tail : NeList{X} -> List{X} .
  eq tail(E L) = L .
  ...
```

Parametrisierte Listen

```
  op reverse : List{X} -> List{X} .
  op reverse : NeList{X} -> NeList{X} .
  eq reverse(L) = $reverse(L, nil) .

  op $reverse : List{X} List{X} -> List{X} .
  eq $reverse(nil, A) = A .
  eq $reverse(E L, A) = $reverse(L, E A) .

  op size : List{X} -> Nat .
  op size : NeList{X} -> NzNat .
  eq size(L) = $size(L, 0) .

  op $size : List{X} Nat -> Nat .
  eq $size(nil, C) = C .
  eq $size(E L, C) = $size(L, C + 1) .
endfm
```

Repetitive Rekursion (Tail Recursion)

Parameterübergabe

Zur Parameterübergabe muss der formale Parameter mit dem aktuellen Parameter in Beziehung gebracht werden:

- Die Signatur des formalen Parameters muss in die Signatur des aktuellen Parameters umbenannt werden und
- der aktuelle Parameter muss die Anforderungen des formalen Parameters erfüllen,

d.h. es muss einen Theoriemorphismus von dem formalen Parameter auf den aktuellen Parameter geben.

Parameterübergabe

Beispiel: Instantiierung von `LIST{X :: TRIV}` mit der Spezifikation der natürlichen Zahlen `NATURAL`.

Die Sorte `Elt` muss in `Natural` umbenannt werden, d.h. der formale Parameter von LIST muss eine Sicht auf `Natural` besitzen.

```
view TRIV2NAT from TRIV to NATURAL is
     sort Elt to Natural .
endv
```


Parameterübergabe

```
fmod LISTNATURAL is
   including LIST{TRIV2NAT} .
   op count : List{TRIV2NAT} Natural -> Natural .

   vars E M : Natural .
   var L : List{TRIV2NAT} .

   eq count(nil, M) = 0 .
   eq count(E L, M) =
       if E == M then count(L, M) + s 0
                 else count(L, M)
       fi .
endfm
red in LISTNATURAL : count(s s 0  s 0  s s 0  s 0, s 0) .
```

Sortierbare Listen

```
fth TAO-SET is
  protecting BOOL .
  including TRIV .
  op _<_ : Elt Elt -> Bool .
  vars X Y Z : Elt .
  ceq X < Z = true if X < Y and Y < Z [nonexec label trans] .
  ceq X = Y if X < Y /\ Y < X [nonexec label antisymmetric] .
endfth
```

“<“ transitiv & antisymmetrisch: Kann instantiiert werden mit totaler Ordnung, partieller Ordnung, “kleiner-gleich“ und “kleiner”

```
fmod SORTABLE-LIST{X :: TAO-SET} is
  protecting NAT .
  sorts NeList{X} List{X} .
  subsort X$Elt < NeList{X} < List{X} .
   . . .
  vars E E' : X$Elt .
  vars A A' L L' : List{X} .
  var N : NeList{X} .
```


Sortierbare Listen

```
sort $Split{X} .

op sort : List{X} -> List{X} .
op sort : NeList{X} -> NeList{X} .
eq sort(nil) = nil .
eq sort(E) = E .
eq sort(E N) = $sort($split(E N, nil, nil)) .

op $sort : $Split{X} -> List{X} .
eq $sort($split(nil, L, L')) =
     $merge(sort(L), sort(L'), nil) .

op $split : List{X} List{X} List{X} -> $Split{X} [ctor] .
eq $split(E, A, A') = $split(nil, A E, A') .
eq $split(E L E', A, A') = $split(L, A E, E' A') .
```

Hilfssorte mit Normalformen
`$split(nil, L, L1)`
wobei
|length(L) – length(L1)| <= 1

Sortierbare Listen

```
op merge : List{X} List{X} -> List{X} .
op merge : NeList{X} List{X} -> NeList{X} .
op merge : List{X} NeList{X} -> NeList{X} .
eq merge(L, L') = $merge(L, L', nil) .

op $merge : List{X} List{X} List{X} -> List{X} .
eq $merge(L, nil, A) = A L .
eq $merge(nil, L, A) = A L .
eq $merge(E L, E' L', A) =
    if E < E' == true
    then $merge(L, E' L', A E)
        else $merge(E L, L', A E')
    fi .
endfm
```

merge(L, L') mischt zwei sortierte Listen

Parameterübergabe

Instantiierung mit SEQ:

```
view TAO2SEQ from TAO-SET to SEQ is
   sort Elt to Seq .
endv

fmod LIST-SORT-SEQ is
   protecting SORTABLE-LIST{TAO2SEQ} .
endfm

red in LIST-SORT-SEQ :
sort(
(0 ; s 0 ; s s 0 ; s s s 0)  (s 0 ; 0)  (s s 0) (0 ; s 0)
     ) .
```

Parametrisierte endliche Mengen

```
fmod SET{X :: TRIV} is
protecting NAT .
sorts NeSet{X} Set{X} .
subsort X$Elt < NeSet{X} < Set{X} .

op empty : -> Set{X} [ctor] .
op _,_ : Set{X} Set{X} -> Set{X}
         [ctor assoc comm id: empty prec 121] .
op _,_ : NeSet{X} Set{X} -> NeSet{X}   [ctor ditto] .

var E : X$Elt .
var N : NeSet{X} .
vars A S S' : Set{X} .
var C : Nat .

eq N, N = N .
```


Parametrisierte endliche Mengen

```
op insert : X$Elt Set{X} -> Set{X} .
eq insert(E, S) = E, S .

op delete : X$Elt Set{X} -> Set{X} .
eq delete(E, (E, S)) = delete(E, S) .
eq delete(E, S) = S [owise] .

op _in_ : X$Elt Set{X} -> Bool .
eq E in (E, S) = true .
eq E in S = false [owise] .
```

Parametrisierte endliche Mengen

```
  op |_| : Set{X} -> Nat .
  op |_| : NeSet{X} -> NzNat .
  eq | S | = $card(S, 0) .

  op $card : Set{X} Nat -> Nat .
  eq $card(empty, C) = C .
  eq $card((N, N, S), C) = $card((N, S), C) .
  eq $card((E, S), C) = $card(S, C + 1) [owise] .

  op union : Set{X} Set{X} -> Set{X} .
  op union : NeSet{X} Set{X} -> NeSet{X} .
  op union : Set{X} NeSet{X} -> NeSet{X} .
  eq union(S, S') = S, S' .
  . . .
endfm
```


Parametrisierte endliche Abbildungen

```
fmod MAP{X :: TRIV, Y :: TRIV} is
  sorts Entry{X,Y} Map{X,Y} .
  subsort Entry{X,Y} < Map{X,Y} .

  op _|->_ : X$Elt Y$Elt -> Entry{X,Y} [ctor] .
  op empty : -> Map{X,Y} [ctor] .
  op _,_ : Map{X,Y} Map{X,Y} -> Map{X,Y}
       [ctor assoc comm id: empty prec 121] .
  op undefined : -> [Y$Elt] [ctor] .
```

“Kind” (Art) der Sorte s: Erweiterung der Sorte s um “Fehlerterme” der Art [s]

Parametrisierte endliche Abbildungen

```
  var D : X$Elt .
  vars R R' : Y$Elt .
  var M : Map{X,Y} .

  op insert : X$Elt Y$Elt Map{X,Y} -> Map{X,Y} .
  eq insert(D, R, (M, D |-> R')) = (M, D |-> R) .
  eq insert(D, R, M) = (M, D |-> R) [owise].

  op _[_] : Map{X,Y} X$Elt -> [Y$Elt] .
  eq (M, D |-> R)[D] = R .
  eq M[D] = undefined [owise] .
endfm
```


Zusammenfassung

- Maude unterstützt Strukturierung von Spezifikationen durch Umbenennung und Parametrisierung.
- Eine parametrisierte Spezifikation hat Theorien als formale Parameter.
- Ein aktueller Parameter SPA muss (modulo Umbenennung) die Signatur des formalen Parameters T enthalten und alle Eigenschaften von T erfüllen, d.h. es muss einen Theoriemorphismus von T nach SPA geben.