

## 12. Interpretation von PVS-Theorien

- Vorspann zu allgemeinen Begriffen: Konsistenz usw.
- Äquivalenz und Quotienten
- Theorie-Interpretation

# Vorspann: Interpretation, Modell, Konsistenz von Theorien

Grundlegende Begriffe, die unabhängig von speziellen Logiken benutzt werden

Die *Semantik* einer logischen Sprache (z.B. Prädikatenlogik erster Stufe, Prädikatenlogik höherer Stufe usw.) wird durch Angabe einer *Interpretation* der nichtlogischen Symbole festgelegt, üblicherweise mit mengentheoretischen Konstrukten:

- Angabe eines 'Universums' (einer Grundmenge), oder im Fall einer getypten Sprache Angabe einer Trägermenge für jeden Typ;
- Angabe von geeigneten (typkorrekten) Konstanten, Funktionen usw. über den entsprechenden Trägermengen für jedes Konstanten-, Funktions- und Prädikatensymbol.

Bezüglich einer Interpretation kann ausgesagt werden, ob eine Formel unter der Interpretation wahr ist oder nicht, oder ob eine Interpretation eine Formel erfüllt.

# Interpretation, Modell, Konsistenz von Theorien (2)

Für einzelne Formeln gibt es die Begriffe:

- Eine Formel heißt *erfüllbar*, wenn es eine Interpretation gibt, die sie erfüllt.
- Eine Formel heißt *gültig* oder *Tautologie*, wenn sie unter *jeder* Interpretation wahr ist.
- Eine Formel heißt *unerfüllbar* oder *widersprüchlich*, wenn es *keine* Interpretation gibt, unter der sie erfüllt ist.

Die Begriffe werden auf Formelmengen analog übertragen:

- Eine Formelmenge heißt *erfüllbar* oder *konsistent*, wenn es eine Interpretation gibt, die alle Formeln aus der Menge erfüllt.
- Eine Formelmenge heißt *unerfüllbar*, *widersprüchlich* oder *inkonsistent*, wenn es *keine* Interpretation gibt, unter der alle Formeln aus der Menge erfüllt sind.

# Interpretation, Modell, Konsistenz von Theorien (3)

Eine Interpretation, die eine Formelmenge erfüllt, wird auch als *Modell* der Formelmenge bezeichnet.

Eine *Theorie* definiert eine Formelmenge auf folgende Weisen:

- die Menge der Formeln, die in jedem Modell der Axiome wahr sind  
“semantischer Folgerungsbegriff”
- die Menge der Formeln, die sich mit Hilfe der (logischen) Schlußregeln aus den Axiomen ableiten lassen  
“(syntaktischer) Ableitungsbegriff”

Im günstigsten Fall sind beide Mengen identisch

↪ Vollständigkeit der Theorie

Im PVS-Kontext steht eine Theorie pragmatisch für die Menge der beweisbaren Formeln.

# Interpretation, Modell, Konsistenz von Theorien (4)

Die Standardtechnik zur Demonstration, dass eine Theorie (d.h. die Menge ihrer Axiome) konsistent ist, ist die Angabe eines Modells für die Theorie.

Eine Interpretation kann auch relativ zu einer anderen Theorie angegeben werden (abstrakte Interpretation).

In diesem Fall muss gezeigt werden, dass die Axiome der interpretierten Theorie Theoreme in der Theorie sind, aus der die Interpretation definiert wird.

~> Konsistenz der ersten Theorie ist relativ zur Konsistenz der zweiten.

# Theorie-Interpretation in PVS

(Weitere Details hierzu finden sich im technischen Bericht *Theory Interpretations in PVS* von SRI)

Syntax:

- Angabe eines Theorienamens, gefolgt von
- einer Liste von Zuordnungen für die Namen (Typen und Konstanten) der Theorie, eingeschlossen in “{. . . }”

Semantik: die Axiome der Theorie müssen als Theoreme der 'Gast-Theorie' nachgewiesen werden

~> relative Konsistenz

# Beispiel: Gruppentheorie

```
group: THEORY
BEGIN
  G: TYPE+
  +: [G, G -> G]
  0: G
  -: [G -> G]
  x, y, z: VAR G
  associative_ax: AXIOM FORALL x, y, z: x + (y + z) = (x + y) + z
  identity_ax: AXIOM FORALL x: x + 0 = x
  inverse_ax: AXIOM FORALL x: x + -x = 0 AND -x + x = 0
  idempotent_is_identity: LEMMA x + x = x => x = 0
END group
```

Theorie-Interpretationen – hier Angabe von 'konkreten' Gruppen:

`group`{`{G := int, + := +, 0 := 0, - := -}`} Integers mit Addition

`group`{`{G := nzreal, + := *, 0 := 1, - := LAMBDA (r:nzreal): 1/r}`}

the multiplicative Gruppe der reellen Zahlen ungleich 0.

# Beispiel: Realisierung von abstrakten Stacks

- als komplexeres Beispiel für Theorie-Interpretation (Standard-Problem im Kontext der formal fundierten Software-Entwicklung auf der Grundlage von algebraischen Spezifikationen)
- als Beispiel für das Zusammenwirken der verschiedenen PVS-Konstrukte bei der Formalisierung komplexerer mathematischer Sachverhalte

Theorie stack: abstrakte Stack-Struktur (ADT)

```
stack[t:TYPE] : DATATYPE
  BEGIN
    empty: empty?
    push(top:t, pop: stack): nonempty?
  END stack
```

## Stack-Beispiel (2)

Interpretation von Stacks: Realisierung der Stack-Struktur durch ein Array zusammen mit einem Zeiger auf das Stack-Ende ('top')

```
cstack[t: TYPE+]: THEORY
  BEGIN
    cstack: TYPE = [# size: nat, elems: [nat > t] #]
    cempty?(s: cstack): bool = (s'size = 0)
    cempty: (cempty?) =
      (# size := 0,
        elems := LAMBDA (n: nat): epsilon(LAMBDA (x:t): true) #)
    cnonempty?(s: cstack): bool = (s'size /= 0)
    ctop(s: (cnonempty?): t = s'elems(s'size 1)
    cpop(s: (cnonempty?): cstack = s WITH ['size := s'size 1]
    cpush(x: t)(s: cstack): (cnonempty?) =
      (# size := s'size + 1,
        elems := s'elems WITH [(s'size) := x] #)
    ce(s1, s2: cstack): bool =
      s1'size = s2'size AND
      FORALL (n: below(s1'size)): s1'elems(n) = s2'elems(n)
```

## Stack-Beispiel (3)

...

```
IMPORTING equivalence_class[cstack, ce], lifteq, lifteqs
```

...

```
equivalence_class[T:TYPE, ==: (equivalence?[T])] : THEORY
BEGIN
  x, y: VAR T
  equiv_class(x): setof[T] = {y | x == y}
  E: TYPE = {A: setof[T] | EXISTS x: A = equiv_class(x)}
  rep(A: E): (A) = epsilon(A)
  CONVERSION equiv_class, rep

  equiv_class_covers: LEMMA FORALL x: EXISTS (A: E): member(x, A)
  equiv_class_separates: LEMMA
    NOT (x == y)
    IMPLIES disjoint?(equiv_class(x), equiv_class(y))
END equivalence_class
```

## Stack-Beispiel (4)

Auswahl-Operator: `epsilon` wählt aus einer (nichtleeren) Menge ein beliebiges Element aus.

```
epsilon [T: NONEMPTY_TYPE]: THEORY
BEGIN
  p: VAR pred[T]
  x: VAR T

  epsilon(p): T

  epsilon_ax: AXIOM (EXISTS x: p(x)) => p(epsilon(p))

END epsilon
```

## Stack-Beispiel (5)

```
lifteq[D, R: TYPE, deq: (equivalence?[D])]: THEORY
```

```
  BEGIN
```

```
    IMPORTING equivalence_class
```

```
    lift(f:(preserves[D, R](deq, =[R]))) (A:E[D,deq]) : R  
      = f(rep(A))
```

```
    CONVERSION lift
```

```
  END lifteq
```

```
lifteqs[D, R: TYPE, deq: (equivalence?[D]), req: (equivalence?[R])]: THEORY
```

```
  BEGIN
```

```
    IMPORTING equivalence_class
```

```
    lift(f:(preserves[D, R](deq, req))) (A:E[D,deq]) : E[R,req]  
      = equiv_class[R,req](f(rep(A)))
```

```
    CONVERSION lift
```

```
  END lifteqs
```

## Stack-Beispiel (6)

Aus dem Prelude:

```
functions [D, R: TYPE]: THEORY
BEGIN
  f, g: VAR [D --> R]
  x, x1, x2: VAR D
  y: VAR R
  Drel: VAR PRED[[D, D]]
  Rrel: VAR PRED[[R, R]]
  ...
  preserves(f, Drel, Rrel): bool =
    FORALL x1, x2: Drel(x1, x2) IMPLIES Rrel(f(x1), f(x2))

  % Curried form
  preserves(Drel, Rrel)(f): bool = preserves(f, Drel, Rrel)
  ...
END functions
```

# Stack-Beispiel (6)

Fortsetzung Theorie cstack

```
estack: TYPE = E
IMPORTING stack[t]
  {{ stack      := estack,
    empty?     := cempty?,
    nonempty?  := cnonempty?,
    empty      := cempty,
%%    push(x: t, s: estack) := cpush(x)(s),
    push := LAMBDA (x: t, A: E[cstack, ce]):
              equiv_class[cstack, ce] (cpush(x)(rep(A))),
%%    top       := ctop,
    top := LAMBDA (A: E[cstack, ce] | cnonempty?(rep(A))): ctop(rep(A)),
%%    pop       := cpop
    pop := LAMBDA (A: E[cstack, ce] | cnonempty?(rep(A))):
          equiv_class[cstack, ce] (cpop(rep(A)))
  }}
END cstack
```