

Binäre Bäume

Binärbäume als weiteres Beispiel für abstrakte Datentypen in PVS mit in Knoten gespeicherten Werten vom Typ T:

```
BinTree [T: TYPE]: DATATYPE
  BEGIN
    empty: empty?
    node (key: T, left:BinTree, right:BinTree): node?
  END BinTree
```

Definition ist völlig analog zu der für lineare Listen.

Unterschiede:

- 2 anstatt einer induktiven Komponente beim Aufbau der Struktur.
- Entsprechend gibt es bei der Definition rekursiver Funktionen über Binärbäumen 2 Rekursionsargumente – für jedes muss gezeigt werden, dass das Argument beim rekursiven Aufruf 'kleiner' ist.

Parametrisierung von Abstrakten Datentypen

Eine DATATYPE-Definition kann als separates Theorie-Äquivalent auf der obersten Ebene (d.h. *nicht* innerhalb einer Theorie) gegeben werden oder innerhalb einer Theorie.

Eine separate DATATYPE-Definition kann genauso wie eine Theorie parametrisiert sein.

Eine DATATYPE-Definition *innerhalb* einer Theorie ist “automatisch” parametrisiert mit den Parametern der Theorie, in die sie eingebettet ist.

Instanziierung von ADTs: wie Instanziierung von Theorien – s. später

9. Mehr zu PVS-Theorien

- Mehr zu parametrisierten Theorien
- Theorie-Abhängigkeiten: Import und Export
Sichtbarkeit von Namen
- Einschränkungen auf Theorie-Parametern: Theorie-Assumptions

Mehr zu Parametrisierten Theorien

Zwischen Theorie-Parametern können Abhängigkeiten bestehen, wie bei abhängigen Typen: d.h. der Wert eines Parameters (repräsentiert durch den Parameter-Namen) kann im Typ eines nachfolgenden Parameters benutzt werden.

Beispiel:

```
sort [T: TYPE, <=: pred[[T,T]] ]: THEORIE
BEGIN
  ...
END sort
```

Theorie-Assumptions

Theorie-Parameter können mit Einschränkungen (engl. *constraints*) belegt werden.

- durch *Assumptions*
- durch entsprechende Typisierung

Syntax:

```
name [<parameter>]: THEORY
BEGIN
  ASSUMING % Assumption Formeln
    n1: ASSUMPTION <formel>
    ...
  ENDASSUMING
  ...
END name
```

Theorie-Assumptions (2)

```
sort [T: TYPE, <=: pred[T,T]] : THEORIE
BEGIN
  ASSUMING
    po_rel: ASSUMPTION partial_order?(<=)
  ENDASSUMING
  ...
END sort
```

Alternativ:

```
sort [T: TYPE, <=: (partial_order?[T])] : THEORIE
BEGIN
  ...
END sort
```

Theorie-Assumptions (3)

Assumptions haben eine duale Funktion:

- Innerhalb der Theorie wirken sie wie Axiome, d.h. als Eigenschaften, die angenommen werden können.
- Wenn eine Theorie verwendet und damit instantiiert wird, müssen entsprechende Instanzen der Assumptions nachgewiesen werden
 \rightsquigarrow Beweisverpflichtungen, TCCs

Theorie-Abhängigkeiten: Import und Export

Eine Kollektion von Theorien ist zunächst flach, d.h. sie enthält keine Struktur. Im PVS-System sind alle Theorien (in einem Verzeichnis) sichtbar und auf der gleichen Ebene angesiedelt.

Die in einer Theorie deklarierten Namen sind zunächst nur innerhalb der Theorie sichtbar.

Ausnahme: Namen, die im `prelude` deklariert wurden, sind überall sichtbar.

Sichtbarmachen von Namen aus benutzer-definierten Theorien und damit eine gewisse hierarchische Abhängigkeitsstruktur zwischen Theorien kann durch *Importieren* und *Exportieren* von Theorien erreicht werden.

Eine Importing-Klausel,

```
IMPORTING <theorie-instanz>
```

kann im Text einer Theorie überall dort auftreten, wo Deklarationen auftreten können, einschließlich der Parameter-Liste und einer ASSUMING-Klausel.

Theorie-Hierarchien: Import und Export (2)

Durch Importierung werden alle in der importierten Theorie deklarierten Namen (mit Ausnahme der Variablen) in der importierten Theorie vom Platz der `IMPORTING`-Klausel an sichtbar; Einschränkungen hierbei sind durch Angabe einer expliziten `EXPORTING`-Klausel (s.u.) möglich.

Eine parametrisierte Theorie kann nur in instantiiertem Form, d.h. mit Angabe von aktuellen Parametern, importiert werden.

Der Name einer Theorie-Instanz besteht aus dem eigentlichen Theorie-Namen gefolgt von der Liste der aktuellen Parameter in [...], z.B.

```
sort [nat, <=]
```

Häufig werden hierbei die formalen Parameter der importierenden Theorie benutzt.

Die genaue Position der Klausel ist hier relevant: es kann erforderlich sein, eine `IMPORTING`-Klausel in der Parameterliste einer Theorie einzuschieben, damit die richtige zu importierende Instanz spezifiziert werden kann.

Theorie-Hierarchien: Import und Export (3)

Eine `EXPORTING`-Klausel erlaubt die Steuerung der Sichtbarkeit von Namen aus der Theorie.

Zusammen mit den in der Theorie deklarierten Namen können auch die importierten Theorie-Instanzen exportiert werden.

Wenn Sichtbarkeit nicht eingeschränkt werden soll, ist eine `EXPORTING`-Klausel nicht notwendig:

Wenn eine Theorie *keine* `EXPORTING`-Klausel enthält, bedeutet es, daß *alle* Namen bei Importierung der Theorie (bzw. einer Instanz) sichtbar gemacht werden; ebenso werden importierte Theorie-Instanzen defaultmäßig re-exportiert.

Eine explizite `EXPORTING`-Klausel ist daher nur selten wirklich notwendig.

Theorie-Hierarchien: Import und Export (4)

Qualifizierung von importierten Namen:

Zur Auflösung von Mehrdeutigkeiten kann es notwendig sein, Namen mit dem Namen der Theorie bzw. der Theorie-Instanz, aus der sie stammen, zu qualifizieren.

Beispiel:

```
the1 [p1: ..., pn:...] THEORY
...
id1: sometype
...
END the1

the2 ... : THEORY
  IMPORTING the1[a1,...,an]
... the1[a1,...,an].id1 ...
... id1[a1,...,an] ...
... id1 ...
```

Die 3 Arten der Notation für den Namen `id1` aus Theorie `the1` sind möglich, soweit die Typüberprüfung die Zuordnung eindeutig auflösen kann.

Beispiele

Die folgenden Beispiele enthalten Definitionen, die zum großen Teil auch im prelude enthalten sind, sie dienen zur Demonstration von Parametrisierung usw.

```
partial_order [t: TYPE] : THEORY
  BEGIN
    <= : PRED[[t,t]]          x, y, z: VAR t
    refl:  AXIOM x <= x
    trans: AXIOM x <= y AND y <= z IMPLIES x <= z
    antisym: AXIOM x <= y AND y <= x IMPLIES x = y;
    < : PRED[[t,t]] = (LAMBDA x, y: x <= y AND x /= y)
  END partial_order
```

```
total_order [t: TYPE] : THEORY
  EXPORTING ALL WITH partial_order[t]
  BEGIN
    IMPORTING partial_order[t]
    x, y: VAR t
    total: AXIOM x <= y OR y <= x
  END total_order
```

Beispiele (2)

```
sort [domain, range: TYPE] : THEORY
  BEGIN
    IMPORTING partial_order[range], total_order[domain]
    Array_type: TYPE = ARRAY[domain -> range]
    A, B, C: VAR Array_type
    sorted?(A): bool =
      (FORALL (x, y: domain): x < y IMPLIES NOT (A(y) <= A(x)))
  END sort
```

Beispiele (3)

```
sorta [domain, range: TYPE,  
      d_order: PRED[[domain, domain]],  
      r_order: PRED[[range, range]]] : THEORY  
BEGIN  
  ASSUMING  
    x, y, z: VAR domain  
    u, v, w: VAR range  
    d_refl:  ASSUMPTION d_order(x,x)  
    d_trans: ASSUMPTION d_order(x, y) & d_order(y, z) => d_order(x, z)  
    d_antisym: ASSUMPTION d_order(x, y) & d_order(y, x) => x = y  
    d_total:  ASSUMPTION d_order(x, y) OR d_order(y, x)  
    ... % Assumptione fuer r_order  
  ENDASSUMING  
  Array_type: TYPE = ARRAY[domain -> range]  
  A, B, C: VAR Array_type  
  sorted?(A): bool = (FORALL (x, y: domain):  
    (d_order(x, y) & x /= y) => NOT r_order(A(y), A(x)))  
END sorta
```

Beispiele (4)

```
partial_order1 [t: TYPE, <=: PRED[[t,t]]] : THEORY
BEGIN
  ASSUMING
    x, y, z: VAR t
    refl: ASSUMPTION x <= x
    trans: ASSUMPTION x <= y AND y <= z IMPLIES x <= z
    antisym: ASSUMPTION x <= y AND y <= x IMPLIES x = y
  ENDASSUMING
  < : PRED[[t,t]] = (LAMBDA x, y: x <= y AND x /= y)
END partial_order1
```

Beispiele (5)

```
orderings[t: TYPE] : theory
BEGIN
  x, y, z: VAR t
  pp, qq: VAR PRED[t]
  <= : VAR PRED[[t,t]]
  reflexive?(<=): bool = (FORALL x: x <= x)
  antisymmetric?(<=): bool = (FORALL x, y: x <= y AND y <= x IMPLIES x = y)
  transitive?(<=): bool =
    (FORALL x, y, z: x <= y AND y <= z IMPLIES x <= z)
  partial_order?(<=): bool =
    reflexive?(<=) AND antisymmetric?(<=) AND transitive?(<=)
  linear?(<=): bool = (FORALL x, y: x <= y OR y <= x)
  total_order?(<=): bool = partial_order?(<=) AND linear?(<=)
  well_founded?(<=): bool =
    (FORALL qq: (EXISTS y: qq(y))
      IMPLIES (EXISTS y: (FORALL x: qq(x) IMPLIES NOT x<=y)))
END orderings
```

Beispiele (6)

```
sorto [domain, range: type,  
      (IMPORTING orderings)  
      d_order: (total_order?[domain]),  
      r_order: (partial_order?[range])] : THEORY  
BEGIN  
  Array_type: TYPE = ARRAY[domain -> range]  
  A, B, C: VAR Array_type  
  sorted?(A): bool =  
    (FORALL (x, y: domain): (d_order(x, y) AND x/=y)  
      IMPLIES NOT r_order(A(y), A(x)))  
END sorto
```