

# Kurzeinführung in den Umgang mit PVS

# Das PVS-System

- PVS läuft als *inferior Lisp*-Prozess in einer Emacs-Oberfläche.
- Online-Tutorial zu Emacs mit `C-h t`.  
Online-Dokumentation zu Emacs mit `C-h i`.  
Online-Übersicht der PVS-Kommandos mit `C-c h`.
- Vor dem ersten Starten: Anlegen eines eigenen Verzeichnisses, in dem PVS-Dateien aufbewahrt werden.
- Starten von PVS im Rechner-Pool mit:  

```
hp2@ws107:~/PVS > ~hp2/PVS2.4/pvs
```

  
in dem entsprechenden Verzeichnis. Auf Frage nach Anlegen eines neuen Kontextes (bei erstmaliger Verwendung) mit `yes` antworten.
- Verlassen von PVS mit `C-x C-c`.

# Umgang mit PVS

Die Arbeit mit PVS läuft grob in drei Phasen ab:

## 1. **Spezifizieren:**

Deklaration von Namen für Typen und Variablen; Definition von Typen, Konstanten, Funktionen; Deklaration von Formeln (Axiome, Lemmata, Theoreme).

## 2. **Typprüfen:**

Überprüfen der statischen Semantik der Spezifikation. Typkorrektheit kann von sogenannten *Typkorrektheitsbedingungen* (type correctness conditions, TCCs) abhängen.

## 3. **Beweisen:**

Beweis der Typkorrektheitsbedingungen, Beweis der deklarierten Lemmata und Theoreme.

# Spezifikationen

- *Theorien* bilden die Strukturierungseinheiten für Spezifikationen in PVS. Theorien haben im wesentlichen folgende Form:

```
<Name> [<formale Parameter>] : THEORY
BEGIN
  ASSUMING
    <Assumptions>
  ENDASSUMING
  IMPORTING <Theorien ...>
  <Deklarationen, Definitionen, Formeln>
END
```

- Anlegen einer neuen Theorie mit M-x nf, danach Angabe eines Namens.
- Speichern von Theorien mit C-x C-s. Nach wichtigen Interaktionen werden die Theorien automatisch gespeichert.
- Bereits bestehende Dateien können mit C-c C-f geladen werden.

# Elemente von PVS Theorien: Typen

- Deklaration mit *name* : TYPE
- vordefinierte Typen: `bool`, `nat`, `int`, `real`, `list`, siehe Prelude-Datei (`M-x view-prelude-file`).
- Definition neuer Typen:

Records:            `rec` : TYPE = `[# a:A, b:B #]`

Funktionstypen:    `fun` : TYPE = `[int, int -> int]`

Subtypen:           `posnat` : TYPE = `{x:nat | x>0}`

# Elemente von PVS Theorien: Konstanten

- Definitionen durch Angabe von Name, Type und Wert

```
five      : nat          = 5
pair      : [nat, bool] = (five, TRUE)
primes    : list[nat]    = (: 2, 3, 5 :)
                               % = cons(2, cons(3, cons(5, null)))
```

- Funktionskonstanten:

```
inc1      : [nat -> nat] = LAMBDA (n:nat): n+1
inc2(n:nat) : nat       = n+2
```

# Elemente von PVS Theorien: Variablen

- Einführung durch Angabe von Bezeichner und Typ

```
x : VAR nat
```

- zur Abkürzung von Funktionsdefinitionen

```
inc1      : [nat -> nat] = LAMBDA n: n+1  
inc2(n)   : nat         = n+2
```

- zur impliziten All-Quantifizierung von Formeln

```
thm : THEOREM  
  x < x + 1
```

bedeutet:

```
thm : THEOREM  
  FORALL (x:nat): x < x + 1
```

# Elemente von PVS Theorien: Formeln

- *Formeln* sind Ausdrücke vom Typ `bool`.
- vordefinierte Junktoren: NOT, AND, OR, IMPLIES, IFF
- Quantifizierte Ausdrücke ( $P : [T \rightarrow \text{bool}]$ , Abk.:  $P : \text{pred}[T]$ ):  
EXISTS  $(x:T) : P(x)$ ,      FORALL  $(y:T) : P(y)$
- In eine Theorie werden Formeln eingeführt durch:  
`<Name> : <Bezeichnung> <Formel>`
- Je nach Bezeichnung werden die Formel unterschiedlich behandelt:
  - AXIOM: Formel wird als wahr angenommen und kann bei Beweisen als Voraussetzung einfließen.
  - LEMMA, THEOREM (etc.): Für die Formel muss (irgendwann) ein Beweis geliefert werden.
- Überblick über die Spezifikationssprache von PVS findet sich in  
`~hp2/PVS2.4/Doc/pvs-language-reference.ps`

# Beispieltheorie

```
aussagenlogik : THEORY
BEGIN
  PROP      : TYPE+           % -- Typ einer aussagenlogischen Formel
  A,B,C    : PROP            % -- einige Aussagensymbole
  valid    : [PROP -> bool]  % -- Gueltigkeit einer aussagenlogischen Formel
  -        : [PROP -> PROP]; % -- Negation
  /\,\/,=> : [PROP,PROP -> PROP]; % -- Konjunktion, Disjunktion, Implikation

  X,Y      : VAR PROP

  valid_not : AXIOM           % -- Axiome beschreiben, wann zusammengesetzte
    valid(-X) IFF NOT valid(X) % -- Formeln gueltig sind

  valid_and : AXIOM
    valid(X /\ Y) IFF valid(X) AND valid(Y)

  valid_or  : AXIOM
    valid(X \/ Y) IFF valid(X) OR valid(Y)

  valid_impl : AXIOM
    valid(X => Y) IFF valid(-X \/ Y)

  trivial   : THEOREM        % ----- Behauptungen
    valid(A => (A /\ A))

END aussagenlogik
```

# Beweiskommandos (1)

Hilfe zu einzelnen Beweiskommandos: (`help <Kommando>`). Überblick über alle Kommandos unter `hp2/PVS2.4/Doc/pvs-prover-guide.ps`

Propositionale Regeln:

- (`flatten`), (`split`), (`case`) siehe oben.
- (`lift-if &rest fnums[*]`): hebt bedingte Ausdrücke (in den Formeln *fnums*) auf oberste Ebene an.

## Beweiskommandos (2)

Quantorenregeln:

- `(inst)`, `(skolem)` siehe oben.
- `(inst? &opt fnum[*] subst)`: Ohne Parameter: heuristische Instanzierung aller Formeln. Mit Schlüsselwort `:subst` und Liste der Form `(var1 term1 var2 term2 ...)` kann Substitution vorgegeben werden.
- `(skolem! &opt fnum[*])`: automatische Generierung von Skolemkonstanten.
- `(skosimp*)`: wiederholtes Skolemisieren und disjunktive Vereinfachungen (`flatten`).

## Beweiskommandos (3)

Gleichheitsregeln:

- `(beta &opt fnum)`: Betareduktion in Formel *fnum*. Auch anwendbar für LET-Ausdrücke.
- `(replace fnum &opt fnums[*] dir[LR])`: *fnum* ist Formel der Art  $l = r$ , *l* wird durch *r* ersetzt. Bei `:dir RL` umgekehrt.
- `(replace*)`: alle möglichen Ersetzungen werden vorgenommen.
- `(case-replace l = r)`: Ersetzt *l* durch *r*, Subgoal: Nachweis der Gleichheit.

## Beweiskommandos (4)

Regeln für Definitionen und Lemmas:

- (`expand name &opt fnum[*] occurrence if-simplifies`): expandiert Definition *name* in Formel *fnum*.  
Mit `:occurrence zahl` kann ein bestimmtes Vorkommen ersetzt werden.  
`:if-simplifies t`: Expansion nur, wenn das Resultat vereinfacht werden kann.
- (`lemma name &opt :subst ("var" "term"...)`): Laden des Lemmas *name* in den Antezedent.
- (`rewrite name &opt fnums[*] subst dir[LR]`):  
Lemma *name* als Termersetzungsregel.

## Der IF - THEN - ELSE - Ausdruck

Bei `IF cond THEN expr1 ELSE expr2 ENDIF` handelt es sich um einen Term und nicht um ein Kontrollkonstrukt. Der Typ dieses Terms ist der (gemeinsame) Typ von `expr1` und `expr2`, wobei `cond` von Typ `bool` ist.

Dem Beweis von `IF A THEN B ELSE C ENDIF` entspricht der Nachweis von `A IMPLIES B` und `NOT A IMPLIES C`. Es lassen sich also die aussagenlogischen Regeln (`split`) und (`flatten`) verwenden.

Durch (`lift-if`) kann eine Funktionsanwendung in einen IF-Ausdruck geschoben werden.

```
|-----  
{1} ... f(IF cond THEN a ELSE b ENDIF) ...
```

Rule? (`lift-if`)

```
|-----  
{1} ... IF cond THEN f(a) ELSE f(b) ENDIF ...
```

# Beispielsitzung (1)

Nach dem Starten von PVS erscheint der Welcome-Buffer. Neue Theorien werden mit `M-x nf` und Angabe eines Namens (hier `predtest`) erzeugt:

```
predtest % [ parameters ]
          : THEORY
BEGIN
% ASSUMING
  % assuming declarations
% ENDASSUMING
END predtest
```

Erstellen der Theorie als Textdatei in Emacs. Speichern mit `C-x C-s`.

```
predtest : THEORY
BEGIN
  T      : TYPE
  P,Q    : [T -> bool]

  predthm : THEOREM
    (FORALL (x:T): P(x) AND Q(x)) IMPLIES
      (FORALL (x:T): P(x)) AND (FORALL (x:T): Q(x))
END predtest
```

## Beispielsitzung (2)

Mit `M-x tc` wird die Theorie auf Typkorrektheit hin überprüft:

```
Parsing prectest
```

```
prectest parsed in 4 seconds
```

```
Typechecking prectest
```

```
prectest typechecked in 8 seconds: No TCCs generated
```

Zum Beweis von `predthm` Cursor auf das Theorem setzen und `M-x pr` eingeben. Man gelangt in einen zweiten Buffer, in dem der Beweiser läuft. Zwischen den Buffern kann man mit `C-x o` hin- und herwechseln.

```
predthm :
```

```
|-----
```

```
{1}      (FORALL x: P(x) AND Q(x))
```

```
          IMPLIES (FORALL x: P(x)) AND (FORALL x: Q(x))
```

```
Rule?
```

PVS erwartet nun die Angabe einer Beweisregel. Beweis siehe Demo.

## Beispielsitzung (3)

Nach Beendigung des Beweises wird dieser in der Datei `predtest.prf` abgespeichert. PVS führt Buch über bereits bewiesene Theoreme und deren Abhängigkeiten untereinander. Status kann mit `M-x status-proof-theory` abgefragt werden:

```
Proof summary for theory predtest
  predthm.....proved - complete
  Theory totals: 1 formulas, 1 attempted, 1 succeeded.
```

Beendigung von PVS mit `C-x C-c`. **Hinweis:** Zur Effizienzsteigerung schreibt PVS den Zustand typkorrekter Theorien in Binärdateien (z.B. `predtest.bin`). Diese können unter Umständen recht umfangreich werden. Bei Platzproblemen können die Binärdateien ohne Probleme gelöscht werden.