

Kurzanleitung zu PVS Teil 2

PVS Beweisregeln (Fortsetzung)

Fallunterscheidung

Beweisregel: (CASE)

Mit (CASE A) kann eine Fallunterscheidung bezüglich der Formel A durchgeführt werden. Es werden zwei Unterziele erzeugt: im ersten wird A als wahr angenommen und kann als Voraussetzung benutzt werden, im zweiten muss die Gültigkeit von A nachgewiesen werden (bzw. wird A als falsch angenommen).

```
beispiel :
{-1}  x!1 < y!1
|-----
{1}   EXISTS r: x!1 < r AND r < y!1

Rule? (case "x!1 >= 0")
Case splitting on
  x!1 >= 0,
this yields 2 subgoals:
beispiel.1 :

{-1}  x!1 >= 0
[-2]  x!1 < y!1
|-----
[1]   EXISTS r: x!1 < r AND r < y!1

Rule? (postpone)
Postponing beispiel.1.

beispiel.2 :

[-1]  x!1 < y!1
|-----
{1}   x!1 >= 0
[2]   EXISTS r: x!1 < r AND r < y!1
```

Als Argumente von (CASE) können mehrere Formeln verwendet werden, wodurch in einem Schritt mehrfache Fallunterscheidungen durchgeführt werden können. Die Re-

gel (CASE $A_1 A_2 \dots A_n$) bewirkt dasselbe wie die schrittweise Anwendung der Regeln (CASE A_1), (CASE A_2), ..., (CASE A_n).

```
(case &rest formulas):
  Splits according to the truth or falsity of the formulas in FORMULAS.
(CASE a b c) on a sequent A |- B generates subgoals:
a, b, c, A |- B;
  a, b, A |- c, B;
  a, A |- b, B;
  A |- a, B.
See also CASE-REPLACE, CASE*
```

Beweisregel: (CASE*)

Mit (CASE* $A_1 A_2 \dots A_n$) kann eine vollständige Fallunterscheidung bezüglich der Formeln A_i durchgeführt werden, bei der in jedem Zweig nach allen Formeln aufgesplittet wird. Die Beweisregel erzeugt in diesem Fall 2^n neue Unterziele.

```
beispiel :
  |-----
{1}  A

Rule? (case* "a>0" "b>0")
Case-splitting fully on
  (a>0 b>0), ,
this yields 4 subgoals:
beispiel.1 :

{-1} b > 0
{-2} a > 0
  |-----
[1]  A

Rule? (postpone)
Postponing beispiel.1.

beispiel.2 :

{-1} a > 0
  |-----
{1}  b > 0
[2]  A

Rule? (postpone)
Postponing beispiel.2.
```

```

beispiel.3 :

{-1}  b > 0
  |-----
{1}   a > 0
[2]   A

Rule? (postpone)
Postponing beispiel.3.
    
```

```

beispiel.4 :

  |-----
{1}  b > 0
{2}  a > 0
[3]  A
    
```

```

(case*/$ &rest formulas) :
  Complete version of CASE command where all the formulas are case
  split along every branch.
    
```

Regeln für Definitionen und Lemmata

Beweisregeln: (EXPAND) und (EXPAND*)

Diese Regeln dienen zur Expansion von Definitionen. Die zu expandierenden Vorkommen des Funktionsnamens können auf einzelne Formeln (mit `:fnum`) und bestimmte Stellen innerhalb dieser Formeln (mit `:occurrence`) eingeschränkt werden.

```

% Im Beispiel ist f(n) = n + 1

beispiel :

{-1}  f(n!1) > g(a)
  |-----
{1}   f(g(a)) - 1 < f(n!1)

Rule? (expand "f" :fnum 1 :occurrence 1)
Expanding the definition of f,
this simplifies to:
beispiel :

[-1]  f(n!1) > g(a)
  |-----
{1}   g(a) < f(n!1)
    
```

Mit der zweiten Form können alle Namen auf einmal expandiert werden.

```
% Im Beispiel ist  $f(n) = n + 1$  und  $g(n) = n * n$ 

beispiel :

|-----
{1}  g(f(n!1)) = f(n!1) * f(n!1)

Rule? (expand* "f" "g")
Expanding the definition(s) of (f g),
Q.E.D.
```

```
(EXPAND FUNCTION-NAME &OPTIONAL
 ((FNUM *) OCCURRENCE IF-SIMPLIFIES ASSERT?):
Expands (and simplifies) the definition of FUNCTION-NAME at a given
OCCURRENCE.  If no OCCURRENCE is given, then all instances of the
definition are expanded.  The OCCURRENCE is given as a number n
referring to the nth occurrence of the function symbol counting
from the left, or as a list of such numbers.
If the IF-SIMPLIFIES flag is T, then any definition expansion occurs only
if the RHS instance simplifies (using the decision procedures).  Note that
the EXPAND step also applies simplification with decision procedures
(i.e. SIMPLIFY with default options) to any sequent formulas where
an expansion has occurred.
ASSERT? can be either NONE (meaning no simplification),
NIL (meaning simplify using SIMPLIFY), or T (meaning simplify using
ASSERT).

(EXPAND*/$ &REST NAMES) :
Expands all the given names and simplifies.
```

Beweisregel: (LEMMA)

Führt ein Lemma in den Antezedenten ein. Optional kann eine partielle Instanzierung angegeben werden, vgl. (INST?).

```

beispiel :
  |-----
{1}  x!1 * x!1 > 0

Rule? (lemma " pos_times_gt")
Applying pos_times_gt
this simplifies to:
beispiel :

{-1}  FORALL (x, y: real):
      x * y > 0 IFF (0 > x AND 0 > y) OR (x > 0 AND y > 0)
  |-----
[1]  x!1 * x!1 > 0
    
```

```

(LEMMA NAME &OPTIONAL (SUBST)):
  Adds lemma named NAME as the first antecedent formula
  after applying the substitutions in SUBST to it.
  Example: (LEMMA "assoc" ("x" 1 "y" 2 "z" 3)).
    
```

Beweisregel: (USE)

Kombiniert (LEMMA) und (INST?).

```

beispiel :
  |-----
{1}  x!1 * x!1 > 0

Rule? (use " pos_times_gt")
Using lemma pos_times_gt,
this simplifies to:
beispiel :

{-1}  x!1 * x!1 > 0 IFF (0 > x!1 AND 0 > x!1) OR (x!1 > 0 AND x!1 > 0)
  |-----
[1]  x!1 * x!1 > 0
    
```

```

(USE/$ LEMMA &OPTIONAL SUBST (IF-MATCH BEST) (INSTANTIATOR INST?)) :
  Introduces lemma LEMMA, then does BETA and INST? (repeatedly) on the lemma.
  The INSTANTIATOR argument may be used to specify an alternative to INST?.
    
```