

Kurzanleitung zu PVS Teil 3

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 Regel (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 Gleichheit

Beweisregel: (BETA)

Führt β -Reduktionen durch. Anwendbar nicht nur auf LAMBDA-Ausdrücke, sondern auch auf LET und Projektionen auf Tupel (PROJ_1(tupel) bzw. tupel'1 etc.).

beispiel :

```
|-----
{1}  LET x = 5, t = (x, y!1) IN t'1 < (LAMBDA z: t'2 * z)(4)
```

Rule? (beta)

Applying beta-reduction,
this simplifies to:

beispiel :

```
|-----
{1}  5 < y!1 * 4
```

(BETA &OPTIONAL ((FNUMS *) REWRITE-FLAG)):

Beta-reduces chosen formulas. If REWRITE-FLAG is LR(RL), then left(right)-hand-side is left undisturbed for rewriting using REWRITE and REWRITE-LEMMA.

Example reduction steps are:

(LAMBDA x, y: x + y)(2, 3) to 2 + 3
(LET x := 1, y := 2 in x + y) to 1 + 2
b((# a:=1, b:= 2 #)) to 2
PROJ_2(2, 3) to 3
cons?(nil) to FALSE
car(cons(2, nil)) to 2.

Beweisregel: (REPLACE)

Ist die Antezedent-Formel $-n$ von der Gestalt $l = r$, so werden mit (REPLACE $-n$ fnums) alle in fnums vorkommenden Teilterme l durch r ersetzt. Soll die Ersetzung umgekehrt durchgeführt werden, so ist die Schlüsselwortoption :DIR RL anzugeben.

beispiel :

```
{-1}  x!1 = 1
|-----
{1}   x!1 * x!1 = 1
```

Rule? (replace -1)
Replacing using formula -1,
this simplifies to:
beispiel :

```
[-1]  x!1 = 1
|-----
{1}   1 * 1 = 1
```

(REPLACE FNUM &OPTIONAL ((FNUMS *) DIR HIDE? ACTUALS?)):

Rewrites the given formulas in FNUMS with the formula FNUM. If FNUM is an antecedent equality, then it rewrites left-to-right if DIR is LR (the default), and right-to-left if DIR is RL. If FNUM is not an antecedent equality, then any occurrence of the formula FNUM in FNUMS is replaced by TRUE if FNUM is an antecedent, FALSE for a succedent. If HIDE? is T, then FNUM is hidden afterward. When ACTUALS? is T, the replacement is done within actuals of names in addition to the expression level replacements.

Beweisregel: (REPLACE*)

Mehrfaches REPLACE: alle (bzw. die mit dem Argument *fnums* angegebenen) Gleichheiten werden für die Ersetzung benutzt.

beispiel :

```
{-1} x!1 = 2
{-2} y!1 = z!1 + x!1
|-----
{1} x!1 * y!1 * z!1 > c
```

Rule? (replace*)

Repeatedly applying the replace rule,
this simplifies to:

beispiel :

```
[-1] x!1 = 2
{-2} y!1 = z!1 + 2
|-----
{1} (2 * (z!1 + 2)) * z!1 > c
```

(REPLACE*/\$ &REST FNUMS) :

Apply left-to-right replacement with formulas in FNUMS.

Beweisregel: (CASE-REPLACE)

Die Regel (CASE-REPLACE) kombiniert die Regeln (CASE) und (REPLACE) für die Fälle, in denen die Fallunterscheidungsformel eine Gleichung darstellt.

```
beispiel :  
  
  |-----  
{1}  x!1 * y!1 * z!1 > c  
  
Rule? (case-replace "x!1 = 0")  
Assuming and applying x!1 = 0,  
this yields 2 subgoals:  
beispiel.1 :  
  
{-1}  x!1 = 0  
  |-----  
{1}  0 * y!1 * z!1 > c  
  
Rule? (postpone)  
Postponing beispiel.1.  
  
beispiel.2 :  
  
  |-----  
{1}  x!1 = 0  
[2]  x!1 * y!1 * z!1 > c
```

Im obigen Beispiel kann die Aufnahme der Formel $x!1 = 0$ in den Antezedenten des ersten neuen Unterziels auch unterdrückt werden: (CASE-REPLACE "x!1 = 0" :hide? T).

```
(case-replace/$ formula &optional hide?) :  
  Case splits on a given FORMULA lhs=rhs and replaces lhs by rhs.  
See also CASE, CASE*. Hides FORMULA when HIDE? is T.
```

Beweisregel: (REWRITE)

(REWRITE) ist die diejenige Regel, mit der in PVS Termersetzung durchgeführt werden kann. Besitzt eine Voraussetzung die Form einer Gleichheit $L = R$, so werden durch (REWRITE) alle im aktuellen Beweisziel auftretenden Terme L' durch R' ersetzt, wobei L' aus L bzw. R' aus R durch geeignete Substitution von Variablen hervorgeht. Als Voraussetzungen können dabei entweder Axiome oder Theoreme, aber auch Antezedent-Formeln benutzt werden.

```
Rule? (help rewrite)

(REWRITE/$ LEMMA-OR-FNUM &OPTIONAL (FNUMS *) SUBST (TARGET-FNUMS *) (DIR LR)
      (ORDER IN)) :
  Rewrites using LEMMA-OR-FNUM (lemma name or fnum) of the form
  H IMPLIES L = R by finding match L' for L and replacing L' by R' with
  subgoal proof obligations for H'.
  A lemma H IMPLIES L is treated as H IMPLIES L = TRUE, and also H can be
  empty.
  FNUMS constrains where to search for a match,
  SUBST takes a partial substitution and tries to find a match extending this,
  TARGET-FNUMS constrains where the rewriting occurs,
  DIR is left-to-right(LR) or right-to-left(RL),
  ORDER is inside-out(IN) or outside-in (OUT).
```

Die Anwendungsmöglichkeiten von (REWRITE) sind vielfältig. Zunächst reicht es für unsere Zwecke aber aus, die Bedeutung einiger weniger Varianten zu kennen. In der einfachsten Form wird (REWRITE) nur der Name des anzuwendenden Axioms oder Theorems übergeben:

```
beispiel :

  |-----
{1}  (i(a) * a) * b = b

Rule? (rewrite "assoziativ")
Found matching substitution:
z: G gets b,
y gets a,
x gets i(a),
Rewriting using assoziativ, matching in *,
this simplifies to:
beispiel :

  |-----
{1}  i(a) * (a * b) = b

Rule?
```

Im Gegensatz zu gerichteten Termersetzungsregeln können Gleichungen natürlich auch von rechts nach links angewendet werden. Dazu dient das Schlüsselwortargument :DIR. Wird als Argument hier r1 angegeben, wird die entsprechende Gleichung in umgekehrter Richtung angewendet:

```
beispiel :  
  
|-----  
{1}  i(a) * (a * b) = b  
  
Rule? (rewrite "assoziativ" :dir rl)  
Found matching substitution:  
z: G gets b,  
y gets a,  
x gets i(a),  
Rewriting using assoziativ, matching in *,  
this simplifies to:  
beispiel :  
  
|-----  
{1}  (i(a) * a) * b = b  
  
Rule?
```

PVS sucht automatisch nach einer geeigneten Variablensubstitution, die die linke Seite der Gleichung mit dem Beweisziel unifiziert. Wie bei (INST?) kann aber auch eine partielle Substitutionsliste über das Schlüsselwortargument `:subst` angegeben werden.

```
beispiel :  
  
|-----  
{1}  (i(a) * a) * b = b  
  
Rule? (rewrite "assoziativ" :subst ("x" "i(a)" "y" "a"))  
Found matching substitution:  
z: G gets b,  
x gets i(a),  
y gets a,  
Rewriting using assoziativ, matching in * where  
  x gets i(a),  
  y gets a,  
this simplifies to:  
beispiel :  
  
|-----  
{1}  i(a) * (a * b) = b  
  
Rule?
```

Die (REWRITE)-Regel ersetzt grundsätzlich alle im Beweisziel vorkommenden passenden Teilterme. Dies lässt sich zwar noch durch Angabe des Schlüsselwortarguments `:TARGET-FNUMS` auf eine bestimmte Formel in der Sequenz einschränken. Tritt aber wie im folgenden Beispiel der zu ersetzende Teilterm – hier `x!1` – mehrfach in einer Formel auf und sollen aber nur bestimmte Vorkommen ersetzt werden, hier z. B. das erste

Vorkommen von $x!1$ durch $e * x!1$, so reicht (leider) (REWRITE) nicht aus.

```
rechtsneutral :
  |-----
  {1}  x!1 * e = x!1

Rule? (rewrite "linksneutral" :dir rl)
Found matching substitution:
x: G gets x!1,
Rewriting using linksneutral, matching in *,
this simplifies to:
rechtsneutral :

  |-----
  {1}  e * x!1 * e = e * x!1

Rule?
```

In diesem Fall muss man sich eines kleinen Umwegs bedienen und die Beweiser-Regel CASE-REPLACE anwenden, welcher als Argument eine Gleichung der Form $L = R$ übergeben wird. Diese Formel wird als neue Annahme in das Beweisziel aufgenommen und gleichzeitig alle im Beweisziel auftretenden Terme L durch die rechte Seite R ersetzt. Im obigen Beispiel verwenden wir als Argument die Gleichung $x!1 * e = (e * x!1) * e$, welches gerade angibt, wie wir die linke Seite im aktuellen Beweisziel verändern möchten:

```
rechtsneutral :

  |-----
  {1}  x!1 * e = x!1

Rule? (CASE-REPLACE "x!1*e = (e*x!1)*e")
Assuming and applying x!1*e = (e*x!1)*e,
this yields 2 subgoals:
rechtsneutral.1 :

{-1}  x!1 * e = (e * x!1) * e
  |-----
  {1}  (e * x!1) * e = x!1

Rule?
```

Hier kann der Beweis nun wie gewohnt fortgesetzt werden. Die eingeführte zusätzliche Annahme ist in diesem Fall nicht weiter nötig; wen sie stört, der kann sie mit der Regel (HIDE -1) verstecken, bzw. mit (DELETE -1) ganz löschen.

Selbstverständlich muss die neu aufgenommene Annahme aber auch gültig sein. Um dies sicherzustellen, erzeugt die Regel (CASE-REPLACE) noch ein zweites Teilziel, wo wir exakt die als Argument verwendete Gleichung beweisen müssen:

```
rechtsneutral.1 :  
  
{-1}  x!1 * e = (e * x!1) * e  
|-----  
{1}   (e * x!1) * e = x!1  
  
Rule? (postpone)  
Postponing rechtsneutral.1.  
  
rechtsneutral.2 :  
  
|-----  
{1}   x!1 * e = (e * x!1) * e  
[2]   x!1 * e = x!1  
  
Rule?
```

Um die Formel {1} zu beweisen, kann nun die Regel REWRITE wie ursprünglich geplant angewendet werden.

```
rechtsneutral.2 :  
  
|-----  
{1}   x!1 * e = (e * x!1) * e  
[2]   x!1 * e = x!1  
  
Rule? (rewrite "linksneutral")  
Found matching substitution:  
x: G gets x!1,  
Rewriting using linksneutral, matching in *,  
  
This completes the proof of rechtsneutral.2.  
  
rechtsneutral.1 :  
  
{-1}  x!1 * e = (e * x!1) * e  
|-----  
{1}   (e * x!1) * e = x!1  
  
Rule?
```