[2.18] Delay evaluation to simplify expressions

Sometimes the CAS will not simplify expressions because it assumes general conditions for the variables. As an example, suppose that we have the expression

$$t^n$$

and we want to extract the exponent *n*. We could try using the identity

$$n = \frac{\ln(t^n)}{\ln(t)}$$

but the CAS will not simplify the right-hand expression to *n*, because this expression is undefined for t=0, and because this identity is *not generally true* for t<0. So, if we constrain the expression for t>0, like this

```
ln(t^n)/(ln(t))|t>Ø
```

the CAS returns 'n' as expected. While this example had a fairly simple solution, you may want to use this technique for more complex expressions in which the constraints are not immediately obvious. In addition, you may need to apply the constraints sequentially to get the desired result. To continue with the same example, suppose that we want to use the constraints

```
t=p^2   and     n=q^2
```

This will result in the desired simplification, since

```
ln(t^n)/(ln(t))|t=p^2 and n=q^2
```

returns q^2, which is just *n*, as desired. But this needs one more substitution, q^2 = n, to complete the simplification, and this *does not* work:

```
(ln(t^n)/(ln(t))|t=p^2 and n=q^2)|q^2=n
```

and in fact returns *Memory error!*

To avoid this problem, Bhuvanesh Bhatt offers the following program:

```
delay(xpr,constr)
Func
©delay(xpr,constr) evaluates xpr and then imposes constraint
©Copyright Bhuvanesh Bhatt
©December 1999
if gettype(constr)≠"EXPR" and gettype(constr)≠"LIST":return "Error: argument"
if gettype(constr)="EXPR"
return xpr|constr
local i,tmp:1→i:while i≤dim(constr):constr[i]→tmp:xpr|tmp→xpr:i+1→i:endwhile:xpr
EndFunc
```

xpr is an expression to be evaluated, and may include constraints. *constr* may be either an expression or a list. If *constr* is a list of constraints, then each constraint is evaluated in the *while()* loop. The call to evaluate our example is

```
delay(ln(t^n)/ln(t)|t=p^2 and n=q^2,p^2=t and q^2=n)
```

which returns *n*.

Finally, note there is a simpler way to extract the exponent in this example, by using *part():*

```
part(t^n,2)
```

returns *n*. This works regardless of the complexity of *t* or *n*, for example,

```
part((a*t+2)^(3*sin(n),2)
```

returns 3*sin(n).

(credit to Bhuvanesh Bhatt)