utilities

the split routines

The provided routine

$$\text{split0}(\text{liste}, n_{R0}, n_{R1}) :=$$

```
cl ← cols(liste) − 1                                                    *
for i ∈ 0 .. rows(liste) − 1
    if liste_{i, n_R0} = 0 ∧ liste_{i, n_R1} = 0
        liste_1 ← stack(liste_1, submatrix(liste, i, i, 0, cl))  if rows(liste_1) > 0
        liste_1 ← submatrix(liste, i, i, 0, cl)  otherwise
    otherwise
        liste_2 ← stack(liste_2, submatrix(liste, i, i, 0, cl))  if rows(liste_2) > 0
        liste_2 ← submatrix(liste, i, i, 0, cl)  otherwise
return (liste_1 ; liste_2)
```

All routines have no error control if one of the lists is empty!

$$\text{split1}(\text{liste}, n_{R0}, n_{R1}) :=$$

```
cl ← cols(liste) − 1                                                    *
liste1_{0, cl} ← 123
liste2 ← liste1
for i ∈ 0 .. rows(liste) − 1
    zeile ← (liste^T)^⟨i⟩^T
    liste1 ← stack(liste1, zeile)  if liste_{i, n_R0} = 0 ∧ liste_{i, n_R1} = 0
    liste2 ← stack(liste2, zeile)  otherwise
return (submatrix(liste1, 1, rows(liste1) − 1, 0, cl) ; submatrix(liste2, 1, rows(liste2) − 1, 0, cl))
```

The trick with transpose/column select/transpose is MUCH slower than submatrix!

```
split2(liste, n_R0, n_R1) :=
    cl ← cols(liste) − 1                                                    *
    liste1_{0,cl} ← 123
    liste2 ← liste1
    for i ∈ 0 .. rows(liste) − 1
        zeile ← submatrix(liste, i, i, 0, cl)
        liste1 ← stack(liste1, zeile)   if liste_{i,n_R0} = 0 ∧ liste_{i,n_R1} = 0
        liste2 ← stack(liste2, zeile)   otherwise
    return ( submatrix(liste1, 1, rows(liste1) − 1, 0, cl)
             submatrix(liste2, 1, rows(liste2) − 1, 0, cl) )
```

I would have thought that elimination of the if/otherwise by using dummy first rows and omitting the second submatrix call by assigning the result to a variable would speed up things a bit more.
split3 need approx down to half the time (not always!?) of your original routine which is not that spectacular and the trick with those dummy columns is not really that nice.
I guess using the same name for the transposed matrix will help saving memory.

```
split3(M, n_R0, n_R1) :=
    M ← M^T                                                                 *
    zl ← rows(M) − 1
    liste1 ← M^⟨0⟩
    liste2 ← liste1
    for i ∈ 0 .. cols(M) − 1
        spalte ← M^⟨i⟩
        liste1 ← augment(liste1, spalte)   if spalte_{n_R0} = 0 ∧ spalte_{n_R1} = 0
        liste2 ← augment(liste2, spalte)   otherwise
    return ( submatrix(liste1, 0, zl, 1, cols(liste1) − 1)^T
             submatrix(liste2, 0, zl, 1, cols(liste2) − 1)^T )
```

This routine is up to 5o times quicker than the original one, but the results are nested vectors.
If flattening them is mandatory that would cost quite some time, I suspect.

$$
\begin{aligned}
&\text{split4}(M, n_{R0}, n_{R1}) := \\
&\quad M \leftarrow M^{T} \\
&\quad zl \leftarrow \text{rows}(M) - 1 \\
&\quad \text{liste1} \leftarrow 0 \\
&\quad \text{liste2} \leftarrow 0 \\
&\quad \text{for } i \in 0 .. \text{cols}(M) - 1 \\
&\quad\quad \text{spalte} \leftarrow M^{\langle i \rangle} \\
&\quad\quad \text{liste1}_{\text{rows(liste1)}} \leftarrow \text{spalte}^{T} \quad \text{if } \text{spalte}_{n_{R0}} = 0 \wedge \text{spalte}_{n_{R1}} = 0 \\
&\quad\quad \text{liste2}_{\text{rows(liste2)}} \leftarrow \text{spalte}^{T} \quad \text{otherwise} \\
&\quad \text{return } \begin{pmatrix} \text{liste1} \\ \text{liste2} \end{pmatrix}
\end{aligned}
\qquad *
$$

So lets try flattening and the speed is in the range of the other routines :-(

$$
\begin{aligned}
&\text{split5}(M, n_{R0}, n_{R1}) := \\
&\quad M \leftarrow M^{T} \\
&\quad zl \leftarrow \text{rows}(M) - 1 \\
&\quad \text{liste1} \leftarrow 0 \\
&\quad \text{liste2} \leftarrow 0 \\
&\quad \text{for } i \in 0 .. \text{cols}(M) - 1 \\
&\quad\quad \text{spalte} \leftarrow M^{\langle i \rangle} \\
&\quad\quad \text{liste1}_{\text{rows(liste1)}} \leftarrow \text{spalte}^{T} \quad \text{if } \text{spalte}_{n_{R0}} = 0 \wedge \text{spalte}_{n_{R1}} = 0 \\
&\quad\quad \text{liste2}_{\text{rows(liste2)}} \leftarrow \text{spalte}^{T} \quad \text{otherwise} \\
&\quad \text{return } \begin{pmatrix} \text{flatten(liste1)} \\ \text{flatten(liste2)} \end{pmatrix}
\end{aligned}
\qquad *
$$

OK, now lets do it the hard way, transferring all elelemnts singly using loops instead of stack or augment. One might think that the built in routines would perform better, but ... surprise!

```
split6(M, a, b) :=
    L1 ← 0
    L2 ← 0
    spalten ← cols(M) − 1
    for zl ∈ 0 .. rows(M) − 1
        if M_{zl,a} = 0 ∧ M_{zl,b} = 0
            for sp ∈ 0 .. spalten
                List1_{L1,sp} ← M_{zl,sp}
            L1 ← L1 + 1
        otherwise
            for sp ∈ 0 .. spalten
                List2_{L2,sp} ← M_{zl,sp}
            L2 ← L2 + 1
    return (List1 / List2)
```

*

So lets try number 5 again but with a rewritten flattening routine (not using stack):

```
split7(M, n_R0, n_R1) :=
    M ← M^T
    zl ← rows(M) − 1
    liste1 ← 0
    liste2 ← 0
    for i ∈ 0 .. cols(M) − 1
        spalte ← M^⟨i⟩
        liste1_{rows(liste1)} ← spalte^T   if spalte_{n_R0} = 0 ∧ spalte_{n_R1} = 0
        liste2_{rows(liste2)} ← spalte^T   otherwise
    return (flatten2(liste1) / flatten2(liste2))
```

*

the split routines

Nur zum Test, ob alle Routinen das gleiche Ergebnis liefern

Size of Matrix to be splitted: max := 1000

Number of times the splitting routines are called for taking the time: anz := 10

Liste := Erzeuge_Feld(max)

$t_7 := Zeit(split7, Liste, anz) = 2.356$

$t_6 := Zeit(split6, Liste, anz) = 1.716$

$t_5 := Zeit(split5, Liste, anz) = 17.129$

$t_4 := Zeit(split4, Liste, anz) = 0.202$

$t_3 := Zeit(split3, Liste, anz) = 16.302$

$t_2 := Zeit(split2, Liste, anz) = 17.862$

$t_1 := Zeit(split1, Liste, anz) = 158.918$

$t_0 := Zeit(split0, Liste, anz) = 18.236$

RANKING

$$csort(T, 0) = \begin{pmatrix} 0.202 & "split4" \\ 1.716 & "split6" \\ 2.356 & "split7" \\ 16.302 & "split3" \\ 17.129 & "split5" \\ 17.862 & "split2" \\ 18.236 & "split0" \\ 158.918 & "split1" \end{pmatrix}$$

```
T := | t ← t
     | for i ∈ 0 .. last(t)
     |   t_{i,1} ← concat("split", num2str(i))
     | return t
```