ETVO CalculatorET (and its online version) is a basic interpreter to handle series and matrices in dioid ET (see Johannes Trunk PhD). The variables describe operators which can be both Event-variant and Time-variant. The series have both a E-gain (gain in the event domain) and a T-gain (gain in the time domain)

The case where the T-gain is 1 (no time multiplier or divider) corresponds to series in E[[d]] and the case where the E-gain is 1 (no event multiplier or divider) corresponds to series in T[[g]]. Finally, the case where E-gain and T-gain is 1 corresponds to series in MinMax[[g,d]] (neither multiplier nor divider).

The representation of the series lies on a core decomposition L.Q.R where

Qed (core) matrix is in E[[d]], and L=(vx,d-1.vx, …) and R=(…,wy.d-1,wy)' are in T[[g]] .
or Qtg (core) matrix is in T[[g]], and the L=(mw,g1.mx,…) R=(…,by.g1,by)' are in E[[d]].

For example : (v2, d-1.v2).Qed.(w3.d-2,w3.d-1,w2)' where Qed is a 2x3 matrix in E[[d]].

IMPORTANT : the results are mostly compatible with the grammar of the parsers, so that it is possible to save intermediate results in text files for future use.

Variables : only 2 types defined by the first letter of the identifier (composed only by alpha numerical symbols)

s__ : series in ET
S__ : matrix with entries in ET

```
->s1= v2.m2.[g3.d2]*.b3.d5.w3
s1=L=(v2 .. d-1.v2)
{[0,2]=[g6.d2]*.(m2.b3.d5)}
R=(w3.d-2 .. w3)'
```

Expressions : *a command starting with % is ignored*

g2, d1 = event-shift and time-shift operators
m2, b3 = m2 (event-multiplier), b3 (event-divider)
v2, w3 = v2 (time-multiplier), w3 (time-divider)
m<1,2>, b<3,1,2> = cyclic multiplier/batch
d<3,2> = cyclic delay
exp1.exp2 = product (which is non commutative)
exp1 + exp2 = sum
inf(exp1,exp2) = inf
[exp]* = Kleene star
lfrac(exp1,exp2) rfrac(exp1,exp2) = residuation
pr+(exp) = projection in causal set
SA(0,1) = entry of a matrix
eps(r,c) = epsilon matrix of size r x c

```
->SA=eps(2,2)
->SA(0,0)=g3.m2.b2
->SA(1,1)=g2.d5.v2.w2
->SA(1,0)=d5
->SA(0,1)=m2.b2.g2.d7
->SAs=[SA]*
```

Functions :

toCoreEd(exp) = gives the decomposition with a core in Ed
toCoreTg(exp) = gives the decomposition with a core in Tg
right(exp) left(exp) = returns the series in the right or left form (term [gn.dt]* on the right/left)

randMM(nbTerms) = randomly generated MinMax[[g,d]] series
randEd(m,b,nbTerms) = randomly generated E[[d]] series gain m/b
randTg(v,w,nbTerms) = randomly generated T[[g]] series gain v/w

```
% random series where the E-gain is 2/2 and the T-gain is 2/3
->sr=randEd(2,2,2).randTg(2,3,1)
sr=L=(m2 .. g1.m2)
{[0,0]=[d2.g1]*.(d-1.v2.w3.d7.g1)
[0,1]=[d2.g1]*.(d-1.v2.w3.d7.g2)}
R=(b2.g1 .. b2)'
```

asRight(exp) = LQR is rewritten in a « flat version »
(s1,…,sy).(…,mx.by.g1,mx.by)' with sj in T[[g]]
(s1,…,sy).(…,vx.wy.d-1,vx.wy)' with sj in E[[d]]

```
->sr=randEd(2,2,2).randTg(2,3,1)
sr=L=(m2 .. g1.m2)
{[0,0]=[d3.g1]*.(v2.w3.d4.g1)
[0,1]=[d3.g1]*.(v2.w3.d4.g1)}
R=(b2.g1 .. b2)'
->asRight(sr)
((v2.w3.d4.g2+d-1.v2.w3.d10.g4).[d9.g4]*).m2.b2.g1+((v2.w3.d4.g2+d-
1.v2.w3.d10.g4).[d9.g4]*).m2.b2
```

asLeft(exp) = LQR is rewritten in a « flat version »
(mx.by,g1.mx.by,…).(s1,…,sx)' with sj in T[[g]]
(vx.wy,d-1.vx.wy,…).(s1,…,sx)' with sj in E[[d]]

asMuVarR(exp) = a « flat version » with m<seq> coefficients

```
->sa=randEd(2,3,2)
sa=(((m2.b3.g2+g1.m2.b3.g1).d0+m2.b3.g3.d2))+[g4.d7]*.(g1.m2.b3.g6.d6+m2.b3.g10.d9)
->asMuVarR(sa)
(((g0.m<1,1,0>.d0+g2.m<0,0,2>.d2))+(g5.m<0,0,2>.d6+g6.m<0,2,0>.d9).[g6.d7]*).d0
```

asMuVarL(exp) = a « flat version » with m<seq> coefficients

asDeltaVarR(exp) = a « flat version » with d<seq> coefficients (if T-gain=1)

```
->sa=randTg(3,3,1)
sa=(((v3.w3.d-2+d-2.v3.w3).g0))+[d3.g1]*.(v3.w3.d1.g1)
-> asDeltaVarR(sa)
 (((d<0,0,-1>.g0))+(d<3,2,1>.g1).[d3.g1]*).g0
```

asDeltaVarL(exp) = a « flat version » with d<seq> coefficients (if T-gain=1)