CoCoA-5 - Slug #1284

CartesianProductList: too slow

22 May 2019 16:15 - John Abbott

Status:	Closed	**Start date:**	22 May 2019
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	enhancing/improving	**Estimated time:**	1.99 hour
Target version:	CoCoA-5.3.0	**Spent time:**	2.15 hours

Description

Cartesian product operator is much faster than CartesianProductList.
Example:

```
t := CpuTime();
L := (0..1) >< (0..1) >< (0..1) >< (0..1) >< (0..1) >< (0..1) >< (0..1) >< (0..1) >< (0..1) >< (0.
.1) >< (0..1) >< (0..1) >< (0..1) >< (0..1) >< (0..1);
TimeFrom(t);
0.134

t := CpuTime();
LL := [0..1 | j in 1..15];
LLL := CartesianProductList(LL);
TimeFrom(t);
19.414
```

Also, if I call CartesianproductList with 16 factors instead 15 as above then the time increases to about 80s, which is much worse than linear!

History

#1 - 22 May 2019 16:18 - John Abbott

The examples above are not tiny, but also not so large. The resulting lists contained 32768 elements, and each element is a list with 15 (small) integers.

I did try the (dodgy) trick from #1228 comment 6, but that seemed to make no difference :-(

I also tried the tuples function (defined in combinatoria.cpkg5). It took 0.1s to generate the list against the explicit cartesian product taking 0.02s; so tuples is rather slow too.

#2 - 22 May 2019 16:30 - John Abbott

CartesianProductList is defined in list.cpkg5

Ange reported that CartesianProductList is slow; maybe I had already noticed this myself, but didn't take heed of it.

I wrote the following function for Ange:

```
define NextIndex(ref ind, limits)
  n := len(ind);
  while n > 0 and ind[n] = limits[n] do
    ind[n] := 1;
    n := n-1;
  endwhile;
```

```
  if n >= 1 then ind[n] := ind[n]+1; endif;
enddefine; -- NextIndex
```

The idea of the function is to generate successive indexes into the various factors of a cartesian product -- this suited Ange's intended use. The initial index should be [1,1,...,1]; the function then generates the next valid index in increasing lex-order.

#3 - 22 May 2019 22:05 - John Abbott

- Status changed from New to Resolved

- Assignee set to John Abbott

- % Done changed from 0 to 60

I suspect that the main problem is that append is terribly slow -- I'm pretty sure it makes needless copies.

Here is an implementation which seems to work better:

```
Define CPL(L)
  If len(L) = 0 Then Return [[]]; endif; -- or should this give error (to be more helpful)?
  if len(L) = 1 Then Return [[x] | x in L[1]]; endif;-- list of 1-tuples, see redmine 1239
  if len(L) = 2 Then Return L[1] >< L[2]; endif;
  if len(L) = 3 Then Return L[1] >< L[2] >< L[3]; endif;

  half := div(len(L),2);
  LHS := first(L,half);
  RHS := last(L,len(L)-half);

  LHSprod := CPL(LHS);
  RHSprod := CPL(RHS);
  ans := [[concat(LHSpart, RHSpart) | RHSpart in RHSprod] | LHSpart in LHSprod];
  return flatten(ans,1);
EndDefine; -- CartesianProductList
```

It generated the list of 32768 lists in less than 0.1s.

Shall I replace the existing impl with this one?
Anna, do you fancy testing it?

#4 - 16 Oct 2019 21:56 - John Abbott

- Status changed from Resolved to Feedback

- Target version changed from CoCoA-5.?.? to CoCoA-5.3.0

- % Done changed from 60 to 90

- Estimated time set to 1.99 h

Apparently I checked this in some time ago -- I suppose not long after the previous comment (5 months ago).

Maybe it could be made still faster, but I doubt it is important to do so.

Changing status to feedback; and target version to 5.3.0 (presumably this impl is also in 5.2.5).

#5 - 22 Oct 2019 11:10 - Anna Maria Bigatti

- Description updated

- Status changed from Feedback to Closed

- % Done changed from 90 to 100