CoCoA-5 - Feature #1262

Split BuiltinFunctions-CoCoALib.C into smaller files

26 Mar 2019 11:27 - John Abbott

Status:	In Progress	**Start date:**	26 Mar 2019
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	30%
Category:	enhancing/improving	**Estimated time:**	0.00 hour
Target version:	CoCoA-5.4.2	**Spent time:**	5.90 hours

Description

Yesterday I tried compiling CoCoA-5 on my small computer (0.5Gbyte RAM), and it had trouble compiling BuiltinFunctions-CoCoALib.C because it has only a small amount of RAM.

We should consider splitting BuiltinFunctions-CoCoALib.C into smaller files.

Perhaps also some other source files? For instance, I note that BuiltinFunctions.C and BuiltinOneLiners-CoCoALib.C produce quite large ".o" files.

History

#1 - 26 Mar 2019 11:39 - John Abbott

I hope this evening to try compiling again on the small computer, and I'll note which files gave problems with swapping.

#2 - 26 Mar 2019 13:53 - Anna Maria Bigatti

- % Done changed from 0 to 10

I had a look.
I think it would make sense to separate off the functions with variable number of args.
(look for DECLARE_ARITYCHECK_FUNCTION)

Maybe it is not an even split, but it is the the same class as the former split between one-liners and other functions.

#3 - 27 Mar 2019 14:21 - John Abbott

- Status changed from New to In Progress

I tried splitting BuiltinFunctions-CoCoALib into two more-or-less-equal parts.

File	Time to compile
BIF-CoCoALib	333
BIF-CoCoALib1	95
BIF-CoCoALib2	123

The gain is less than I had hoped, but still 220s instead of 330s is progress.

The other slow-to-compile file is **BuiltinOneLiners-CoCoALib** which took about 280s.

It would be nice to find a good criterion for splitting the file into 3 or even 4 parts. I should do more experiments to find out what a good size might be...

I also want to update the OS on the small computer; that may change compilation times.

#4 - 27 Mar 2019 14:38 - John Abbott

Other compilation times:

```
BIF-GSL    88s
BIF-Norm   81s
BIF-other  60s about
```

#5 - 27 Mar 2019 14:39 - Anna Maria Bigatti

Do you know how much the overhead for including library.H is? that is quite slow by itself.
(try compiling an examples/ex-XXX)

#6 - 28 Mar 2019 10:58 - John Abbott

- % Done changed from 10 to 20

The time to compile (and link) ex-empty.C was about 20s.

I tried splitting BuiltinFunctions-CoCoALib.C into quarters, but each quarter took about 60s to compile; so splitting into two "halves" appears to be a better compromise.

I have now installed BunsenLabs "Helium" (instead of the old version "Hydrogen").
The compiler is now version 6.3 rather than 4.9, and compilation times have changed.
For instance, it now takes about 150s to compile BuiltinFunction-CoCoALib.C. Perhaps tonight I'll see what happens when I split the file (again).

BTW I noticed that it took quite a long time to compile BuiltinFunctions-Normaliz.C even though Normaliz was absent -- this must surely be wrong!

#7 - 28 Mar 2019 13:30 - John Abbott

I have just checked (using fgrep) and in BuiltinFunctions-CoCoALib.C there are 28 fns with variable arity, and 129 with fixed arity. So if we use this criterion to split the file, the split will be quite uneven (but it might already make an improvement).

#8 - 28 Mar 2019 14:14 - Anna Maria Bigatti

John Abbott wrote:

> I have just checked (using fgrep) and in BuiltinFunctions-CoCoALib.C there are 28 fns with variable arity, and 129 with fixed arity. So if we use this criterion to split the file, the split will be quite uneven (but it might already make an improvement).

True, it is uneven, but it'd make life easier for making new variable-arity functions ;-)

#9 - 28 Mar 2019 14:22 - Anna Maria Bigatti

John Abbott wrote:

> BTW I noticed that it took quite a long time to compile BuiltinFunctions-Normaliz.C even though Normaliz was absent -- this must surely be wrong!

if a library is missing we define the missing functions by

```
DECLARE_MISSING_EXTLIB(NmzXXXXX, "NORMALIZ")
```

this requires including quite a few files, among which library.H (from BuiltinFunctions.H)

#10 - 28 Mar 2019 21:37 - John Abbott

Here are the compilations timings for the updated computer:

```
Compiling AST.o
real    0m54.451s
user    0m42.212s

Compiling Lexer.o
real    0m48.666s
user    0m37.980s

Compiling Interpreter.o
real    3m46.070s
user    2m27.732s

Compiling Parser.o
real    1m2.698s
user    0m59.264s

Compiling BuiltInFunctions.o
real    1m25.513s
user    1m16.468s

Compiling BuiltInFunctions-CoCoALib.o
real    3m9.038s
user    2m6.824s

Compiling BuiltInOneLiners-CoCoALib.o
real    2m8.872s
user    1m44.848s

Compiling BuiltInFunctions-Frobby.o
real    0m53.534s
user    0m35.624s
```

I have arbitrarily excluded those which took less than 40 sec real time.
Anyway, the slow compilations are clear: **Interpreter.C**, **BuiltinFunctions-CoCoALib.C** and **BuiltinOneLiners-CoCoALib.C**

I tried compiling some examples: it takes about 15-20s to compile and link an example (actually ex-empty took longer than ex-matrix1... very odd!)

#11 - 29 Mar 2019 14:16 - John Abbott

I note also that my compilation test above was without any external libraries.
So the time to compile BuiltinFunctions-Frobby.C is the time to make the "empty" version!!

#12 - 29 Mar 2019 14:23 - Anna Maria Bigatti

It seems that many includes in BuiltInFunctions.H are useless, or useful for only a few functions. Cleaning them might help! (experimenting)

#13 - 29 Mar 2019 15:23 - John Abbott

I suggest putting DECLARE_MISSING_EXTLIB into a smaller header file (if possible).

Then files such as BuiltinFunctions-Frobby.C can be something like:

```
#include "CoCoA/PREPROCESSOR_DEFNS.H"

#if !defined(CoCoA_WITH_FROBBY)

#include "MissingExtlib.H"
  DECLARE_MISSING_EXTLIB(FrbDimension, "FROBBY")
  ....

#else

  #include "BuiltinFunctions.H"
  #include "BuiltinOneLiners.H"

  ... usual code ...
#endif
```

This should enable almost instant compilation when CoCoA_WITH_FROBBY is not defined.

#14 - 01 Apr 2019 11:36 - John Abbott

- % Done changed from 20 to 30

Yesterday I ran the compiler with the **-E** option, which just does the preprocessing steps. Since currently I do not have Normaliz enabled, I looked at the output for BuiltinFunctions-Normaliz.C. After preprocessing the file is more than 6Mbytes long -- this is what the compiler proper receives! No wonder the small computer takes so long to process it! Clearly way too much junk is included.

I am very tempted to suggest modifying the macro **DECLARE_MISSING_EXTLIB** so that it is **independent** of the other **DECLARE_XYZ** macros. The disadvantage is that there will be slight duplication of the "internal interface"; but since I do not expect we will ever change this, the duplication is not really a problem (but it must be clearly documented... just in case some does change it).

#15 - 01 Apr 2019 15:27 - Anna Maria Bigatti

John Abbott wrote:

> I am very tempted to suggest modifying the macro **DECLARE_MISSING_EXTLIB** so that it is **independent** of the other **DECLARE_XYZ** macros. The disadvantage is that there will be slight duplication of the "internal interface"; but since I do not expect we will ever change this, the duplication is not really a problem (but it must be clearly documented... just in case some does change it).

I get VERY anxious about duplicating code.
I've noticed, and tested, that many includes in BuiltInFunctions.H are useful only for some file. This is no wonder, as we originally had everything in BuiltinFunctions.C.
I think we should start by cleaning and reorganizing the code before anything else.

As a start, CoCoALibSupplement is no longer included in ExtLib BuiltInFunction-XXX.C files.

#16 - 01 Apr 2019 17:30 - John Abbott

Indeed "duplication" is generally undesirable (as is the use of goto), but it may still be a good idea in certain "unusual" circumstances (as can a goto command).

Removing unnecessary includes is certainly a good idea, but in this case I believe it is a separate issue. The definitions of the functions which report that Normaliz is not present should require (almost?) no includes.

I think that the current implementation is poor because the code itself is misleading: it uses the mechanism for defining a "normal" builtin function, but actually wants to define a "special" function.

#17 - 02 Apr 2019 11:50 - John Abbott

I believe that built-in functions (for extlibs, at least) are "registered" by constructing a global object. So the actual registration is performed by the ctor calls for these objects (the objects themselves do nothing).

An idea is to create a second ctor for these objects which can be called just with the string "Normaliz not present".

The object type is **AddBuiltIn**. Its ctor is in **BuiltinFunctions.H** around line 120.

#18 - 02 Apr 2019 21:52 - John Abbott

Anna has just sent me some code with redundant #include directives removed.

Most compilation times are the same. There were two improvements: Lexer.C compiles now in 40s (previously 48s), and BuiltinFunctions-CoCoALib.C compiles in 150s (previously 190s).

#19 - 11 Mar 2020 21:32 - John Abbott

- Target version changed from CoCoA-5.?.? to CoCoA-5.4.0

#20 - 03 Feb 2022 19:53 - John Abbott

- Target version changed from CoCoA-5.4.0 to CoCoA-5.4.2