CoCoA-5 - Feature #414

New fn to increase max recursion depth

22 Nov 2013 17:59 - John Abbott

Status:	Closed	**Start date:**	22 Nov 2013
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	CoCoA-5 function: new	**Estimated time:**	8.00 hours
Target version:	CoCoA-5.1.0 Easter14	**Spent time:**	7.85 hours

Description

Add a new fn to allow the user to increase the max recursion depth (currently hardwired to 2000)

What should the fn do with "ridiculous" inputs?
What values are considered not "ridiculous"?

History

#1 - 22 Nov 2013 18:07 - John Abbott

Candidates for the name of the new fn:
SetMaxCallDepth
SetCallDepthLimit

It is probably best not to call it something like SetRecursionLimit because it simply counts depth of nested fn calls, and a single recursion level might entail more than a single nested call (I think).

PS edit to add other suggestions

#2 - 22 Nov 2013 18:14 - Anna Maria Bigatti

John Abbott wrote:

> Candidates for the name of the new fn:
> SetMaxCallDepth
> SetCallDepthLimit

I like SetCallDepthLimit

#3 - 06 Dec 2013 22:02 - John Abbott

- Status changed from New to In Progress

- Assignee set to John Abbott

- % Done changed from 0 to 10

The relevant lines in the source code are:

Interpreter.H:879-880
Interpreter.C:1015

There is an unwelcome complication. The identifier MAX_NESTING is defined as a constant global (static) data member, and is used as a compile-time constant when declaring the C-array RuntimeEnvironment::frames. This array cannot be resized.

I hope to replace the static data member by a non-static one (*i.e.* potentially different values in different instances), and must replace the C-array by a C++ vector.

I'll try asking Giovanni why he used a C-array instead of a C++ vector. While I cannot say that I have much knowledge of his code, I don't believe that my proposed changes could cause trouble... [famous last words]

#4 - 06 Dec 2013 22:39 - John Abbott

- % Done changed from 10 to 30

I've just made a "mindless" change to the interpreter code, and amazingly everything compiles and all the tests pass!

Is this too good to be true?

Anyway, now it should be quite easy (famous last words?) to implement a fn which can increase the max call depth. But it's too late to try now.

#5 - 07 Dec 2013 17:46 - John Abbott

Should the capability to alter the max fn call depth be a function or a command?

Will it be possible to **decrease** the max call depth? Could this ever be useful?

If it's a function, what value should it return? The previous max depth?

If it's a command, what syntax should it have? C4 has a Set command for setting various system parameters; it is not present in C5.

Should changing the max call depth be allowed only at top level?
If it is permitted in other circumstances, what should happen if the max call depth is set to a value below the current call depth? (immediate error?)

JAA thinks: allowed only at top level (this can easily be enforced if we use a special command); max depth may be increased or decreased (no technical reason to forbid decreasing it).

Comment? Opinions? Ideas?

#6 - 24 Feb 2014 18:20 - John Abbott

- % Done changed from 30 to 40

Anna said it would be best to keep the change as simple as possible, so it should be a function (since we're not sure how easy it might be to add a new command).

The manual should probably contain a warning not to use the fn unless you know what you're doing. Probably the fn name should be "ugly" too :-)

#7 - 24 Feb 2014 22:42 - John Abbott

I have a first impl, but it SEGVs if I increase the stack size; don't know why.
The CoCoA-5 input is this: (initial stack size is set to 20)

```
Define SumFact(N) if N=0 then return 0; else return N+SumFact(N-1); EndIf; EndDefine;
SetStackSize(50); --> might cause the vector to reallocate
SumFact(21);
```

Giovanni can you give me a hand here?

I notice that the frames in the stack contain raw pointers to other frames; why?
These raw pointers are problematic if the std::vector reallocates; this is probably the cause of the SEGV, but it's not clear who's trying to follow these dangling pointers.

#8 - 25 Feb 2014 10:18 - Giovanni Lagorio

It's complicated; short answer: a frame needs to "point/reference" other frames for accessing non-local variables (*e.g.* in a lambda).
Now, using a raw pointer is (supposedly) more efficient than keeping an index, which has to be transformed into a pointer later (by adding the current begin of the stack with the index). Obviously, keeping a raw pointer means that frames cannot be moved around. It was an array, and not a vector, for a reason ;-)
I don't remember the details, so I'm not sure whether this is the only place where the code assumes that frames do not move. I suggest to revert your changes and raise the constant if needed (actually, you could probably dynamically allocate the array once, so you could use a command line option to set the limit, but the limit must be set once for all for each run of the interpreter)

#9 - 25 Feb 2014 10:40 - John Abbott

I had suspected as much. Some guys wanted a stack size of about 250000, and asked for a command to increase stack size.

The command line option would surely be the simplest solution; I wonder if it would be enough -- I need to think about it (on the whole I don't much like command line options).

It still surprised me that I got a SEGV when the first command I executed was SetStackSize (so there should not have been any need to refer to non-local variables).

#10 - 26 Feb 2014 11:46 - John Abbott

Here is a reason why I prefer a built-in fn for increasing max stack height over a command line option: a user discovers part way through a long session (with several costly computations) that one fn call fails because of insufficient stack space; with a command line option the user has to discard all computations done so far (since CoCoA-5 does not have a "save session" facility), and start a whole new session.

If I recall correctly, we imposed a limit on the stack height to protect the careless/inexpert user from his own mistakes (*e.g.* writing an infinitely recursive fn). It is not good that our solution impedes expert users trying to perform difficult computations.

Now I feel more "justified" in my preference for a built-in fn, but that doesn't mean that it is practicable...

#11 - 25 Mar 2014 17:21 - John Abbott

- Target version changed from CoCoA-5.0.9 to CoCoA-5.1.0 Easter14

This is going to be a pain to sort out -- delaying until 5.1.0.

Probably command line option is simplest "quick fix".

#12 - 17 Apr 2014 01:03 - John Abbott

Do the "quick fix"! Not satisfactory, but a proper soln is going to be costly :-(

#13 - 17 Apr 2014 18:46 - John Abbott

- Estimated time set to 3.00 h

#14 - 17 Apr 2014 18:46 - John Abbott

- Estimated time changed from 3.00 h to 8.00 h

#15 - 17 Apr 2014 20:10 - John Abbott

- % Done changed from 40 to 60

I've implemented a "quick fix":

- ctor RuntimeEnvironment now has extra arg for specifying MaxStackSize
- ctor for Interpreter now has extra arg for specifying MaxStackSize
- changed main (in @Main.C:153,175-183,265) to accept command line arg
- changed C5.C:932 to pass extra arg (constant=10000) to ctor for Interpreter

All C5 tests pass!

I suppose we ought to allow Emacs UI to have option for setting stack size.

#16 - 17 Apr 2014 20:18 - John Abbott

Works OK up to stack size of about 5400; at 5500 I get a SEGV. Why????

Also get SEGV without --stacksize option; I have increased default max stack size to 5000 (because higher than that causes problems).

Makes no sense :-(
Aaaaahhh it seems that the C++ stack overflowed -- seems to need almost 3kbytes per CoCoA-5 stackframe (in a simple test on my machine). Why so much?

#17 - 22 Apr 2014 13:50 - John Abbott

- Status changed from In Progress to Resolved

- % Done changed from 60 to 70

Here is the very simple test case I used:

```
Define iter(n) If n=0 Then Return n EndIf; Return 1+iter(n-1); EndDefine;
For i:=1 To 12 do
  iter(10*2^i);
EndFor;
```

Changing state to resolved; I think it will be hard to do much better than this without making a major change to the interpreter (*i.e.* avoiding using so much C++ stack space).

#18 - 02 May 2014 16:53 - John Abbott

- Status changed from Resolved to Closed

- % Done changed from 70 to 100

A proper solution will be very costly and highly invasive (and Giovanni's not keen to help). Furthermore, it is probably better that programs which push the limits be written in C++ using CoCoALib rather than CoCoA-5 (with interpreter overhead).

The current implementation is a reasonable compromise (considering our very limited resources); so closing.

#19 - 02 May 2014 17:05 - John Abbott

Adding note about original requester: andrei_zarojanu (at yahoo.com)
I'll email him about our partial solution -- hope it'll be enough for him!