Making timelimit with CAS that does not support it

Nasser M. Abbasi

June 28, 2025 Compiled on June 28, 2025 at 9:54am

Contents

1 Example using Reduce **2**

2 Example using Sympy **4**

I run some tests using different CAS systems that do not support setting a timelimit on its calls.

For example Reduce, Maxima, Sympy, Mupad and GIAC and Sagemath (which is used to call Fricas, Maxima and GIAC) also have no built-in timelimit command.

They do not have a direct way to insure a call to some command such as integrate should not take more than say 30 seconds and to return an error if the call exceeds this timelimit.

CAS systems such as Maple, Fricas and Mathematica have built-in command for this. This document is meant to describe how to add timelimit to the other systems that do not support this.

This method only works and have been tested on Linux.

After much trials and errors, I settled down to the following method to add timelimit. It is best described using an example. But this is the idea:

A driver script in Python is used, which creates for each command a small script which has the other CAS system commands in it.

Then Python issues `subprocess.Popen( the_command , shell=True)` to run the command. The command has this form `timeout 5 other_cas_name < "' + script_name + '"'` where `script_name` is the name of the script just created. In the above `other_cas_name` is the name of the other CAS, such as Reduce, giac, maxima and so on.

Next the python script will wait using `p.wait()`. When this wait call returns, it will check if an output file was created from running the command or not.

If no such file is found, an exception is generated which means timeout. If the output file was created then this means the other CAS completed in time and the output of the command will be inside the output file.

Lets say we want to integrate $\int \sin x \, dx$ and set 30 seconds time limit and find out if the call timed out or not.

A python script is used as the driver. The Python script creates a file where the command is written to it, then it uses

```
subprocess.Popen( the_command , shell=True)
p.wait()
```

Where in the above the command has `timeout 30` in its start.

This Linux command invokes the CAS system passing it the name of the file to process. The file has in it all the commands for the other CAS system and where to write the result of the

integration to.

To find if the command completed in time on not, the Python script then checks if the output file from the command exists or not after the call to `p.wait()` returns.

This output file contains the result or output of the command that CAS should have written the result to. This file is deleted each time. So if the file does not exist, it means CAS has timed out.

Lets make a fully working example to show all the details. For illustration, the following are examples using different CAS systems.

1 Example using Reduce

Create this python file, calling it say `reduce_command.py` (need to change to the folder where this file is saved to).

```
import subprocess
import os, sys, datetime, ntpath, time

if __name__ == "__main__":

   root_tree =  "/mnt/g/REDUCE"
   os.chdir(root_tree)
   os.getcwd()

   reduce_script_name = './reduce_script.red'
   the_command = 'rm -f "' + reduce_script_name +  '"'
   os.system(the_command )

   reduce_TMP_file = root_tree + "/" + 'TMP_listA.txt'
   the_command = 'rm -f "' + reduce_TMP_file +  '"'
   os.system(the_command)

   result_file = open(reduce_script_name, "w")
   result_file.write("load_package rlfi,algint;\n")
   result_file.write("off nat;\n")
   result_file.write("linelength 10000000;\n")
   result_file.write('sep := ","$\n')
   result_file.write("anti:=int(sin(x),x)$\n")
   result_file.write('out "'+ reduce_TMP_file + '";\n')
   result_file.write('write """, anti, """;\n')
   result_file.write('shut "'+ reduce_TMP_file + '";\n')
   result_file.close()

   starting_time =  time.time()
   the_command =   'timeout 30 redcsl < "' + reduce_script_name + '"'
   p = subprocess.Popen( the_command , shell=True)
   p.wait()
   time_used  = time.time() - starting_time

   try:
      with open(reduce_TMP_file) as xxx:
          reduce_output = xxx.readlines()

      print("completed with no time out. This is the result ",reduce_output[1])
   except Exception as ee:
         print("Exception in call to queue.get. TIME OUT ")
         print("type(exception).__name__=",type(ee).__name__)
```

```
        print("looks like time out!")

  print("done....")
```

Now type this command to run the above Python script

```
python ./reduce_command.py
```

It will print

```
>python ./reduce_command.py
Reduce (CSL, rev 6687),  9-Jan-2024 ...

1:
2:
3:
t$

4:
5:
6: 7: 8:
9:

*** End-of-file read
completed with no time out. This is the result  " - cos(x)"$

done....
>
```

Lets now try different integral which is known to hang Reduce. We will see it times out after 30 seconds as expected.

Changing the line `result_file.write("anti:=int(sin(x),x)$\n")` in the above python file with `result_file.write("int(sec(x)^3/(a+b*cos(x)+c*cos(x)^2),x)$\n")` and re-running the Python script gives

```
>time python ./reduce_command.py
Reduce (CSL, rev 6687),  9-Jan-2024 ...

1:
2:
3:
t$

4:
5:
~~~ SIGNAL ~~~: SIGTERM
Exception in call to queue.get. TIME OUT
type(exception).__name__= FileNotFoundError
looks like time out!
done....

real    0m30.087s
user    0m29.738s
sys     0m2.297s
>
```

We see from the above that the script did detect timeout has occured and that Reduce was allowed only 30 seconds to run based on the output from the `time` command used.

If the above integrate command was run directly in Reduce, it would have hanged or taken very long time.

2 Example using Sympy

This section shows how to do the same thing with sympy.

Create this python file, calling it say `sympy_command.py`

```
import subprocess
import os, sys, datetime, ntpath, time

if __name__ == "__main__":

   root_tree =  "/mnt/g/SYMPY"
   os.chdir(root_tree)
   os.getcwd()

   sympy_script_name = './sympy_script.py'
   the_command = 'rm -f "' + sympy_script_name +  '"'
   os.system(the_command )

   sympy_TMP_file = root_tree + "/" + 'TMP_listA.txt'
   the_command = 'rm -f "' + sympy_TMP_file +  '"'
   os.system(the_command)

   result_file = open(sympy_script_name, "w")
   result_file.write("from sympy import *\n")
   result_file.write("x,a,b,c = symbols('x a b c')\n")
   result_file.write("anti=integrate(sec(x)**3/(a+b*cos(x)+c*cos(x)**2),x)\n")
#make sure to do this after the above command and not before it.
   result_file.write("f=open(\""+sympy_TMP_file+ "\",\"w\")\n")
   result_file.write("f.write(str(anti)+\"\\n\")\n")
   result_file.write("f.close()\n")
   result_file.close()

   starting_time =  time.time()
   the_command =   'timeout 5 python < "' + sympy_script_name + '"'
   p = subprocess.Popen( the_command , shell=True)
   p.wait()
   time_used  = time.time() - starting_time

   try:
      with open(sympy_TMP_file) as xxx:
          sympy_output = xxx.readlines()

      print("completed with no time out. This is the result ",sympy_output)
   except Exception as ee:
         print("Exception in call to queue.get. TIME OUT ")
         print("type(exception).__name__=",type(ee).__name__)
         print("looks like time out!")

   print("done....")
```

In the above, we gave sympy only 5 seconds. The result is this

```
>python ./sympy_command.py
Exception in call to queue.get. TIME OUT
```

```
type(exception).__name__= FileNotFoundError
looks like time out!
done....
>
```

We see, it timed out as 5 second was too short.

Changing the timeout to 50 seconds by changing the line `the_command = 'timeout 5 python < "' + s` to `the_command = 'timeout 50 python < "' + sympy_script_name + '"'` and re-running the command gives

```
>time python ./sympy_command.py
completed with no time out. This is the result  ['Integral(sec(x)**3/(a + b*cos(x
    ) + c*cos(x)**2), x)\n']
done....

real    0m36.509s
user    0m35.915s
sys     0m0.069s
>
```

We see that now it did not time out (the command took 36 seconds). But sympy did not evaluate the integral.

These examples show how to add timelimit to CAS systems which does not support built-in commands to do this.

For each different CAS a custom script needs to be written. Will add more examples for different CAS systems later.