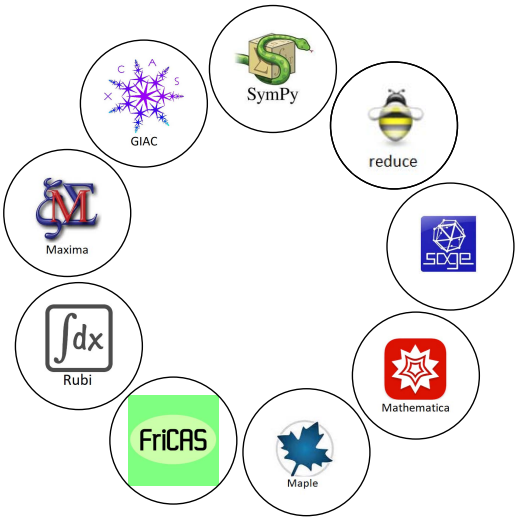
Computer Algebra Independent Integration Tests

January 2024 special edition with Reduce CAS



Nasser M. Abbasi

January 2024

Contents

CHAPTER 1

Introduction

This report gives the result of running the computer algebra independent integration problems.

The listing of the problems used by this report are

1. MIT_bee_integration_problems.zip
2. handbook_integration_problems.zip
3. CAS_integration_tests_2023_Mathematica_format.m
4. CAS_integration_tests_2023_Maple_and_Mupad_format.zip
5. CAS_integration_tests_2023_SAGE_format.zip
6. CAS_integration_tests_2023_Sympy_format.zip

The Mathematica/Rubi format file above can be read into Mathematica using the following commands

```
SetDirectory[NotebookDirectory[]] (*where the above .m file was save*)
lst=First@ReadList["CAS_integration_tests_2023_Mathematica_format.m",Expression];
Length[lst]
```

`lst[[1]]` will be the first integrand,var and `lst[[2]]` will be the second one and so on.

The Rubi test suite files were downloaded from rulebasedintegration.org.

The current number of problems in this test suite is [3809].

1.1 Listing of CAS systems tested

The following are the CAS systems tested:

1. Mathematica 13.3.1 (August 16, 2023) on windows 10.
2. Rubi 4.17.3 (Sept 25, 2023) on Mathematica 13.3.1 on windows 10
3. Maple 2023.1 (July, 12, 2023) on windows 10.
4. Maxima 5.47 (June 1, 2023) using Lisp SBCL 2.3.0 on Linux via sagemath 10.1 (Aug 20, 2023).
5. FriCAS 1.3.9 (July 8, 2023) based on sbcl 2.3.0 on Linux via sagemath 10.1 (Aug 20, 2023).
6. Giac/Xcas 1.9.0-57 (June 26, 2023) on Linux via sagemath 10.1 (Aug 20, 2023).

7. Sympy 1.12 (May 10, 2023) Using Python 3.11.3 on Linux.
8. Mupad using Matlab 2021a with Symbolic Math Toolbox Version 8.7 on windows 10.
9. Reduce CSL rev. 6657. December 10, 2023. On Linux.

Maxima and Fricas and Giac are called using Sagemath. This was done using Sagemath `integrate` command by changing the name of the algorithm to use the different CAS systems.

Sympy was run directly in Python not via sagemath.

Reduce was run directly.

1.2 Results

Important note: A number of problems in this test suite have no antiderivative in closed form. This means the antiderivative of these integrals can not be expressed in terms of elementary, special functions or `Hypergeometric2F1` functions.

If a CAS returns the above integral unevaluated within the time limit, then the result is counted as passed and assigned an A grade.

However, if CAS times out, then it is assigned an F grade even if the integral is not integrable, as this implies CAS could not determine that the integral is not integrable in the time limit.

If a CAS returns an antiderivative to such an integral, it is assigned an A grade automatically and this special result is listed in the introduction section of each individual test report to make it easy to identify as this can be important result to investigate.

The results given in in the table below reflects the above.

Table 1.1: Percentage solved for each CAS

System	solved	Failed
Mathematica	% 99.606 (3794)	% 0.394 (15)
Rubi	% 99.396 (3786)	% 0.604 (23)
Fricas	% 89.525 (3410)	% 10.475 (399)
Maple	% 87.897 (3348)	% 12.103 (461)
Giac	% 79.312 (3021)	% 20.688 (788)
Maxima	% 75.663 (2882)	% 24.337 (927)
Mupad	% 73.326 (2793)	% 26.674 (1016)
Reduce	% 72.828 (2774)	% 27.172 (1035)
Sympy	% 69.782 (2658)	% 30.218 (1151)

The table below gives additional break down of the grading of quality of the antiderivatives generated by each CAS. The grading is given using the letters A,B,C and F with A being the best quality. The grading is accomplished by comparing the antiderivative generated with the optimal antiderivatives included in the test suite. The following table describes the meaning of these grades.

Table 1.2: Description of grading applied to integration result

grade	description
A	Integral was solved and antiderivative is optimal in quality and leaf size.
B	Integral was solved and antiderivative is optimal in quality but leaf size is larger than twice the optimal antiderivatives leaf size.
C	Integral was solved and antiderivative is non-optimal in quality. This can be due to one or more of the following reasons 1. antiderivative contains a hypergeometric function and the optimal antiderivative does not. 2. antiderivative contains a special function and the optimal antiderivative does not. 3. antiderivative contains the imaginary unit and the optimal antiderivative does not.
F	Integral was not solved. Either the integral was returned unevaluated within the time limit, or it timed out, or CAS hanged or crashed or an exception was raised.

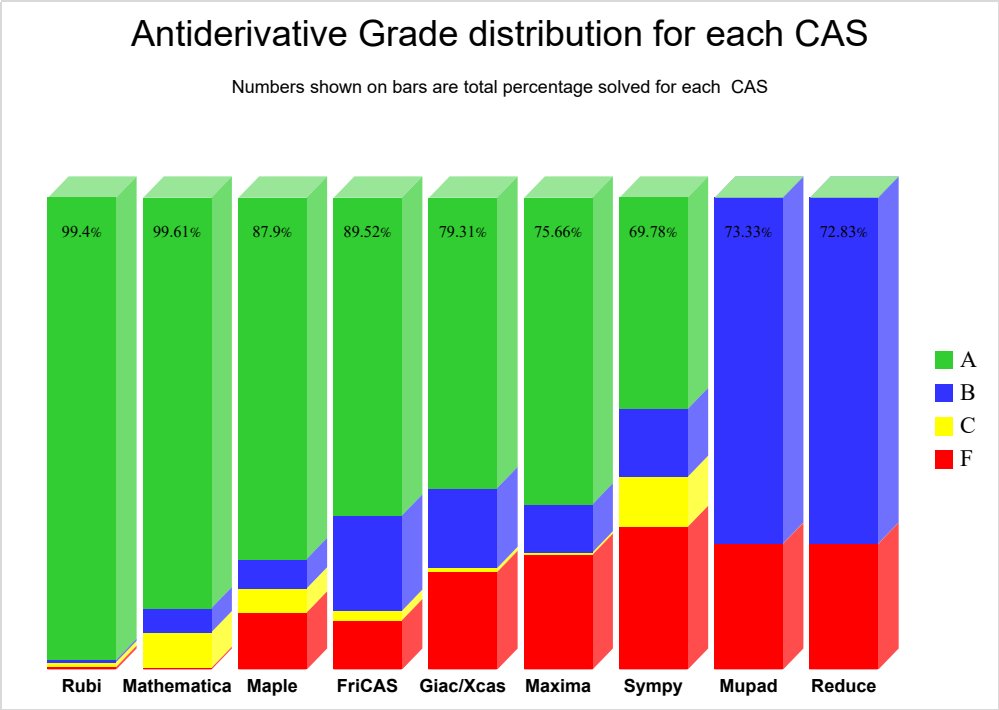
Grading is implemented for all CAS systems in this version except for CAS Mupad where a grade of B is automatically assigned as a place holder for all integrals it completes on time.

The following table summarizes the grading results.

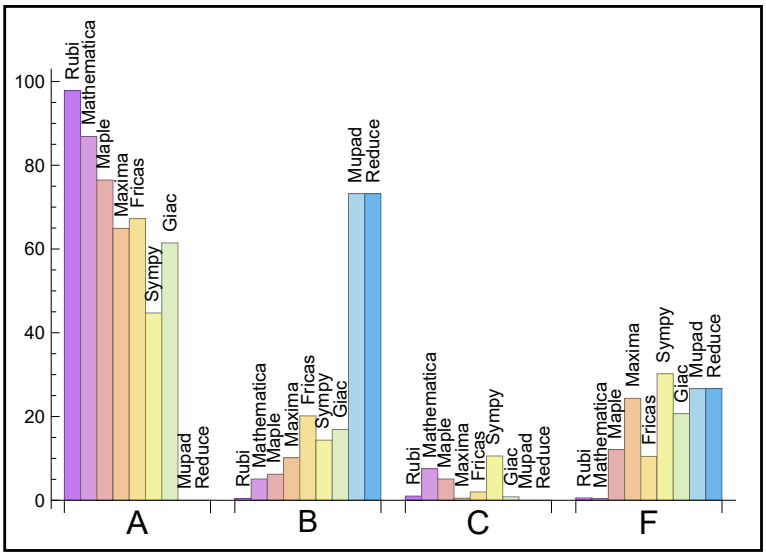
Table 1.3: Antiderivative Grade distribution for each CAS

System	% A grade	% B grade	% C grade	% F grade
Rubi	97.82	0.45	1.02	0.6
Mathematica	86.87	5.09	7.53	0.39
Maple	76.48	6.22	5.09	12.1
Fricas	67.26	20.16	2.	10.48
Maxima	64.93	10.16	0.5	24.34
Giac	61.46	16.91	0.84	20.69
Sympy	44.74	14.36	10.58	30.22
Reduce	N/A	73.22	0.	26.67
Mupad	N/A	73.22	0.	26.67

The following Bar chart is an illustration of the data in the above table.



The figure below compares the CAS systems for each grade level.



1.3 Time and leaf size Performance

The table below summarizes the performance of each CAS system in terms of time used and leaf size of results.

Mean size is the average leaf size produced by the CAS (before any normalization). The Normalized mean is relative to the mean size of the optimal anti-derivative given in the input files.

For example, if CAS has **Normalized mean** of 3, then the mean size of its leaf size is 3 times as large as the mean size of the optimal leaf size.

Median size is value of leaf size where half the values are larger than this and half are smaller (before any normalization). i.e. The Middle value.

Similarly the **Normalized median** is relative to the median leaf size of the optimal.

For example, if a CAS has Normalized median of 1.2, then its median is 1.2 as large as the median leaf size of the optimal.

Table 1.4: Time and leaf size performance for each CAS

System	Mean time (sec)	Mean size	Normalized mean	Median size	Normalized median
Reduce	0.09	112.42	1.53	36.	1.
Rubi	0.21	71.88	1.	43.	1.
Maxima	0.23	76.7	1.38	32.	0.9
Mupad	0.27	76.27	1.27	28	0.87
Mathematica	0.29	66.49	1.1	39.	1.
Fricas	0.36	156.66	1.83	41.	1.12
Maple	0.5	114.53	1.3	30.	0.87
Giac	1.9	102.39	1.6	35.	0.99
Sympy	3.2	209.21	6.4	37.	1.06

1.4 Performance based on number of rules Rubi used

This section shows how each CAS performed based on the number of rules Rubi needed to solve the same integral. One diagram is given for each CAS.

On the y axis is the percentage solved which Rubi itself needed the number of rules given the x axis. These plots show that as more rules are needed then most CAS system percentage of solving decreases which indicates the integral is becoming more complicated to solve.

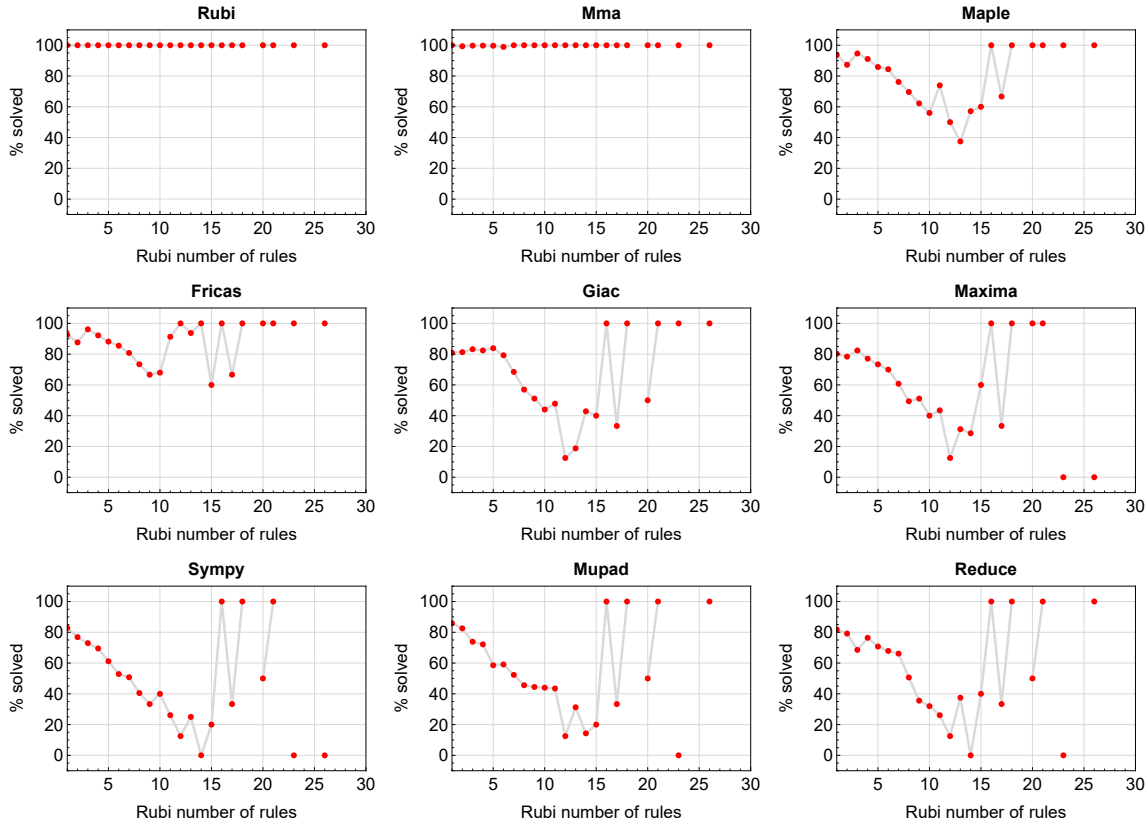


Figure 1.1: Solving statistics per number of Rubi rules used

1.5 Performance based on number of steps Rubi used

This section shows how each CAS performed based on the number of steps Rubi needed to solve the same integral. Note that the number of steps Rubi needed can be much higher than the number of rules, as the same rule could be used more than once.

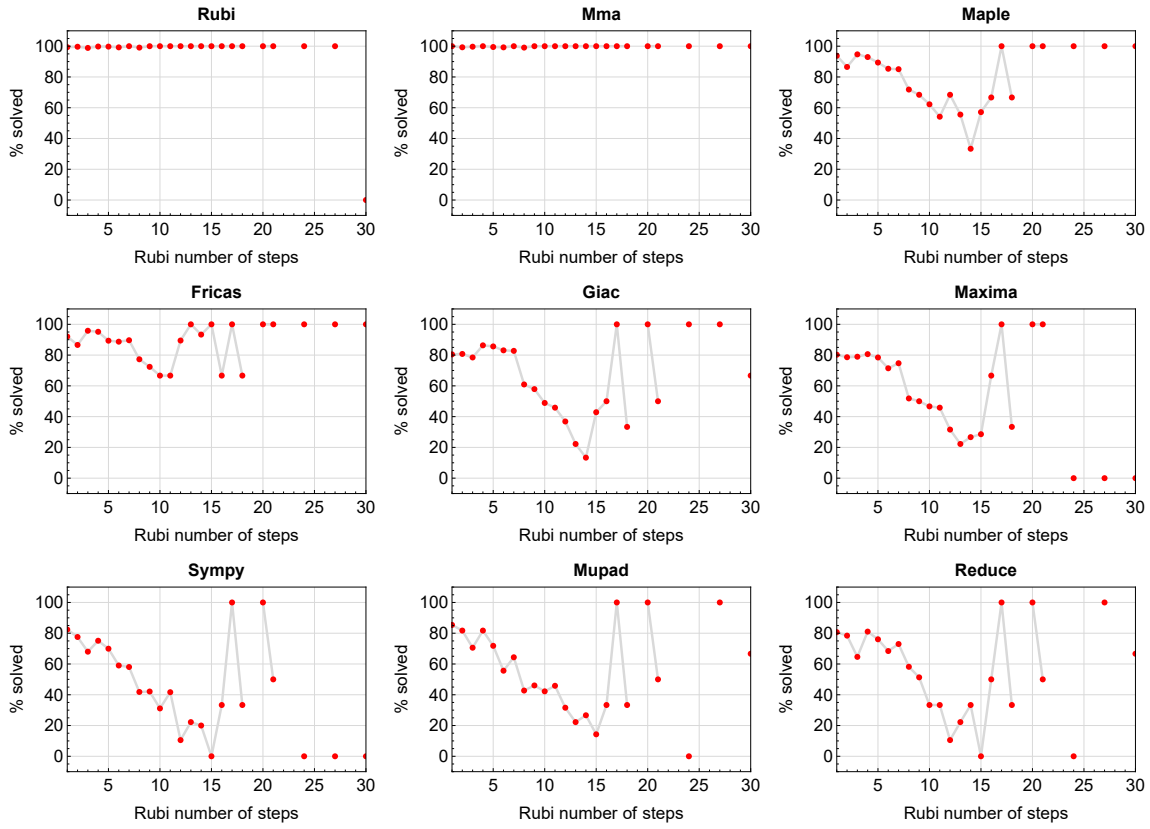


Figure 1.2: Solving statistics per number of Rubi steps used

The above diagram show that the precentage of solved intergals decreases for most CAS systems as the number of steps increases. As expected, for integrals that required less steps by Rubi, CAS systems had more success which indicates the integral was not as hard to solve. As Rubi needed more steps to solve the integral, the solved percentage decreased for most CAS systems which indicates the integral is becoming harder to solve.

1.6 Solved integrals histogram based on leaf size of result

The following shows the distribution of solved integrals for each CAS system based on leaf size of the antiderivatives produced by each CAS. It shows that most integrals solved produced leaf size less than about 100 to 150. The bin size used is 40.

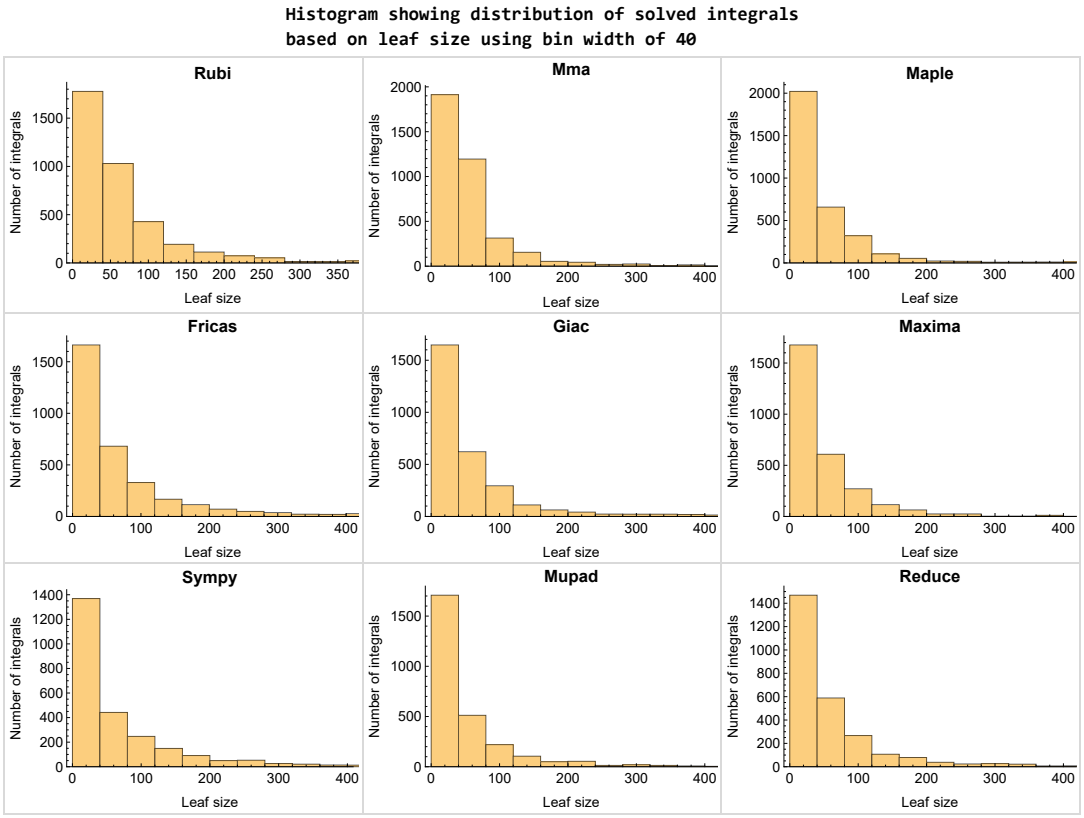


Figure 1.3: Solved integrals based on leaf size distribution

1.7 Solved integrals histogram based on CPU time used

The following shows the distribution of solved integrals for each CAS system based on CPU time used in seconds. The bin size used is 0.1 second.

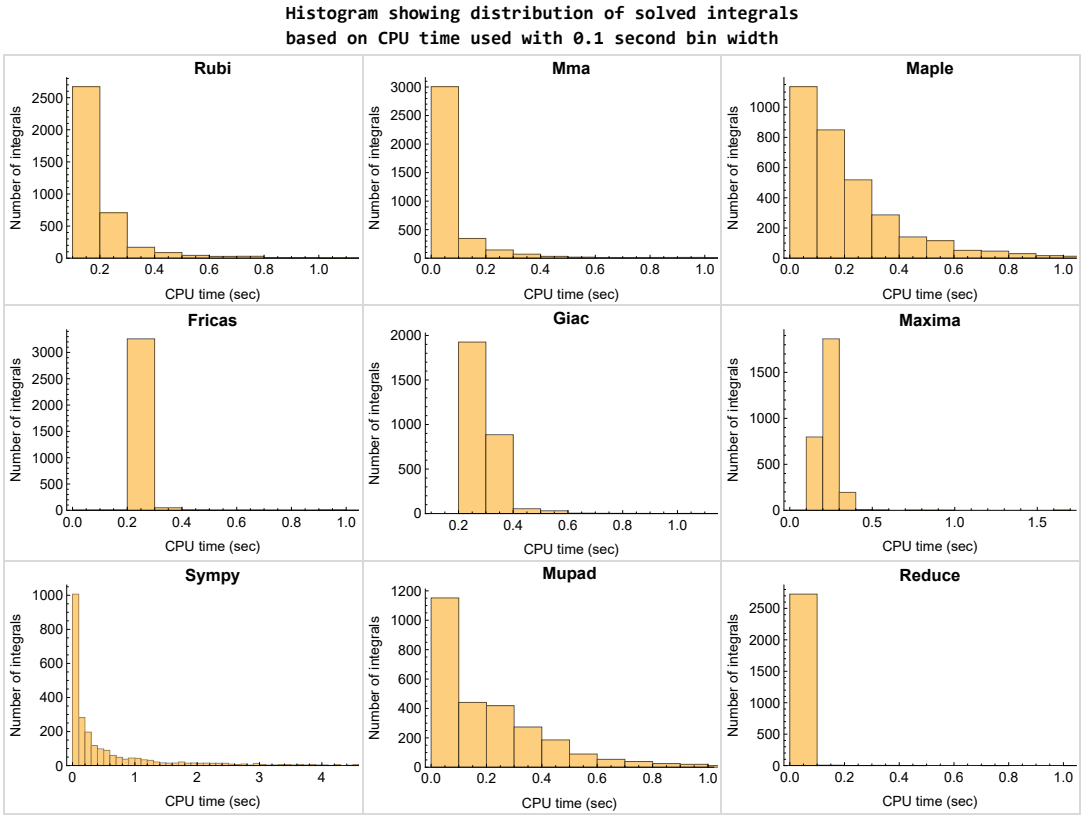


Figure 1.4: Solved integrals histogram based on CPU time used

1.8 Leaf size vs. CPU time used

The following gives the relation between the CPU time used to solve an integral and the leaf size of the antiderivative.

The result for Fricas, Maxima and Giac is shifted more to the right than the other CAS system due to the use of sagemath to call them, which causes an initial slight delay in the timing to start the integration due to overhead of starting a new process each time.

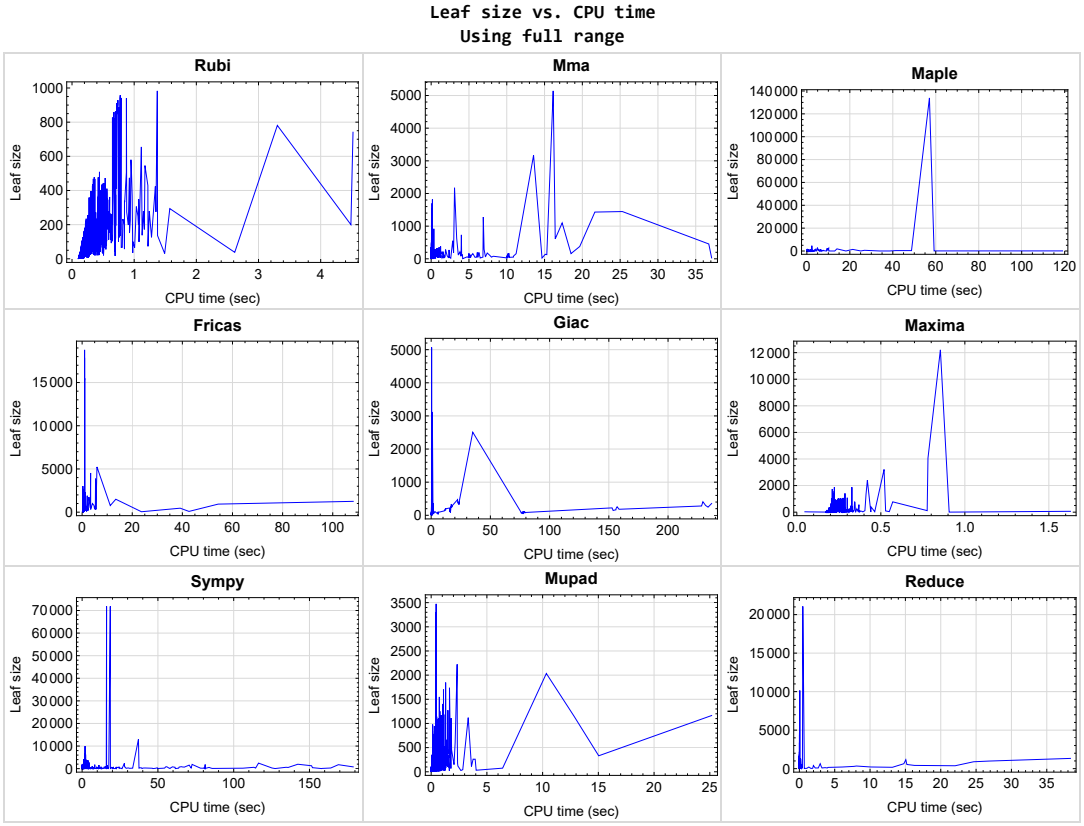


Figure 1.5: Leaf size vs. CPU time. Full range

1.9 Performance per integrand type

The following are the different integrand types the test suite contains.

1. Independent tests.
2. Algebraic Binomial problems (products involving powers of binomials and monomials).

3. Algebraic Trinomial problems (products involving powers of trinomials, binomials and monomials).
4. Miscellaneous Algebraic functions.
5. Exponentials.
6. Logarithms.
7. Trigonometric.
8. Inverse Trigonometric.
9. Hyperbolic functions.
10. Inverse Hyperbolic functions.
11. Special functions.
12. Sam Blake input file.
13. Waldek Hebisch input file.
14. MIT Bee integration.
15. Few problems from Ryzhik and Gradshteyn table of integrals handbook.

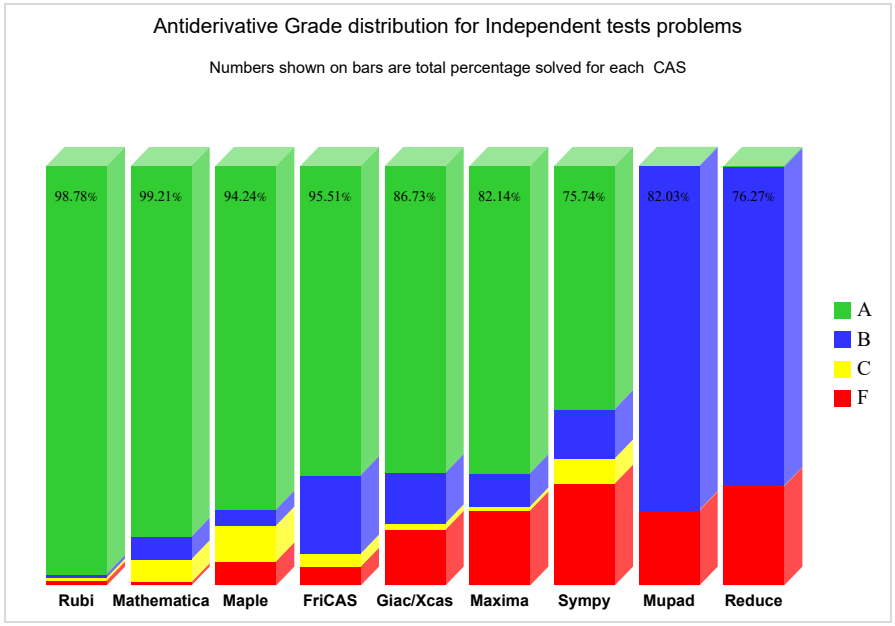
The following table gives percentage solved of each CAS per integrand type.

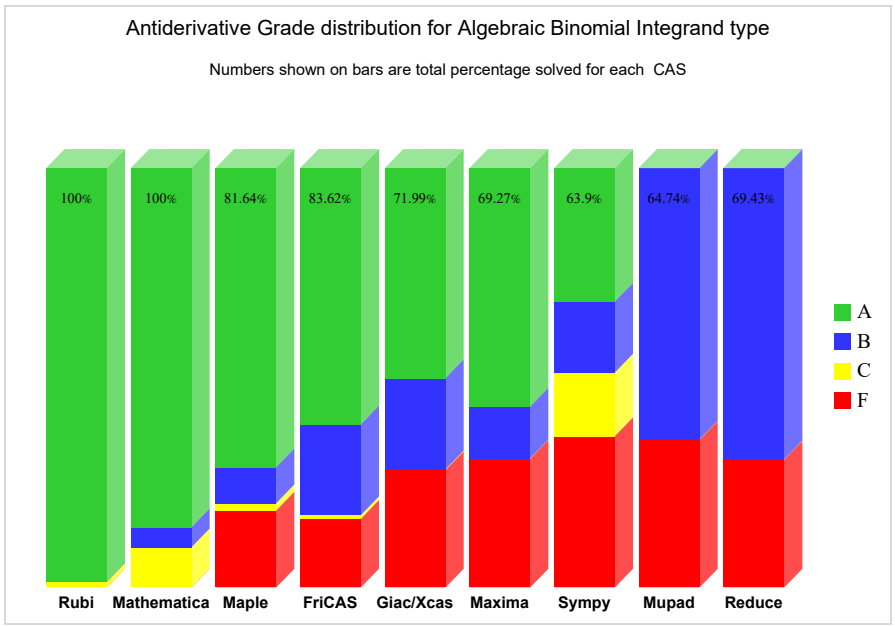
Table 1.5: Percentage solved per integrand type

Integrand type	problems	Rubi	Mathematica	Maple	Maxima	Fricas	Sympy	Giac	Reduce	Mupad
Independent tests	1892	98.78	99.21	94.24	82.14	95.51	75.74	86.73	76.27	82.03
Algebraic Binomial	1917	100.	100.	81.64	69.27	83.62	63.9	71.99	69.43	64.74
Algebraic Trinomial	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Algebraic Miscellaneous	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Exponentials	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Logarithms	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Trigonometric	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Inverse Trigonometric	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Hyperbolic	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Inverse Hyperbolic	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Special functions	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Sam Blake file	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Waldek Hebisch file	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
MIT Bee integration	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Table of integrals	0	0.	0.	0.	0.	0.	0.	0.	0.	0.

In addition to the above table, for each type of integrand listed above, 3D chart is made which shows how each CAS performed on that specific integrand type.

These charts and the table above can be used to show where each CAS relative strength or weakness in the area of integration.


Antiderivative Grade distribution for Independent tests problems
Numbers shown on bars are total percentage solved for each CAS
98.78%
99.21%
94.24%
95.51%
86.73%
82.14%
75.74%
82.03%
76.27%
A
B
C
F
Rubi
Mathematica
Maple
FriCAS
Giac/Xcas
Maxima
Sympy
Mupad
Reduce


Antiderivative Grade distribution for Algebraic Binomial Integrand type
Numbers shown on bars are total percentage solved for each CAS
100%
100%
81.64%
83.62%
71.99%
69.27%
63.9%
64.74%
69.43%
A
B
C
F
Rubi
Mathematica
Maple
FriCAS
Giac/Xcas
Maxima
Sympy
Mupad
Reduce

1.10 Maximum leaf size ratio for each CAS against the optimal result

The following table gives the largest ratio found in each test file, between each CAS antiderivative and the optimal antiderivative.

For each test input file, the problem with the largest ratio $\frac{\text{CAS leaf size}}{\text{Optimal leaf size}}$ is recorded with the corresponding problem number.

In each column in the table below, the first number is the maximum leaf size ratio, and the number that follows inside the parentheses is the problem number in that specific file where this maximum ratio was found. This ratio is determined only when CAS solved the the problem and also when an optimal antiderivative is known.

If it happens that a CAS was not able to solve all the integrals in the input test file, or if it was not possible to obtain leaf size for the CAS result for all the problems in the file, then a zero is used for the ratio and -1 is used for the problem number.

This makes it easier to locate the problem. In the future, a direct link will be added as well.

Table 1.6: Maximum leaf size ratio for each CAS against the optimal result

#	Rubi	Mathematica	Maple	Maxima	FriCAS	Sympy	Giac	Mupad	Redu
1	1.8 (133)	3.9 (50)	4.5 (170)	3.8 (169)	4. (45)	4789.3 (145)	4.2 (164)	0. (-1)	5.7 (
2	1.9 (26)	5. (26)	3.6 (17)	113.1 (21)	14.3 (13)	16.8 (5)	4.6 (2)	0. (-1)	1.8 (
3	1.1 (4)	2. (7)	2. (6)	11.1 (7)	2. (8)	1.9 (5)	1.9 (5)	0. (-1)	2.4 (
4	6.8 (5)	14.3 (13)	11.7 (8)	29.7 (8)	5.5 (43)	4.8 (40)	5.3 (1)	0. (-1)	3.4 (
5	2. (225)	54.7 (278)	11.9 (280)	8.1 (280)	7.7 (280)	39.8 (123)	19.5 (141)	0. (-1)	8.4 (
6	1. (1)	1.4 (3)	2.2 (4)	1.9 (1)	1.4 (7)	0.8 (4)	2.3 (5)	0. (-1)	1.5 (
7	2.2 (3)	5.6 (7)	1.8 (3)	2.8 (3)	6.7 (9)	45.4 (9)	1.9 (3)	0. (-1)	1.8 (
8	2.9 (70)	5.3 (31)	4.5 (57)	6.5 (11)	5. (42)	26.4 (71)	5.8 (40)	0. (-1)	7.2 (
9	2.2 (112)	6.8 (316)	3.5 (323)	12.1 (328)	4.2 (341)	4789.3 (251)	15. (328)	0. (-1)	5.7 (
10	4. (604)	10.9 (446)	367.9 (417)	36.9 (399)	93.4 (137)	124.9 (217)	18.8 (537)	0. (-1)	9.9 (

Continued on next page

Table 1.6 – continued from previous page

file #	Rubi	Mathematica	Maple	Maxima	FriCAS	Sympy	Giac	Mupad	Redu
11	7.7 (82)	2.8 (24)	24.7 (55)	2.7 (2)	14.9 (77)	43. (17)	6.6 (50)	0. (-1)	19.9
12	1.8 (6)	2.3 (4)	1.2 (8)	1.5 (2)	3.3 (3)	3.4 (3)	1.6 (2)	0. (-1)	1.8 (
13	7.1 (369)	23.8 (1323)	27.7 (1323)	32.9 (1323)	32.9 (1323)	136.1 (671)	34. (1323)	0. (-1)	34.6 (132

1.11 Pass/Fail per test file for each CAS system

The following table gives the number of passed integrals and number of failed integrals per test number. There are 210 tests. Each tests corresponds to one input file.

Table 1.7: Pass/Fail per test file for each CAS

#	Rubi		MMA		Maple		Maxima		FriCAS		Sympy		Giac		Mupad		
	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pa
1	175	0	175	0	173	2	166	9	174	1	165	10	170	5	169	6	16
2	33	2	34	1	28	7	16	19	25	10	9	26	17	18	9	26	11
3	13	1	14	0	12	2	8	6	13	1	9	5	10	4	11	3	8
4	48	2	50	0	33	17	26	24	50	0	19	31	41	9	12	38	23
5	279	5	284	0	282	2	251	33	281	3	254	30	269	15	270	14	26
6	3	4	7	0	5	2	3	4	7	0	5	2	5	2	7	0	4
7	7	2	9	0	9	0	7	2	9	0	5	4	9	0	9	0	6
8	113	0	113	0	113	0	111	2	112	1	107	6	111	2	106	7	10
9	376	0	376	0	376	0	374	2	376	0	363	13	375	1	372	4	35
10	704	1	705	0	656	49	565	140	662	43	460	245	590	115	542	163	46
11	110	6	102	14	88	28	20	96	90	26	29	87	36	80	37	79	27
12	8	0	8	0	8	0	7	1	8	0	8	0	8	0	8	0	8
13	1917	0	1917	0	1565	352	1328	589	1603	314	1225	692	1380	537	1241	676	13

1.12 Timing

The command `AbsoluteTiming[]` was used in Mathematica to obtain the elapsed time for each integrate call. In Maple, the command `Usage` was used as in the following example

```
cpu_time := Usage(assign ('result_of_int',int(expr,x)),output='realtime'
```

For all other CAS systems, the elapsed time to complete each integral was found by taking the difference between the time after the call completed from the time before the call was made. This was done using Python's `time.time()` call.

All elapsed times shown are in seconds. A time limit of 3 CPU minutes was used for each integral. If the integrate command did not complete within this time limit, the integral was aborted and considered to have failed and assigned an F grade. The time used by failed integrals due to time out was not counted in the final statistics.

For Reduce CAS, since it has no support for timelimit, there was no time limit used. But the time used was still recorded.

1.13 Verification

A verification phase was applied on the result of integration for `Rubi` and `Mathematica` and `Maple`.

Future version of this report will implement verification for the other CAS systems. For the integrals whose result was not run through a verification phase, it is assumed that the antiderivative was correct.

Verification phase also had 3 minutes time out. An integral whose result was not verified could still be correct, but further investigation is needed on those integrals. These integrals were marked in the summary table below and also in each integral separate section so they are easy to identify and locate.

1.14 Important notes about some of the results

1.14.1 Important note about Maxima results

Since tests were run in a batch mode, and using an automated script, then any integral where Maxima needed an interactive response from the user to answer a question during the evaluation of the integral will fail.

The exception raised is `ValueError`. Therefore Maxima results is lower than what would result if Maxima was run directly and each question was answered correctly.

The percentage of such failures were not counted for each test file, but for an example, for the `Timofeev` test file, there were about 14 such integrals out of total 705, or about 2 percent. This percentage can be higher or lower depending on the specific input test file.

Such integrals can be identified by looking at the output of the integration in each section for Maxima. The exception message will indicate the cause of error.

Maxima integrate was run using SageMath with the following settings set by default

```
'besselexpand : true'
'display2d : false'
'domain : complex'
'keepfloat : true'
'load(to_poly_solve)'
'load(simplify_sum)'
```

```
'load(abs_integrate)' 'load(diag)'
```

SageMath automatic loading of Maxima `abs_integrate` was found to cause some problems. So the following code was added to disable this effect.

```
from sage.interfaces.maxima_lib import maxima_lib
maxima_lib.set('extra_definite_integration_methods', '[]')
maxima_lib.set('extra_integration_methods', '[]')
```

See `https://ask.sagemath.org/question/43088/integrate-results-that-are-different-from-using-maxima/` for reference.

1.14.2 Important note about FriCAS result

There were few integrals which failed due to SageMath interface and not because FriCAS system could not do the integration.

These will fail With error `Exception raised: NotImplementedError`.

The number of such cases seems to be very small. About 1 or 2 percent of all integrals. These can be identified by looking at the exception message given in the result.

1.14.3 Important note about finding leaf size of antiderivative

For Mathematica, Rubi, and Maple, the builtin system function `LeafSize` was used to find the leaf size of each antiderivative.

The other CAS systems (SageMath and Sympy) do not have special builtin function for this purpose at this time. Therefore the leaf size for Fricas and Sympy antiderivative was determined using the following function, thanks to user `slelievre` at `https://ask.sagemath.org/question/57123/could-we-have-a-leaf_count-function-in-base-sagemath/`

```
def tree_size(expr):
    r"""
    Return the tree size of this expression.
    """
    if expr not in SR:
        # deal with lists, tuples, vectors
        return 1 + sum(tree_size(a) for a in expr)
    expr = SR(expr)
    x, aa = expr.operator(), expr.operands()
    if x is None:
```

```
    return 1
  else:
    return 1 + sum(tree_size(a) for a in aa)
```

For Sympy, which was called directly from Python, the following code was used to obtain the leafsize of its result

```
try:
 # 1.7 is a fudge factor since it is low side from actual leaf count
 leafCount = round(1.7*count_ops(anti))

 except Exception as ee:
      leafCount =1
```

1.14.4 Important note about Mupad results

Matlab's symbolic toolbox does not have a leaf count function to measure the size of the antiderivative. Maple was used to determine the leaf size of Mupad output by post processing Mupad result.

Currently no grading of the antiderivative for Mupad is implemented. If it can integrate the problem, it was assigned a B grade automatically as a placeholder. In the future, when grading function is implemented for Mupad, the tests will be rerun again.

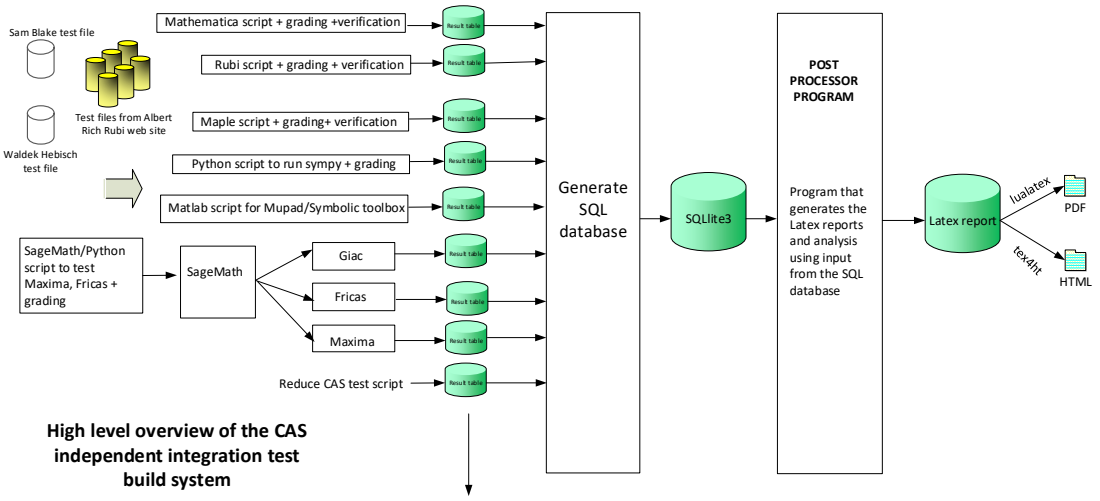
The following is an example of using Matlab's symbolic toolbox (Mupad) to solve an integral

```
integrand = evalin(symengine,'cos(x)*sin(x)')
the_variable = evalin(symengine,'x')
anti = int(integrand,the_variable)
```

Which gives `sin(x)^2/2`

1.15 Design of the test system

The following diagram gives a high level view of the current test build system.



High level overview of the CAS independent integration test build system

One record (line) per one integral result. The line is CSV comma separated. This is description of each record

1. integer, the problem number.
2. integer. 0 for failed, 1 for passed, -1 for timeout, -2 for CAS specific exception. (this is not the grade field)
3. integer. Leaf size of result.
4. integer. Leaf size of the optimal antiderivative.
5. number. CPU time used to solve this integral. 0 if failed.
6. string. The integral in Latex format
7. string. The input used in CAS own syntax.
8. string. The result (antiderivative) produced by CAS in Latex format
9. string. The optimal antiderivative in Latex format.
10. integer. 0 or 1. Indicates if problem has known antiderivative or not
11. String. The result (antiderivative) in CAS own syntax.
12. String. The grade of the antiderivative. Can be “A”, “B”, “C”, or “F”
13. String. Small string description of why the grade was given.
14. integer. 1 if result was verified or 0 if not verified. (For mma, rubi and maple only)

*The following fields are present only in **Rubi** Table file*

15. integer. Number of steps used.
16. integer. Number of rules used.
17. integer. Integrand leaf size.
18. real number. Ratio. Field 16 over field 17
19. String of form “{n,n,..}” which is list of the rules used by Rubi
20. String. The optimal antiderivative in Mathematica syntx

Nasser M. Abbasi
January 13, 2024
Design.vsdx

CHAPTER 2

LINKS TO INDIVIDUAL TEST REPORTS

These are links to each test report. The number in square brackets to right of the link is the number of integrals in the test. The list of numbers in the curly brackets after that (if any) is the list of the integrals in that specific test which were solved by any CAS which are known not to have antiderivative. This makes it easier to find these integrals and do more investigation into them.

2.1 Tests completed

1. 0_Independent_test_suites/1_Apostol_Problems [175]
2. 0_Independent_test_suites/2_Bondarenko_Problems [35]
3. 0_Independent_test_suites/3_Bronstein_Problems [14]
4. 0_Independent_test_suites/4_Charlwood_Problems [50]
5. 0_Independent_test_suites/5_Hearn_Problems [284] { **Maxima: 145.** }
6. 0_Independent_test_suites/6_Hebisch_Problems [7]
7. 0_Independent_test_suites/7_Jeffrey_Problems [9]
8. 0_Independent_test_suites/8_Moses_Problems [113]
9. 0_Independent_test_suites/9_Stewart_Problems [376]
10. 0_Independent_test_suites/10_Timofeev_Problems [705]
11. 0_Independent_test_suites/11_Welz_Problems [116]
12. 0_Independent_test_suites/12_Wester_Problems [8]
13. 1_Algebraic_functions/1.1_Binomial_products/1.1.1_Linear/13_1.1.1.2-a+b_x-^{m}-c+d_x-^{n} [1917]

2.2 Tests yet to be completed

These tests below are still to be done. Links shown are only place holders and do not work now.

14. 1_Algebraic_functions/1.1_Binomial_products/1.1.1_Linear/14_1.1.1.3-a+b_x-^{m}-c+d_x-^{n}-e+f_x-^{p}
15. 1_Algebraic_functions/1.1_Binomial_products/1.1.1_Linear/15_1.1.1.4-a+b_x-^{m}-c+d_x-^{n}-e+f_x-^{p}-g+h_x-^{q}
16. 1_Algebraic_functions/1.1_Binomial_products/1.1.1_Linear/16_1.1.1.5_P-x-a+b_x-^{m}-c+d_x-^{n}
17. 1_Algebraic_functions/1.1_Binomial_products/1.1.1_Linear/17_1.1.1.6_P-x-a+b_x-^{m}-c+d_x-^{n}-e+f_x-^{p}
18. 1_Algebraic_functions/1.1_Binomial_products/1.1.1_Linear/18_1.1.1.7_P-x-a+b_x-^{m}-c+d_x-^{n}-e+f_x-^{p}-g+h_x-^{q}
19. 1_Algebraic_functions/1.1_Binomial_products/1.1.2_Quadratic/19_1.1.2.2-c_x-^{m}-a+b_x^{2}-^{p}
20. 1_Algebraic_functions/1.1_Binomial_products/1.1.2_Quadratic/20_1.1.2.3-a+b_x^{2}-^{p}-c+d_x^{2}-^{q}
21. 1_Algebraic_functions/1.1_Binomial_products/1.1.2_Quadratic/21_1.1.2.4-e_x-^{m}-a+b_x^{2}-^{p}-c+d_x^{2}-^{q}
22. 1_Algebraic_functions/1.1_Binomial_products/1.1.2_Quadratic/22_1.1.2.5-a+b_x^{2}-^{p}-c+d_x^{2}-^{q}-e+f_x^{2}-^{r}
23. 1_Algebraic_functions/1.1_Binomial_products/1.1.2_Quadratic/23_1.1.2.6-g_x-^{m}-a+b_x^{2}-^{p}-c+d_x^{2}-^{q}-e+f_x^{2}-^{r}
24. 1_Algebraic_functions/1.1_Binomial_products/1.1.2_Quadratic/24_1.1.2.8_P-x-c_x-^{m}-a+b_x^{2}-^{p}
25. 1_Algebraic_functions/1.1_Binomial_products/1.1.3_General/25_1.1.3.2-c_x-^{m}-a+b_x^{n}-^{p}
26. 1_Algebraic_functions/1.1_Binomial_products/1.1.3_General/26_1.1.3.3-a+b_x^{n}-^{p}-c+d_x^{n}-^{q}
27. 1_Algebraic_functions/1.1_Binomial_products/1.1.3_General/27_1.1.3.4-e_x-^{m}-a+b_x^{n}-^{p}-c+d_x^{n}-^{q}

28. 1_Algebraic_functions/1.1_Binomial_products/1.1.3_General/28_1.1.3.6-g_x-m-a+b_xn-p-c+d_xn-q-e+f_xn-r
29. 1_Algebraic_functions/1.1_Binomial_products/1.1.3_General/29_1.1.3.8_P-x-c_x-m-a+b_xn-p
30. 1_Algebraic_functions/1.1_Binomial_products/1.1.4_Improper/30_1.1.4.2-c_x-m-a_xj+b_xn-p
31. 1_Algebraic_functions/1.1_Binomial_products/1.1.4_Improper/31_1.1.4.3-e_x-m-a_xj+b_xk-p-c+d_xn-q
32. 1_Algebraic_functions/1.2_Trinomial_products/1.2.1_Quadratic/32_1.2.1.1-a+b_x+c_x2-p
33. 1_Algebraic_functions/1.2_Trinomial_products/1.2.1_Quadratic/33_1.2.1.2-d+e_x-m-a+b_x+c_x2-p
34. 1_Algebraic_functions/1.2_Trinomial_products/1.2.1_Quadratic/34_1.2.1.3-d+e_x-m-f+g_x-a+b_x+c_x2-p
35. 1_Algebraic_functions/1.2_Trinomial_products/1.2.1_Quadratic/35_1.2.1.4-d+e_x-m-f+g_x-n-a+b_x+c_x2-p
36. 1_Algebraic_functions/1.2_Trinomial_products/1.2.1_Quadratic/36_1.2.1.5-a+b_x+c_x2-p-d+e_x+f_x2-q
37. 1_Algebraic_functions/1.2_Trinomial_products/1.2.1_Quadratic/37_1.2.1.6-g+h_x-m-a+b_x+c_x2-p-d+e_x+f_x2-q
38. 1_Algebraic_functions/1.2_Trinomial_products/1.2.1_Quadratic/38_1.2.1.9_P-x-d+e_x-m-a+b_x+c_x2-p
39. 1_Algebraic_functions/1.2_Trinomial_products/1.2.2_Quartic/39_1.2.2.2-d_x-m-a+b_x2+c_x4-p
40. 1_Algebraic_functions/1.2_Trinomial_products/1.2.2_Quartic/40_1.2.2.3-d+e_x2-m-a+b_x2+c_x4-p
41. 1_Algebraic_functions/1.2_Trinomial_products/1.2.2_Quartic/41_1.2.2.4-f_x-m-d+e_x2-q-a+b_x2+c_x4-p
42. 1_Algebraic_functions/1.2_Trinomial_products/1.2.2_Quartic/42_1.2.2.5_P-x-a+b_x2+c_x4-p
43. 1_Algebraic_functions/1.2_Trinomial_products/1.2.2_Quartic/43_1.2.2.6_P-x-d_x-m-a+b_x2+c_x4-p

44. 1_Algebraic_functions/1.2_Trinomial_products/1.2.2_Quartic/44_1.2.2.7_P-x-d+e_x^2-q-a+b_x^2+c_x^4-p
45. 1_Algebraic_functions/1.2_Trinomial_products/1.2.2_Quartic/45_1.2.2.8_P-x-d+e_x-q-a+b_x^2+c_x^4-p
46. 1_Algebraic_functions/1.2_Trinomial_products/1.2.3_General/46_1.2.3.2-d_x-m-a+b_x^n+c_x^2_n-p
47. 1_Algebraic_functions/1.2_Trinomial_products/1.2.3_General/47_1.2.3.3-d+e_x^n-q-a+b_x^n+c_x^2_n-p
48. 1_Algebraic_functions/1.2_Trinomial_products/1.2.3_General/48_1.2.3.4-f_x-m-d+e_x^n-q-a+b_x^n+c_x^2_n-p
49. 1_Algebraic_functions/1.2_Trinomial_products/1.2.3_General/49_1.2.3.5_P-x-d_x-m-a+b_x^n+c_x^2_n-p
50. 1_Algebraic_functions/1.2_Trinomial_products/1.2.4_Improper/50_1.2.4.2-d_x-m-a_x^q+b_x^n+c_x^2_n-q-p
51. 1_Algebraic_functions/1.3_Miscellaneous/51_1.3.1_Rational_functions
52. 1_Algebraic_functions/1.3_Miscellaneous/52_1.3.2_Algebraic_functions
53. 2_Exponentials/53_2.1_u-F^c-a+b_x-n
54. 2_Exponentials/54_2.2-c+d_x-m-F^g-e+f_x-n-a+b-F^g-e+f_x-n-p
55. 2_Exponentials/55_2.3_Exponential_functions
56. 3_Logarithms/56_3.1.2-d_x-m-a+b_log-c_x^n-p
57. 3_Logarithms/57_3.1.4-f_x-m-d+e_x^r-q-a+b_log-c_x^n-p
58. 3_Logarithms/58_3.1.5_u-a+b_log-c_x^n-p
59. 3_Logarithms/59_3.2.1-f+g_x-m-A+B_log-e-a+b_x-over-c+d_x-n-p
60. 3_Logarithms/60_3.2.2-f+g_x-m-h+i_x-q-A+B_log-e-a+b_x-over-c+d_x_n_p
61. 3_Logarithms/61_3.2.3_u_log-e-f-a+b_x-p-c+d_x-q-r-s
62. 3_Logarithms/62_3.3_u-a+b_log-c-d+e_x-n-p
63. 3_Logarithms/63_3.4_u-a+b_log-c-d+e_x^m-n-p

64. 3_Logarithms/64_3.5_Logarithm_functions
65. 4_Trig_functions/4.1_Sine/65_4.1.0-a_sin-^{m}-b_trg-^{n}
66. 4_Trig_functions/4.1_Sine/66_4.1.10-c+d_x-^{m}-a+b_sin-^{n}
67. 4_Trig_functions/4.1_Sine/67_4.1.1.1-a+b_sin-^{n}
68. 4_Trig_functions/4.1_Sine/68_4.1.11-e_x-^{m}-a+b_x^{n}-^{p}_sin
69. 4_Trig_functions/4.1_Sine/69_4.1.12-e_x-^{m}-a+b_sin-c+d_x^{n}-^{p}
70. 4_Trig_functions/4.1_Sine/70_4.1.1.2-g_cos-^{p}-a+b_sin-^{m}
71. 4_Trig_functions/4.1_Sine/71_4.1.13-d+e_x-^{m}_sin-a+b_x+c_x^{2}-^{n}
72. 4_Trig_functions/4.1_Sine/72_4.1.1.3-g_tan-^{p}-a+b_sin-^{m}
73. 4_Trig_functions/4.1_Sine/73_4.1.2.1-a+b_sin-^{m}-c+d_sin-^{n}
74. 4_Trig_functions/4.1_Sine/74_4.1.2.2-g_cos-^{p}-a+b_sin-^{m}-c+d_sin-^{n}
75. 4_Trig_functions/4.1_Sine/75_4.1.2.3-g_sin-^{p}-a+b_sin-^{m}-c+d_sin-^{n}
76. 4_Trig_functions/4.1_Sine/76_4.1.3.1-a+b_sin-^{m}-c+d_sin-^{n}-A+B_sin-
77. 4_Trig_functions/4.1_Sine/77_4.1.4.1-a+b_sin-^{m}-A+B_sin+C_sin^{2}-
78. 4_Trig_functions/4.1_Sine/78_4.1.4.2-a+b_sin-^{m}-c+d_sin-^{n}-A+B_sin+C_sin^{2}-
79. 4_Trig_functions/4.1_Sine/79_4.1.7-d_trig-^{m}-a+b-c_sin-^{n}-^{p}
80. 4_Trig_functions/4.1_Sine/80_4.1.8-a+b_sin-^{m}-c+d_trig-^{n}
81. 4_Trig_functions/4.1_Sine/81_4.1.9_trig^{m}-a+b_sin^{n}+c_sin-2_n-^{p}
82. 4_Trig_functions/4.2_Cosine/82_4.2.0-a_cos-^{m}-b_trg-^{n}
83. 4_Trig_functions/4.2_Cosine/83_4.2.10-c+d_x-^{m}-a+b_cos-^{n}
84. 4_Trig_functions/4.2_Cosine/84_4.2.1.1-a+b_cos-^{n}
85. 4_Trig_functions/4.2_Cosine/85_4.2.12-e_x-^{m}-a+b_cos-c+d_x^{n}-^{p}
86. 4_Trig_functions/4.2_Cosine/86_4.2.1.2-g_sin-^{p}-a+b_cos-^{m}
87. 4_Trig_functions/4.2_Cosine/87_4.2.13-d+e_x-^{m}_cos-a+b_x+c_x^{2}-^{n}
88. 4_Trig_functions/4.2_Cosine/88_4.2.1.3-g_tan-^{p}-a+b_cos-^{m}
89. 4_Trig_functions/4.2_Cosine/89_4.2.2.1-a+b_cos-^{m}-c+d_cos-^{n}
90. 4_Trig_functions/4.2_Cosine/90_4.2.2.2-g_sin-^{p}-a+b_cos-^{m}-c+d_cos-^{n}

91. 4_Trig_functions/4.2_Cosine/91_4.2.2.3-g_cos-p-a+b_cos-m-c+d_cos-n
92. 4_Trig_functions/4.2_Cosine/92_4.2.3.1-a+b_cos-m-c+d_cos-n-A+B_cos-
93. 4_Trig_functions/4.2_Cosine/93_4.2.4.1-a+b_cos-m-A+B_cos+C_cos2-
94. 4_Trig_functions/4.2_Cosine/94_4.2.4.2-a+b_cos-m-c+d_cos-n-A+B_cos+C_cos2-
95. 4_Trig_functions/4.2_Cosine/95_4.2.7-d_trig-m-a+b-c_cos-n-p
96. 4_Trig_functions/4.2_Cosine/96_4.2.8-a+b_cos-m-c+d_trig-n
97. 4_Trig_functions/4.2_Cosine/97_4.2.9_trigm-a+b_cosn+c_cos-2_n-p
98. 4_Trig_functions/4.3_Tangent/98_4.3.0-a_trg-m-b_tan-n
99. 4_Trig_functions/4.3_Tangent/99_4.3.10-c+d_x-m-a+b_tan-n
100. 4_Trig_functions/4.3_Tangent/100_4.3.11-e_x-m-a+b_tan-c+d_xn-p
101. 4_Trig_functions/4.3_Tangent/101_4.3.1.2-d_sec-m-a+b_tan-n
102. 4_Trig_functions/4.3_Tangent/102_4.3.1.3-d_sin-m-a+b_tan-n
103. 4_Trig_functions/4.3_Tangent/103_4.3.2.1-a+b_tan-m-c+d_tan-n
104. 4_Trig_functions/4.3_Tangent/104_4.3.3.1-a+b_tan-m-c+d_tan-n-A+B_tan-
105. 4_Trig_functions/4.3_Tangent/105_4.3.4.2-a+b_tan-m-c+d_tan-n-A+B_tan+C_tan2-
106. 4_Trig_functions/4.3_Tangent/106_4.3.7-d_trig-m-a+b-c_tan-n-p
107. 4_Trig_functions/4.3_Tangent/107_4.3.9_trigm-a+b_tann+c_tan-2_n-p
108. 4_Trig_functions/4.4_Cotangent/108_4.4.0-a_trg-m-b_cot-n
109. 4_Trig_functions/4.4_Cotangent/109_4.4.10-c+d_x-m-a+b_cot-n
110. 4_Trig_functions/4.4_Cotangent/110_4.4.1.2-d_csc-m-a+b_cot-n
111. 4_Trig_functions/4.4_Cotangent/111_4.4.1.3-d_cos-m-a+b_cot-n
112. 4_Trig_functions/4.4_Cotangent/112_4.4.2.1-a+b_cot-m-c+d_cot-n
113. 4_Trig_functions/4.4_Cotangent/113_4.4.7-d_trig-m-a+b-c_cot-n-p
114. 4_Trig_functions/4.4_Cotangent/114_4.4.9_trigm-a+b_cotn+c_cot-2_n-p

115. 4_Trig_functions/4.5_Secant/115_4.5.0-a_sec-^{m}-b_trg-^{n}
116. 4_Trig_functions/4.5_Secant/116_4.5.10-c+d_x-^{m}-a+b_sec-^{n}
117. 4_Trig_functions/4.5_Secant/117_4.5.11-e_x-^{m}-a+b_sec-c+d_x^{n}-^{p}
118. 4_Trig_functions/4.5_Secant/118_4.5.1.2-d_sec-^{n}-a+b_sec-^{m}
119. 4_Trig_functions/4.5_Secant/119_4.5.1.3-d_sin-^{n}-a+b_sec-^{m}
120. 4_Trig_functions/4.5_Secant/120_4.5.1.4-d_tan-^{n}-a+b_sec-^{m}
121. 4_Trig_functions/4.5_Secant/121_4.5.2.1-a+b_sec-^{m}-c+d_sec-^{n}
122. 4_Trig_functions/4.5_Secant/122_4.5.2.3-g_sec-^{p}-a+b_sec-^{m}-c+d_sec_^{n}
123. 4_Trig_functions/4.5_Secant/123_4.5.3.1-a+b_sec-^{m}-d_sec-^{n}-A+B_sec-
124. 4_Trig_functions/4.5_Secant/124_4.5.4.1-a+b_sec-^{m}-A+B_sec+C_sec^{2}-
125. 4_Trig_functions/4.5_Secant/125_4.5.4.2-a+b_sec-^{m}-d_sec-^{n}-A+B_sec+C_sec^{2}-
126. 4_Trig_functions/4.5_Secant/126_4.5.7-d_trig-^{m}-a+b-c_sec-^{n}-^{p}
127. 4_Trig_functions/4.6_Cosecant/127_4.6.0-a_csc-^{m}-b_trg-^{n}
128. 4_Trig_functions/4.6_Cosecant/128_4.6.11-e_x-^{m}-a+b_csc-c+d_x^{n}-^{p}
129. 4_Trig_functions/4.6_Cosecant/129_4.6.1.2-d_csc-^{n}-a+b_csc-^{m}
130. 4_Trig_functions/4.6_Cosecant/130_4.6.1.3-d_cos-^{n}-a+b_csc-^{m}
131. 4_Trig_functions/4.6_Cosecant/131_4.6.1.4-d_cot-^{n}-a+b_csc-^{m}
132. 4_Trig_functions/4.6_Cosecant/132_4.6.3.1-a+b_csc-^{m}-d_csc-^{n}-A+B_csc-
133. 4_Trig_functions/4.6_Cosecant/133_4.6.4.2-a+b_csc-^{m}-d_csc-^{n}-A+B_csc+C_csc^{2}-
134. 4_Trig_functions/4.6_Cosecant/134_4.6.7-d_trig-^{m}-a+b-c_csc-^{n}-^{p}
135. 4_Trig_functions/4.7_Miscellaneous/135_4.7.1-c_trig-^{m}-d_trig-^{n}
136. 4_Trig_functions/4.7_Miscellaneous/136_4.7.2_trig^{m}-a_trig+b_trig-^{n}
137. 4_Trig_functions/4.7_Miscellaneous/137_4.7.3-c+d_x-^{m}_trig^{n}_trig^{p}
138. 4_Trig_functions/4.7_Miscellaneous/138_4.7.4_x^{m}-a+b_trig^{n}-^{p}

139. 4_Trig_functions/4.7_Miscellaneous/139_4.7.5_x^m_trig-a+b_log-c_x^n-p

140. 4_Trig_functions/4.7_Miscellaneous/140_4.7.6_f^-a+b_x+c_x^2-trig-d+e_x+f_x^2-n

141. 4_Trig_functions/4.7_Miscellaneous/141_4.7.7_Trig_functions

142. 5_Inverse_trig_functions/5.1_Inverse_sine/142_5.1.2-d_x-m-a+b_arcsin-c_x-n

143. 5_Inverse_trig_functions/5.1_Inverse_sine/143_5.1.4-f_x-m-d+e_x^2-p-a+b_arcsin-c_x-n

144. 5_Inverse_trig_functions/5.1_Inverse_sine/144_5.1.5_Inverse_sine_functions

145. 5_Inverse_trig_functions/5.2_Inverse_cosine/145_5.2.2-d_x-m-a+b_arccos-c_x-n

146. 5_Inverse_trig_functions/5.2_Inverse_cosine/146_5.2.4-f_x-m-d+e_x^2-p-a+b_arccos-c_x-n

147. 5_Inverse_trig_functions/5.2_Inverse_cosine/147_5.2.5_Inverse_cosine_functions

148. 5_Inverse_trig_functions/5.3_Inverse_tangent/148_5.3.2-d_x-m-a+b_arctan-c_xn-p

149. 5_Inverse_trig_functions/5.3_Inverse_tangent/149_5.3.3-d+e_x-m-a+b_arctan-c_xn-p

150. 5_Inverse_trig_functions/5.3_Inverse_tangent/150_5.3.4_u-a+b_arctan-c_x-p

151. 5_Inverse_trig_functions/5.3_Inverse_tangent/151_5.3.5_u-a+b_arctan-c+d_x-p

152. 5_Inverse_trig_functions/5.3_Inverse_tangent/152_5.3.6_Exponentials_of_inverse_tangent

153. 5_Inverse_trig_functions/5.3_Inverse_tangent/153_5.3.7_Inverse_tangent_functions

154. 5_Inverse_trig_functions/5.4_Inverse_cotangent/154_5.4.1_Inverse_cotangent_functions

155. 5_Inverse_trig_functions/5.4_Inverse_cotangent/155_5.4.2_Exponentials_of_inverse_cotangent

156. 5_Inverse_trig_functions/5.5_Inverse_secant/156_5.5.1_u-a+b_arcsec-c_x-^{n}
157. 5_Inverse_trig_functions/5.5_Inverse_secant/157_5.5.2_Inverse_secant_functions
158. 5_Inverse_trig_functions/5.6_Inverse_cosecant/158_5.6.1_u-a+b_arccsc-c_x-^{n}
159. 5_Inverse_trig_functions/5.6_Inverse_cosecant/159_5.6.2_Inverse_cosecant_functions
160. 6_Hyperbolic_functions/6.1_Hyperbolic_sine/160_6.1.1-c+d_x-^{m}-a+b_sinh-^{n}
161. 6_Hyperbolic_functions/6.1_Hyperbolic_sine/161_6.1.3-e_x-^{m}-a+b_sinh-c+d_x^{n}-^{p}
162. 6_Hyperbolic_functions/6.1_Hyperbolic_sine/162_6.1.4-d+e_x-^{m}_sinh-a+b_x+c_x^{2}-^{n}
163. 6_Hyperbolic_functions/6.1_Hyperbolic_sine/163_6.1.5_Hyperbolic_sine_functions
164. 6_Hyperbolic_functions/6.1_Hyperbolic_sine/164_6.1.7_hyper^{m}-a+b_sinh^{n}-^{p}
165. 6_Hyperbolic_functions/6.2_Hyperbolic_cosine/165_6.2.1-c+d_x-^{m}-a+b_cosh-^{n}
166. 6_Hyperbolic_functions/6.2_Hyperbolic_cosine/166_6.2.2-e_x-^{m}-a+b_x^{n}-^{p}_cosh
167. 6_Hyperbolic_functions/6.2_Hyperbolic_cosine/167_6.2.3-e_x-^{m}-a+b_cosh-c+d_x^{n}-^{p}
168. 6_Hyperbolic_functions/6.2_Hyperbolic_cosine/168_6.2.4-d+e_x-^{m}_cosh-a+b_x+c_x^{2}-^{n}
169. 6_Hyperbolic_functions/6.2_Hyperbolic_cosine/169_6.2.5_Hyperbolic_cosine_functions
170. 6_Hyperbolic_functions/6.2_Hyperbolic_cosine/170_6.2.7_hyper^{m}-a+b_cosh^{n}-^{p}
171. 6_Hyperbolic_functions/6.3_Hyperbolic_tangent/171_6.3.1-c+d_x-^{m}-a+b_tanh-^{n}

172. 6_Hyperbolic_functions/6.3_Hyperbolic_tangent/172_6.3.2_Hyperbolic_tangent_functions

173. 6_Hyperbolic_functions/6.3_Hyperbolic_tangent/173_6.3.7-d_hyper-^{m}-a+b-c_tanh-^{n}-^{p}

174. 6_Hyperbolic_functions/6.4_Hyperbolic_cotangent/174_6.4.1-c+d_x-^{m}-a+b_coth-^{n}

175. 6_Hyperbolic_functions/6.4_Hyperbolic_cotangent/175_6.4.2_Hyperbolic_cotangent_functions

176. 6_Hyperbolic_functions/6.4_Hyperbolic_cotangent/176_6.4.7-d_hyper-^{m}-a+b-c_coth-^{n}-^{p}

177. 6_Hyperbolic_functions/6.5_Hyperbolic_secant/177_6.5.1-c+d_x-^{m}-a+b_sech-^{n}

178. 6_Hyperbolic_functions/6.5_Hyperbolic_secant/178_6.5.2-e_x-^{m}-a+b_sech-c+d_x^{n}-^{p}

179. 6_Hyperbolic_functions/6.5_Hyperbolic_secant/179_6.5.3_Hyperbolic_secant_functions

180. 6_Hyperbolic_functions/6.5_Hyperbolic_secant/180_6.5.7-d_hyper-^{m}-a+b-c_sech-^{n}-^{p}

181. 6_Hyperbolic_functions/6.6_Hyperbolic_cosecant/181_6.6.1-c+d_x-^{m}-a+b_csch-^{n}

182. 6_Hyperbolic_functions/6.6_Hyperbolic_cosecant/182_6.6.2-e_x-^{m}-a+b_csch-c+d_x^{n}-^{p}

183. 6_Hyperbolic_functions/6.6_Hyperbolic_cosecant/183_6.6.3_Hyperbolic_cosecant_functions

184. 6_Hyperbolic_functions/6.6_Hyperbolic_cosecant/184_6.6.7-d_hyper-^{m}-a+b-c_csch-^{n}-^{p}

185. 6_Hyperbolic_functions/6.7_Miscellaneous/185_6.7.1_Hyperbolic_functions

186. 7_Inverse_hyperbolic_functions/7.1_Inverse_hyperbolic_sine/186_7.1.2-d_x-^{m}-a+b_arcsinh-c_x-^{n}

187. 7_Inverse_hyperbolic_functions/7.1_Inverse_hyperbolic_sine/187_7.1.4-f_x-^{m}-d+e_x^{2}-^{p}-a+b_arcsinh-c_x-^{n}

188. 7_Inverse_hyperbolic_functions/7.1_Inverse_hyperbolic_sine/188_7.1.5_Inverse_hyperbolic_sine_functions
189. 7_Inverse_hyperbolic_functions/7.2_Inverse_hyperbolic_cosine/189_7.2.2-d_x-^{m}-a+b_arccosh-c_x-^{n}
190. 7_Inverse_hyperbolic_functions/7.2_Inverse_hyperbolic_cosine/190_7.2.4-f_x-^{m}-d+e_x^{2}-^{p}-a+b_arccosh-c_x-^{n}
191. 7_Inverse_hyperbolic_functions/7.2_Inverse_hyperbolic_cosine/191_7.2.5_Inverse_hyperbolic_cosine_functions
192. 7_Inverse_hyperbolic_functions/7.3_Inverse_hyperbolic_tangent/192_7.3.2-d_x-^{m}-a+b_arctanh-c_x^{n}-^{p}
193. 7_Inverse_hyperbolic_functions/7.3_Inverse_hyperbolic_tangent/193_7.3.3-d+e_x-^{m}-a+b_arctanh-c_x^{n}-^{p}
194. 7_Inverse_hyperbolic_functions/7.3_Inverse_hyperbolic_tangent/194_7.3.4_u-a+b_arctanh-c_x-^{p}
195. 7_Inverse_hyperbolic_functions/7.3_Inverse_hyperbolic_tangent/195_7.3.5_u-a+b_arctanh-c+d_x-^{p}
196. 7_Inverse_hyperbolic_functions/7.3_Inverse_hyperbolic_tangent/196_7.3.6_Exponentials_of_inverse_hyperbolic_tangent_functions
197. 7_Inverse_hyperbolic_functions/7.3_Inverse_hyperbolic_tangent/197_7.3.7_Inverse_hyperbolic_tangent_functions
198. 7_Inverse_hyperbolic_functions/7.4_Inverse_hyperbolic_cotangent/198_7.4.1_Inverse_hyperbolic_cotangent_functions
199. 7_Inverse_hyperbolic_functions/7.4_Inverse_hyperbolic_cotangent/199_7.4.2_Exponentials_of_inverse_hyperbolic_cotangent_functions
200. 7_Inverse_hyperbolic_functions/7.5_Inverse_hyperbolic_secant/200_7.5.1_u-a+b_arcsech-c_x-^{n}
201. 7_Inverse_hyperbolic_functions/7.5_Inverse_hyperbolic_secant/201_7.5.2_Inverse_hyperbolic_secant_functions
202. 7_Inverse_hyperbolic_functions/7.6_Inverse_hyperbolic_cosecant/202_7.6.1_u-a+b_arccsch-c_x-^{n}
203. 7_Inverse_hyperbolic_functions/7.6_Inverse_hyperbolic_cosecant/203_7.6.2_Inverse_hyperbolic_cosecant_functions

204. 8_Special_functions/204_8.1_Error_functions
205. 8_Special_functions/205_8.2_Fresnel_integral_functions
206. 8_Special_functions/206_8.4_Trig_integral_functions
207. 8_Special_functions/207_8.5_Hyperbolic_integral_functions
208. 8_Special_functions/208_8.8_Polylogarithm_function
209. 209_Blake_problems
210. 210_1_Hebisch
211. 210_2_Hebisch
212. 210_3_Hebisch
213. 210_4_Hebisch
214. 11_MIT
215. 12_table_of_integrals
216. 213_Goursat

CHAPTER 3

Listing of integrals solved by CAS which has no known antiderivatives

3.1 Test file Number [5]

3.1.1 Maxima

Integral number [145]

$$\int x \cos(k \csc(x)) \cot(x) \csc(x)\, dx$$

[C] time = 0.222839 (sec), size = 240 ,normalized size = 21.82

$$-\frac{\left(x e^{\left(\frac{4 k \cos(2x)\cos(x)}{\cos(2x)^2+\sin(2x)^2-2\cos(2x)+1}+\frac{4 k \sin(2x)\sin(x)}{\cos(2x)^2+\sin(2x)^2-2\cos(2x)+1}\right)}+x e^{\left(\frac{4 k \cos(x)}{\cos(2x)^2+\sin(2x)^2-2\cos(2x)+1}\right)}\right) e^{\left(-\frac{2 k \cos(2x)\cos(x)}{\cos(2x)^2+\sin(2x)^2-2\cos(2x)+1}-\frac{2 k \sin(2x)\sin(x)}{\cos(2x)^2+\sin(2x)^2-2\cos(2x)+1}-\frac{2 k \cos(x)}{\cos(2x)^2+\sin(2x)^2-2\cos(2x)+1}\right)} \sin\left(\frac{2(k\cos(x)\sin(2x)-k\cos(2x)\sin(x)+k\sin(x))}{\cos(2x)^2+\sin(2x)^2-2\cos(2x)+1}\right)}{2k}$$

```
[In] integrate(x*cos(x)*cos(k/sin(x))/sin(x)^2,x, algorithm=""maxima"")
```

output

```
-1/2*(x*e^(4*k*cos(2*x)*cos(x)/(cos(2*x)^2 + sin(2*x)^2 - 2*cos(2*x) + 1) + 4*
k*sin(2*x)*sin(x)/(cos(2*x)^2 + sin(2*x)^2 - 2*cos(2*x) + 1)) + x*e^(4*k*cos(x
)/(cos(2*x)^2 + sin(2*x)^2 - 2*cos(2*x) + 1)))*e^(-2*k*cos(2*x)*cos(x)/(cos(2*
x)^2 + sin(2*x)^2 - 2*cos(2*x) + 1) - 2*k*sin(2*x)*sin(x)/(cos(2*x)^2 + sin(2*
x)^2 - 2*cos(2*x) + 1) - 2*k*cos(x)/(cos(2*x)^2 + sin(2*x)^2 - 2*cos(2*x) + 1)
)*sin(2*(k*cos(x)*sin(2*x) - k*cos(2*x)*sin(x) + k*sin(x))/(cos(2*x)^2 + sin(2
*x)^2 - 2*cos(2*x) + 1))/k
```

CHAPTER 4

Appendix

4.1 Listing of grading functions

The following are the current version of the grading functions used for grading the quality of the antiderivative with reference to the optimal antiderivative included in the test suite.

There is a version for Maple and for Mathematica/Rubi. There is a version for grading Sympy and version for use with Sagemath.

The following are links to the current source code.

1. Mathematica and Rubi grading function GradeAntiderivative.m
2. Maple grading function GradeAntiderivative.mpl
3. Sympy grading function grade_sympy.py
4. Sagemath grading function grade_sagemath.py

The following are the listings of source code of the grading functions.

4.1.1 Mathematica and Rubi grading function

```
(* Original version thanks to Albert Rich emailed on 03/21/2017 *)
(* ::Package:: *)

(* Nasser: April 7,2022. add second output which gives reason for the grade *)
(*                       Small rewrite of logic in main function to make it*)
(*                       match Maple's logic. No change in functionality otherwise*)

(* ::Subsection:: *)
(*GradeAntiderivative[result,optimal]*)


(* ::Text:: *)
(*If result and optimal are mathematical expressions, *)
(*          GradeAntiderivative[result,optimal] returns*)
(*  "F" if the result fails to integrate an expression that*)
(*      is integrable*)
(*  "C" if result involves higher level functions than necessary*)
(*  "B" if result is more than twice the size of the optimal*)
(*      antiderivative*)
(*  "A" if result can be considered optimal*)
```

```
GradeAntiderivative[result_,optimal_] := Module[{expnResult,expnOptimal,leafCountResult,leaf
    expnResult  = ExpnType[result];
    expnOptimal = ExpnType[optimal];
    leafCountResult = LeafCount[result];
    leafCountOptimal = LeafCount[optimal];

    (*Print["expnResult=",expnResult," expnOptimal=",expnOptimal];*)
    If[expnResult<=expnOptimal,
        If[Not[FreeQ[result,Complex]], (*result contains complex*)
            If[Not[FreeQ[optimal,Complex]], (*optimal contains complex*)
                If[leafCountResult<=2*leafCountOptimal,
                    finalresult={"A"," "}
                    ,(*ELSE*)
                    finalresult={"B","Both result and optimal contain complex but leaf count
                  ]
            ,(*ELSE*)
                finalresult={"C","Result contains complex when optimal does not."}
            ]
        ,(*ELSE*)(*result does not contains complex*)
            If[leafCountResult<=2*leafCountOptimal,
               finalresult={"A"," "}
            ,(*ELSE*)
               finalresult={"B","Leaf count is larger than twice the leaf count of optimal.
              ]
        ]
    ,(*ELSE*) (*expnResult>expnOptimal*)
        If[FreeQ[result,Integrate] && FreeQ[result,Int],
            finalresult={"C","Result contains higher order function than in optimal. Order "
            ,
            finalresult={"F","Contains unresolved integral."}
        ]
    ];

    finalresult
]

(* ::Text:: *)
(*The following summarizes the type number assigned an *)
(*expression based on the functions it involves*)
(*1 = rational function*)
```

```
(*2 = algebraic function*)
(*3 = elementary function*)
(*4 = special function*)
(*5 = hyperpergeometric function*)
(*6 = appell function*)
(*7 = rootsum function*)
(*8 = integrate function*)
(*9 = unknown function*)


ExpnType[expn_] :=
  If[AtomQ[expn],
    1,
  If[ListQ[expn],
    Max[Map[ExpnType,expn]],
  If[Head[expn]===Power,
    If[IntegerQ[expn[[2]]],
      ExpnType[expn[[1]]],
    If[Head[expn[[2]]]===Rational,
      If[IntegerQ[expn[[1]]] || Head[expn[[1]]]===Rational,
        1,
      Max[ExpnType[expn[[1]]],2]],
    Max[ExpnType[expn[[1]]],ExpnType[expn[[2]]],3]]],
  If[Head[expn]===Plus || Head[expn]===Times,
    Max[ExpnType[First[expn]],ExpnType[Rest[expn]]],
  If[ElementaryFunctionQ[Head[expn]],
    Max[3,ExpnType[expn[[1]]]],
  If[SpecialFunctionQ[Head[expn]],
    Apply[Max,Append[Map[ExpnType,Apply[List,expn]],4]],
  If[HypergeometricFunctionQ[Head[expn]],
    Apply[Max,Append[Map[ExpnType,Apply[List,expn]],5]],
  If[AppellFunctionQ[Head[expn]],
    Apply[Max,Append[Map[ExpnType,Apply[List,expn]],6]],
  If[Head[expn]===RootSum,
    Apply[Max,Append[Map[ExpnType,Apply[List,expn]],7]],
  If[Head[expn]===Integrate || Head[expn]===Int,
    Apply[Max,Append[Map[ExpnType,Apply[List,expn]],8]],
  9]]]]]]]]]]


ElementaryFunctionQ[func_] :=
```

```
  MemberQ[{
  Exp,Log,
  Sin,Cos,Tan,Cot,Sec,Csc,
  ArcSin,ArcCos,ArcTan,ArcCot,ArcSec,ArcCsc,
  Sinh,Cosh,Tanh,Coth,Sech,Csch,
  ArcSinh,ArcCosh,ArcTanh,ArcCoth,ArcSech,ArcCsch
},func]


SpecialFunctionQ[func_] :=
  MemberQ[{
  Erf, Erfc, Erfi,
  FresnelS, FresnelC,
  ExpIntegralE, ExpIntegralEi, LogIntegral,
  SinIntegral, CosIntegral, SinhIntegral, CoshIntegral,
  Gamma, LogGamma, PolyGamma,
  Zeta, PolyLog, ProductLog,
  EllipticF, EllipticE, EllipticPi
},func]


HypergeometricFunctionQ[func_] :=
  MemberQ[{Hypergeometric1F1,Hypergeometric2F1,HypergeometricPFQ},func]


AppellFunctionQ[func_] :=
  MemberQ[{AppellF1},func]
```

4.1.2 Maple grading function

```
# File: GradeAntiderivative.mpl
# Original version thanks to Albert Rich emailed on 03/21/2017

#Nasser 03/22/2017  Use Maple leaf count instead since buildin
#Nasser 03/23/2017  missing 'ln' for ElementaryFunctionQ added
#Nasser 03/24/2017  corrected the check for complex result
#Nasser 10/27/2017  check for leafsize and do not call ExpnType()
#                   if leaf size is "too large". Set at 500,000
#Nasser 12/22/2019  Added debug flag, added 'dilog' to special functions
#                   see problem 156, file Apostol_Problems
```

```
#Nasser 4/07/2022   add second output which gives reason for the grade

GradeAntiderivative := proc(result,optimal)
local leaf_count_result,
        leaf_count_optimal,
        ExpnType_result,
        ExpnType_optimal,
        debug:=false;

        leaf_count_result:=leafcount(result);
        #do NOT call ExpnType() if leaf size is too large. Recursion problem
        if leaf_count_result > 500000 then
            return "B","result has leaf size over 500,000. Avoiding possible recursion issue
        fi;

        leaf_count_optimal := leafcount(optimal);
        ExpnType_result  := ExpnType(result);
        ExpnType_optimal := ExpnType(optimal);

        if debug then
            print("ExpnType_result",ExpnType_result," ExpnType_optimal=",ExpnType_optimal);
        fi;

# If result and optimal are mathematical expressions,
#  GradeAntiderivative[result,optimal] returns
#   "F" if the result fails to integrate an expression that
#       is integrable
#   "C" if result involves higher level functions than necessary
#   "B" if result is more than twice the size of the optimal
#       antiderivative
#   "A" if result can be considered optimal

    #This check below actually is not needed, since I only
    #call this grading only for passed integrals. i.e. I check
    #for "F" before calling this. But no harm of keeping it here.
    #just in case.


    if not type(result,freeof('int')) then
        return "F","Result contains unresolved integral";
    fi;
```

```
if ExpnType_result<=ExpnType_optimal then
    if debug then
         print("ExpnType_result<=ExpnType_optimal");
    fi;
    if is_contains_complex(result) then
        if is_contains_complex(optimal) then
            if debug then
                    print("both result and optimal complex");
            fi;
            if leaf_count_result<=2*leaf_count_optimal then
                return "A"," ";
            else
               return "B",cat("Both result and optimal contain complex but leaf count of
                              convert(leaf_count_result,string)," vs. $2 (",
                              convert(leaf_count_optimal,string)," ) = ",convert(2*leaf
            end if
        else #result contains complex but optimal is not
            if debug then
                    print("result contains complex but optimal is not");
            fi;
            return "C","Result contains complex when optimal does not.";
        fi;
    else # result do not contain complex
         # this assumes optimal do not as well. No check is needed here.
        if debug then
               print("result do not contain complex, this assumes optimal do not as well
        fi;
        if leaf_count_result<=2*leaf_count_optimal then
            if debug then
                print("leaf_count_result<=2*leaf_count_optimal");
            fi;
            return "A"," ";
        else
            if debug then
                print("leaf_count_result>2*leaf_count_optimal");
            fi;
            return "B",cat("Leaf count of result is larger than twice the leaf count of
                              convert(leaf_count_result,string),"$ vs. $2(",
                              convert(leaf_count_optimal,string),")=",convert(2*leaf_co
```

```
            fi;
        fi;
    else #ExpnType(result) > ExpnType(optimal)
        if debug then
            print("ExpnType(result) > ExpnType(optimal)");
        fi;
        return "C",cat("Result contains higher order function than in optimal. Order ",
                       convert(ExpnType_result,string)," vs. order ",
                       convert(ExpnType_optimal,string),".");
    fi;

end proc:

#
# is_contains_complex(result)
# takes expressions and returns true if it contains "I" else false
#
#Nasser 032417
is_contains_complex:= proc(expression)
  return (has(expression,I));
end proc:

# The following summarizes the type number assigned an expression
# based on the functions it involves
# 1 = rational function
# 2 = algebraic function
# 3 = elementary function
# 4 = special function
# 5 = hyperpergeometric function
# 6 = appell function
# 7 = rootsum function
# 8 = integrate function
# 9 = unknown function

ExpnType := proc(expn)
  if type(expn,'atomic') then
    1
  elif type(expn,'list') then
    apply(max,map(ExpnType,expn))
  elif type(expn,'sqrt') then
    if type(op(1,expn),'rational') then
```

```
      1
    else
       max(2,ExpnType(op(1,expn)))
    end if
  elif type(expn,'`^`') then
    if type(op(2,expn),'integer') then
      ExpnType(op(1,expn))
    elif type(op(2,expn),'rational') then
      if type(op(1,expn),'rational') then
         1
      else
         max(2,ExpnType(op(1,expn)))
      end if
    else
         max(3,ExpnType(op(1,expn)),ExpnType(op(2,expn)))
    end if
  elif type(expn,'`+`') or type(expn,'`*`') then
    max(ExpnType(op(1,expn)),max(ExpnType(rest(expn))))
  elif ElementaryFunctionQ(op(0,expn)) then
    max(3,ExpnType(op(1,expn)))
  elif SpecialFunctionQ(op(0,expn)) then
    max(4,apply(max,map(ExpnType,[op(expn)])))
  elif HypergeometricFunctionQ(op(0,expn)) then
    max(5,apply(max,map(ExpnType,[op(expn)])))
  elif AppellFunctionQ(op(0,expn)) then
    max(6,apply(max,map(ExpnType,[op(expn)])))
  elif op(0,expn)='int' then
    max(8,apply(max,map(ExpnType,[op(expn)]))) else
  9
  end if
end proc:


ElementaryFunctionQ := proc(func)
  member(func,[
        exp,log,ln,
        sin,cos,tan,cot,sec,csc,
        arcsin,arccos,arctan,arccot,arcsec,arccsc,
        sinh,cosh,tanh,coth,sech,csch,
        arcsinh,arccosh,arctanh,arccoth,arcsech,arccsch])
end proc:
```

```
SpecialFunctionQ := proc(func)
  member(func,[
        erf,erfc,erfi,
        FresnelS,FresnelC,
        Ei,Ei,Li,Si,Ci,Shi,Chi,
        GAMMA,lnGAMMA,Psi,Zeta,polylog,dilog,LambertW,
        EllipticF,EllipticE,EllipticPi])
end proc:

HypergeometricFunctionQ := proc(func)
  member(func,[Hypergeometric1F1,hypergeom,HypergeometricPFQ])
end proc:

AppellFunctionQ := proc(func)
  member(func,[AppellF1])
end proc:

# u is a sum or product.  rest(u) returns all but the
# first term or factor of u.
rest := proc(u) local v;
  if nops(u)=2 then
     op(2,u)
  else
     apply(op(0,u),op(2..nops(u),u))
  end if
end proc:

#leafcount(u) returns the number of nodes in u.
#Nasser 3/23/17  Replaced by build-in leafCount from package in Maple
leafcount := proc(u)
   MmaTranslator[Mma][LeafCount](u);
end proc:
```

4.1.3 Sympy grading function

```
#Dec 24, 2019. Nasser M. Abbasi:
#               Port of original Maple grading function by
#               Albert Rich to use with Sympy/Python
#Dec 27, 2019  Nasser. Added `RootSum`. See problem 177, Timofeev file
#               added 'exp_polar'
from sympy import *

def leaf_count(expr):
   #sympy do not have leaf count function. This is approximation
   return round(1.7*count_ops(expr))

def is_sqrt(expr):
   if isinstance(expr,Pow):
      if expr.args[1] == Rational(1,2):
         return True
      else:
         return False
   else:
      return False

def is_elementary_function(func):
   return func in [exp,log,ln,sin,cos,tan,cot,sec,csc,
         asin,acos,atan,acot,asec,acsc,sinh,cosh,tanh,coth,sech,csch,
         asinh,acosh,atanh,acoth,asech,acsch
      ]

def is_special_function(func):
   return func in [ erf,erfc,erfi,
          fresnels,fresnelc,Ei,Ei,Li,Si,Ci,Shi,Chi,
          gamma,loggamma,digamma,zeta,polylog,LambertW,
          elliptic_f,elliptic_e,elliptic_pi,exp_polar
       ]

def is_hypergeometric_function(func):
   return func in [hyper]

def is_appell_function(func):
   return func in [appellf1]
```

```
def is_atom(expn):
    try:
        if expn.isAtom or isinstance(expn,int) or isinstance(expn,float):
            return True
        else:
            return False

    except AttributeError as error:
        return False

def expnType(expn):
    debug=False
    if debug:
        print("expn=",expn,"type(expn)=",type(expn))

    if is_atom(expn):
      return 1
    elif isinstance(expn,list):
        return max(map(expnType, expn))    #apply(max,map(ExpnType,expn))
    elif  is_sqrt(expn):
        if isinstance(expn.args[0],Rational): #type(op(1,expn),'rational')
            return 1
        else:
            return max(2,expnType(expn.args[0]))  #max(2,ExpnType(op(1,expn)))
    elif isinstance(expn,Pow):   #type(expn,'^')
        if isinstance(expn.args[1],Integer):  #type(op(2,expn),'integer')
            return expnType(expn.args[0])   #ExpnType(op(1,expn))
        elif isinstance(expn.args[1],Rational):  #type(op(2,expn),'rational')
            if isinstance(expn.args[0],Rational): #type(op(1,expn),'rational')
                return 1
            else:
                return max(2,expnType(expn.args[0]))  #max(2,ExpnType(op(1,expn)))
        else:
            return max(3,expnType(expn.args[0]),expnType(expn.args[1])) #max(3,ExpnType(op(1,expn)),Expr
    elif isinstance(expn,Add) or isinstance(expn,Mul): #type(expn,'+') or type(expn,'*')
        m1 = expnType(expn.args[0])
        m2 = expnType(list(expn.args[1:]))
        return max(m1,m2)  #max(ExpnType(op(1,expn)),max(ExpnType(rest(expn))))
    elif is_elementary_function(expn.func):  #ElementaryFunctionQ(op(0,expn))
        return max(3,expnType(expn.args[0]))  #max(3,ExpnType(op(1,expn)))
    elif is_special_function(expn.func): #SpecialFunctionQ(op(0,expn))
```

```
        m1 = max(map(expnType, list(expn.args)))
        return max(4,m1)    #max(4,apply(max,map(ExpnType,[op(expn)])))
    elif is_hypergeometric_function(expn.func): #HypergeometricFunctionQ(op(0,expn))
        m1 = max(map(expnType, list(expn.args)))
        return max(5,m1)    #max(5,apply(max,map(ExpnType,[op(expn)])))
    elif is_appell_function(expn.func):
        m1 = max(map(expnType, list(expn.args)))
        return max(6,m1)    #max(5,apply(max,map(ExpnType,[op(expn)])))
    elif isinstance(expn,RootSum):
        m1 = max(map(expnType, list(expn.args))) #Apply[Max,Append[Map[ExpnType,Apply[List,expn]],7]],
        return max(7,m1)
    elif str(expn).find("Integral") != -1:
        m1 = max(map(expnType, list(expn.args)))
        return max(8,m1)    #max(5,apply(max,map(ExpnType,[op(expn)])))
    else:
        return 9

#main function
def grade_antiderivative(result,optimal):

    #print ("Enter grade_antiderivative for sagemath")
    #print("Enter grade_antiderivative, result=",result," optimal=",optimal)

    leaf_count_result  = leaf_count(result)
    leaf_count_optimal = leaf_count(optimal)

    #print("leaf_count_result=",leaf_count_result)
    #print("leaf_count_optimal=",leaf_count_optimal)

    expnType_result  = expnType(result)
    expnType_optimal = expnType(optimal)

    if str(result).find("Integral") != -1:
        grade = "F"
        grade_annotation =""
    else:
        if expnType_result <= expnType_optimal:
            if result.has(I):
                if optimal.has(I): #both result and optimal complex
                    if leaf_count_result <= 2*leaf_count_optimal:
                        grade = "A"
```

```
                grade_annotation =""
            else:
                grade = "B"
                grade_annotation ="Both result and optimal contain complex but leaf count of result is lar
        else: #result contains complex but optimal is not
            grade = "C"
            grade_annotation ="Result contains complex when optimal does not."
    else: # result do not contain complex, this assumes optimal do not as well
        if leaf_count_result <= 2*leaf_count_optimal:
            grade = "A"
            grade_annotation =""
        else:
            grade = "B"
            grade_annotation ="Leaf count of result is larger than twice the leaf count of optimal. "+str(
else:
    grade = "C"
    grade_annotation ="Result contains higher order function than in optimal. Order "+str(ExpnType


#print("Before returning. grade=",grade, " grade_annotation=",grade_annotation)

return grade, grade_annotation
```

4.1.4 SageMath grading function

```
#Dec 24, 2019. Nasser: Ported original Maple grading function by
#             Albert Rich to use with Sagemath. This is used to
#             grade Fricas, Giac and Maxima results.
#Dec 24, 2019. Nasser: Added 'exp_integral_e' and 'sng', 'sin_integral'
#             'arctan2','floor','abs','log_integral'
#June 4, 2022  Made default grade_annotation "none" instead of "" due
#             issue later when reading the file.
#July 14, 2022. Added ellipticF. This is until they fix sagemath, then remove it.

from sage.all import *
from sage.symbolic.operators import add_vararg, mul_vararg

debug=False;

def tree_size(expr):
```

```
    r"""
    Return the tree size of this expression.
    """
    #print("Enter tree_size, expr is ",expr)

    if expr not in SR:
        # deal with lists, tuples, vectors
        return 1 + sum(tree_size(a) for a in expr)
    expr = SR(expr)
    x, aa = expr.operator(), expr.operands()
    if x is None:
        return 1
    else:
        return 1 + sum(tree_size(a) for a in aa)

def is_sqrt(expr):
    if expr.operator() == operator.pow:    #isinstance(expr,Pow):
        if expr.operands()[1]==1/2: #expr.args[1] == Rational(1,2):
            if debug: print ("expr is sqrt")
            return True
        else:
            return False
    else:
        return False

def is_elementary_function(func):
    #debug=False
    m = func.name() in ['exp','log','ln',
            'sin','cos','tan','cot','sec','csc',
            'arcsin','arccos','arctan','arccot','arcsec','arccsc',
            'sinh','cosh','tanh','coth','sech','csch',
            'arcsinh','arccosh','arctanh','arccoth','arcsech','arccsch','sgn',
        'arctan2','floor','abs'
        ]
    if debug:
        if m:
            print ("func ", func , " is elementary_function")
        else:
            print ("func ", func , " is NOT elementary_function")
```

```
    return m

def is_special_function(func):
    #debug=False
    if debug:
        print ("type(func)=", type(func))

    m= func.name() in ['erf','erfc','erfi','fresnel_sin','fresnel_cos','Ei',
          'Ei','Li','Si','sin_integral','Ci','cos_integral','Shi','sinh_integral'
          'Chi','cosh_integral','gamma','log_gamma','psi,zeta',
          'polylog','lambert_w','elliptic_f','elliptic_e','ellipticF',
          'elliptic_pi','exp_integral_e','log_integral']

    if debug:
        print ("m=",m)
        if m:
            print ("func ", func ," is special_function")
        else:
            print ("func ", func ," is NOT special_function")


    return m


def is_hypergeometric_function(func):
    return func.name() in ['hypergeometric','hypergeometric_M','hypergeometric_U']

def is_appell_function(func):
    return func.name() in ['hypergeometric']    #[appellf1] can't find this in sagemath

def is_atom(expn):

    #debug=False
    if debug:
        print ("Enter is_atom, expn=",expn)

    if not hasattr(expn, 'parent'):
        return False


    #thanks to answer at https://ask.sagemath.org/question/49179/what-is-sagemath-equivalent-to-atomic
```

```
    try:
        if expn.parent() is SR:
            return expn.operator() is None
        if expn.parent() in (ZZ, QQ, AA, QQbar):
            return expn in expn.parent() # Should always return True
        if hasattr(expn.parent(),"base_ring") and hasattr(expn.parent(),"gens"):
            return expn in expn.parent().base_ring() or expn in expn.parent().gens()

        return False

    except AttributeError as error:
        print("Exception,AttributeError in  is_atom")
        print ("cought exception" , type(error).__name__ )
        return False


def expnType(expn):

    if debug:
        print (">>>>>Enter expnType, expn=", expn)
        print (">>>>>is_atom(expn)=", is_atom(expn))

    if is_atom(expn):
        return 1
    elif type(expn)==list:   #isinstance(expn,list):
        return max(map(expnType, expn))   #apply(max,map(ExpnType,expn))
    elif  is_sqrt(expn):
        if  type(expn.operands()[0])==Rational: #type(isinstance(expn.args[0],Rational):
            return 1
        else:
            return max(2,expnType(expn.operands()[0]))  #max(2,expnType(expn.args[0]))
    elif expn.operator() == operator.pow:   #isinstance(expn,Pow)
        if type(expn.operands()[1])==Integer:  #isinstance(expn.args[1],Integer)
            return expnType(expn.operands()[0])   #expnType(expn.args[0])
        elif type(expn.operands()[1])==Rational:  #isinstance(expn.args[1],Rational)
            if type(expn.operands()[0])==Rational: #isinstance(expn.args[0],Rational)
                return 1
            else:
                return max(2,expnType(expn.operands()[0]))  #max(2,expnType(expn.args[0]))
        else:
            return max(3,expnType(expn.operands()[0]),expnType(expn.operands()[1])) #max(3,expnType(expr
```

```
    elif expn.operator() == add_vararg or expn.operator() == mul_vararg: #isinstance(expn,Add) or isins
        m1 = expnType(expn.operands()[0]) #expnType(expn.args[0])
        m2 = expnType(expn.operands()[1:]) #expnType(list(expn.args[1:]))
        return max(m1,m2)  #max(ExpnType(op(1,expn)),max(ExpnType(rest(expn))))
    elif is_elementary_function(expn.operator()):  #is_elementary_function(expn.func)
        return max(3,expnType(expn.operands()[0]))
    elif is_special_function(expn.operator()): #is_special_function(expn.func)
        m1 = max(map(expnType, expn.operands()))      #max(map(expnType, list(expn.args)))
        return max(4,m1)   #max(4,m1)
    elif is_hypergeometric_function(expn.operator()): #is_hypergeometric_function(expn.func)
        m1 = max(map(expnType, expn.operands()))      #max(map(expnType, list(expn.args)))
        return max(5,m1)   #max(5,m1)
    elif is_appell_function(expn.operator()):
        m1 = max(map(expnType, expn.operands()))      #max(map(expnType, list(expn.args)))
        return max(6,m1)   #max(6,m1)
    elif str(expn).find("Integral") != -1: #this will never happen, since it
            #is checked before calling the grading function that is passed.
            #but kept it here.
        m1 = max(map(expnType, expn.operands()))      #max(map(expnType, list(expn.args)))
        return max(8,m1)   #max(5,apply(max,map(ExpnType,[op(expn)])))
    else:
        return 9

#main function
def grade_antiderivative(result,optimal):


    if debug:
        print ("Enter grade_antiderivative for sagemath")
        print("Enter grade_antiderivative, result=",result)
        print("Enter grade_antiderivative, optimal=",optimal)
        print("type(anti)=",type(result))
        print("type(optimal)=",type(optimal))

    leaf_count_result  = tree_size(result) #leaf_count(result)
    leaf_count_optimal = tree_size(optimal) #leaf_count(optimal)

    #if debug: print ("leaf_count_result=", leaf_count_result, "leaf_count_optimal=",leaf_count_optimal)


    expnType_result  = expnType(result)
```

```
expnType_optimal = expnType(optimal)

if debug: print ("expnType_result=", expnType_result, "expnType_optimal=",expnType_optimal)

if expnType_result <= expnType_optimal:
    if result.has(I):
        if optimal.has(I): #both result and optimal complex
            if leaf_count_result <= 2*leaf_count_optimal:
                grade = "A"
                grade_annotation =" "
            else:
                grade = "B"
                grade_annotation ="Both result and optimal contain complex but leaf count of result is larger
        else: #result contains complex but optimal is not
            grade = "C"
            grade_annotation ="Result contains complex when optimal does not."
    else: # result do not contain complex, this assumes optimal do not as well
        if leaf_count_result <= 2*leaf_count_optimal:
            grade = "A"
            grade_annotation =" "
        else:
            grade = "B"
            grade_annotation ="Leaf count of result is larger than twice the leaf count of optimal. "+str(lea
else:
    grade = "C"
    grade_annotation ="Result contains higher order function than in optimal. Order "+str(expnType_re


print("Before returning. grade=",grade, " grade_annotation=",grade_annotation)

return grade, grade_annotation
```