Computer algebra independent integration tests. Elementary algebraic integrals

Nasser M. Abbasi

September 18, 2021 Compiled on September 18, 2021 at 6:44pm

Contents

Chapter 1

Introduction

This report gives the result of running the computer algebra independent integration tests. Elementary Algebraic integrals version.

This is a subset of Rubi test suite thanks to Albert Rich, which includes only the algebraic integrals with elementray optimal antiderivatives. It also includes a subset of a test file provided thanks to Sam Blake.

The following zip files contain the raw integrals used in this report.

Mathematica format Mathematica_syntax_CAS_integration_elementary_version.zip

Maple and Mupad format Maple_syntax_CAS_integration_elementary_version.zip

Sympy format SYMPY_syntax_CAS_integration_elementary_version.zip

Sage math format SAGE_syntax_CAS_integration_elementary_version.zip

The current number of problems in this test suite is [22622].

1.1 Listing of CAS systems tested

The following systems were tested at this time.

1. Mathematica 12.3.1 (64 bit) on windows 10.
2. Rubi 4.16.1 in Mathematica 12 on windows 10.
3. Maple 2021.1 (64 bit) on windows 10.
4. Maxima 5.44 on Linux. (via sagemath 9.3)
5. Fricas 1.3.7 on Linux (via sagemath 9.3)
6. Sympy 1.8 under Python 3.8.8 using Anaconda distribution.
7. Giac/Xcas 1.7 on Linux. (via sagemath 9.3)
8. Mupad using Matlab 2021a with Symbolic Math Toolbox Version 8.7 under windows 10 (64 bit)

9. IntegrateAlgebraic under Mathematica 12.3.1 on windows 10. https://github.com/stblake/algebraic_integration.

Maxima, Fricas and Giac/Xcas were called from inside SageMath. This was done using SageMath integrate command by changing the name of the algorithm to use the different CAS systems.

Sympy was called directly using Python.

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. RootSum and RootOf are not allowed.

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.

System	% solved	% Failed
Mathematica	96.65 (21864)	3.35 (758)
Rubi	96.16 (21754)	3.84 (868)
Fricas	93.75 (21207)	6.25 (1415)
Maple	91.89 (20788)	8.11 (1834)
IntegrateAlgebraic	61.28 (13863)	38.72 (8759)
Giac	77.45 (17520)	22.55 (5102)
Mupad	70.46 (15939)	29.54 (6683)
Maxima	68.31 (15452)	31.69 (7170)
Sympy	54.15 (12250)	45.85 (10372)

Table 1.1: Percentage solved for each CAS

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.

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.

Table 1.2: Description of grading applied to integration result

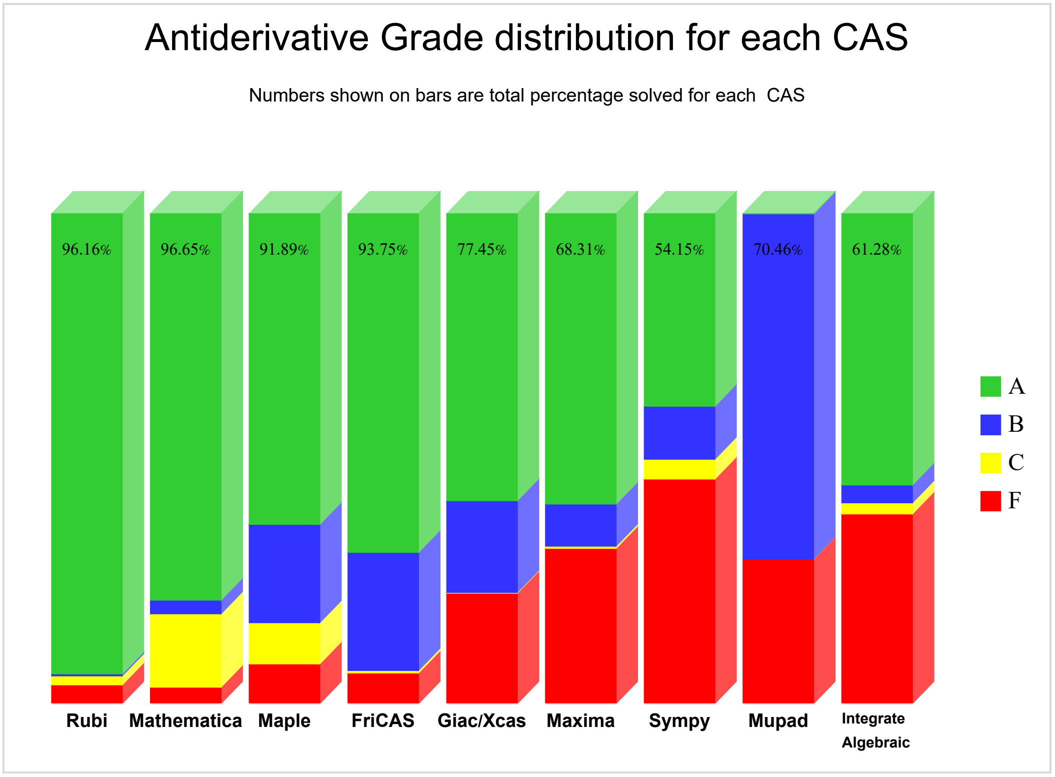
Grading is implemented for all CAS systems in this version except for CAS Mupad where a grade of B is automatically assigned as a place holder for all integrals it completes on time.

The following table summarizes the grading results.

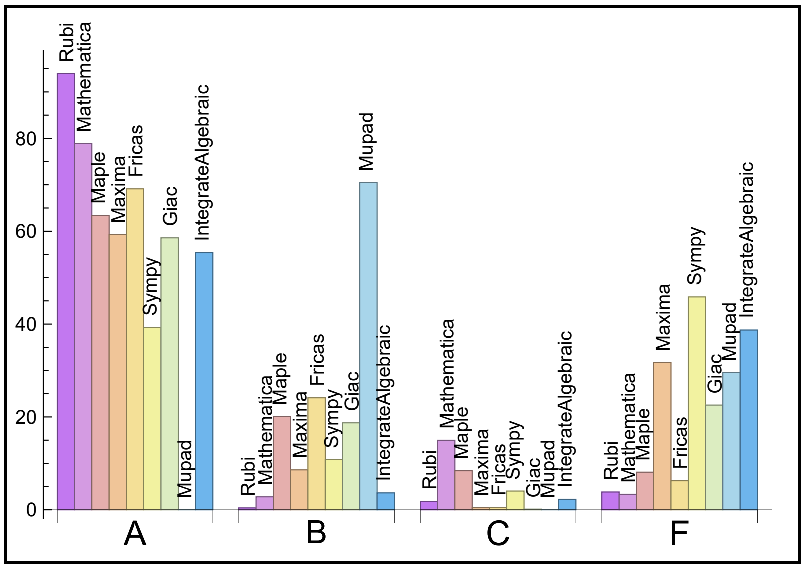
System	% A grade	% B grade	% C grade	% F grade
Rubi	93.93	0.41	1.82	3.84
Mathematica	78.86	2.79	14.98	3.35
IntegrateAlgebraic	55.37	3.64	2.27	38.72
Fricas	69.12	24.12	0.5	6.25
Maple	63.41	20.08	8.41	8.11
Maxima	59.26	8.6	0.45	31.69
Giac	58.57	18.73	0.15	22.55
Sympy	39.29	10.82	4.04	45.85
Mupad	N/A	70.46	0.	29.54

Table 1.3: Antiderivative Grade distribution for each CAS

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



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



1.2.1 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.

System	Mean time (sec)	Mean size	Normalized mean	Median size	Normalized median
Rubi	0.25	136.14	1.1	94.	1.
Mathematica	0.3	219.07	2.1	74.	0.92
Maple	0.34	883.32	4.8	92.	1.06
Maxima	1.31	140.62	1.32	76.	0.98
Fricas	2.42	410.66	2.46	120.	1.4
Sympy	9.7	259.23	2.75	83.	1.2
Giac	0.98	285.73	1.83	97.	1.06
Mupad	1.86	880.96	3.34	77.	0.98
IntegrateAlgebraic	2.84	246.87	1.54	97.	1.

Table 1.4: Time and leaf size performance for each CAS

1.3 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 easy to locate the problem. In the future, a direct link will be added as well. To help make the table fit, `Mathematica` was abbreviated to `MMA` and `IntegrateAlgebraic` to `I.A.`

Table 1.5: Maximum leaf size ratio for each CAS against the optimal result

file #	Rubi	MMA	Maple	Maxima	FriCAS	Sympy	Giac	Mupad	I.A.
1	7.1 (369)	23.8 (1217)	30.9 (1217)	32.9 (1217)	32.9 (1217)	136.1 (671)	34. (1217)	38.1 (1217)	10. (1041)
2	1.3 (329)	16.5 (962)	22.6 (962)	22.2 (1577)	21.8 (962)	76.7 (2548)	46.7 (802)	328.9 (2161)	46.2 (665)
3	2.6 (44)	2.4 (44)	4.6 (35)	2.8 (33)	10.8 (21)	49.2 (33)	10. (33)	23.6 (21)	2.6 (43)
4	1. (1)	1.5 (17)	11. (25)	4. (25)	9. (25)	59.8 (27)	19.9 (25)	1.7 (3)	3.2 (9)
5	2.6 (35)	4.9 (45)	23.3 (53)	1.7 (35)	6.4 (7)	5.3 (35)	16.5 (52)	330.8 (32)	15.4 (51)
6	8.2 (583)	6.9 (582)	7.9 (196)	10. (196)	10. (196)	55.3 (519)	8. (425)	10.1 (196)	3.6 (425)
7	1.2 (134)	6.4 (94)	147.4 (69)	4.4 (73)	19.9 (130)	10.2 (24)	5.9 (69)	37.7 (26)	30.6 (88)

Continued on next page

Table 1.5 – continued from previous page

file #	Rubi	MMA	Maple	Maxima	FriCAS	Sympy	Giac	Mupad	I.A.
8	1. (578)	11.4 (708)	46.7 (736)	3.1 (313)	17.2 (811)	28.8 (322)	8.6 (535)	224.1 (484)	2.1 (692)
9	1. (1)	2.6 (29)	15.2 (23)	1.3 (15)	8.1 (26)	3. (21)	3. (30)	1.7 (21)	12.4 (29)
10	1. (1)	1.1 (13)	10.4 (9)	2. (9)	7. (9)	43. (10)	13.8 (9)	2.5 (1)	1.3 (13)
11	1.2 (169)	1.9 (45)	2. (158)	3.6 (157)	5.2 (26)	47.9 (55)	4. (153)	1.8 (129)	1.4 (43)
12	8.4 (2158)	13.4 (2357)	141.8 (2357)	13.2 (1809)	23. (2357)	84.8 (1312)	28.4 (2260)	16.4 (2357)	11.3 (2079)
13	4.3 (81)	4.6 (236)	17.9 (168)	4. (35)	15.1 (168)	27.8 (141)	6. (183)	62.5 (109)	2.8 (198)
14	4.2 (492)	12.3 (672)	77.4 (336)	29.1 (684)	17.4 (237)	36.6 (124)	9.8 (673)	78.2 (320)	16.5 (645)
15	1. (1)	0.9 (9)	51.1 (9)	2.5 (9)	28.4 (9)	78.7 (3)	49.5 (10)	7.2 (9)	0. (-1)
16	1. (1)	3.8 (45)	10.4 (43)	10. (43)	47.2 (62)	14.9 (417)	8.1 (424)	34.3 (124)	3. (6)
17	1. (1)	10. (231)	51.5 (207)	11.2 (251)	10.2 (251)	10. (231)	10.6 (234)	12.5 (251)	8. (251)
18	1. (1)	6.4 (228)	4.9 (220)	3.2 (114)	4. (220)	21.6 (220)	6.3 (220)	3.4 (190)	1.9 (28)
19	2.8 (64)	3.9 (25)	5.8 (55)	2.2 (64)	7.2 (108)	16.4 (44)	3.1 (55)	3.5 (25)	8.1 (25)
20	2. (2088)	23.9 (1971)	70.8 (2020)	28.7 (477)	36.4 (1962)	67.3 (1143)	39.4 (1707)	209.3 (1969)	228.8 (1646)
21	1.3 (1297)	7.2 (2335)	82.2 (1035)	50.9 (1940)	46.2 (1278)	64.1 (936)	28.9 (1401)	179.2 (1977)	148. (1990)
22	2.1 (576)	58.6 (328)	116.1 (552)	5.9 (381)	34. (418)	71.6 (629)	20. (631)	201.9 (561)	228.8 (316)
23	1. (1)	10.3 (6)	425.1 (75)	2.7 (92)	30.2 (109)	1.2 (16)	13.3 (5)	3. (97)	70.4 (110)
24	1. (129)	9.7 (37)	14197.2 (12)	6.6 (27)	30.2 (117)	8.6 (14)	5.8 (37)	110.2 (16)	5.6 (24)
25	1.8 (76)	42.8 (200)	421. (258)	89. (258)	123.4 (258)	114.1 (258)	119.2 (258)	101.3 (258)	40.9 (196)
26	1.7 (464)	4.3 (295)	9.5 (688)	5.4 (343)	25.4 (844)	28.5 (856)	13.7 (688)	91.3 (855)	26.2 (752)
27	1.7 (154)	13.9 (210)	50.7 (162)	6.5 (76)	33.5 (97)	15.8 (206)	110.2 (158)	360. (197)	2.5 (164)
28	1.7 (274)	32.6 (281)	26. (114)	5.6 (50)	55.6 (227)	47.5 (156)	35.3 (231)	123.2 (231)	28.8 (238)
29	1. (59)	1.5 (25)	15.8 (54)	1.4 (106)	2.6 (46)	43. (11)	21.7 (25)	107.5 (41)	1. (103)
30	1.6 (133)	2.4 (134)	13.8 (37)	1.6 (129)	48.1 (58)	27.3 (39)	20.4 (58)	94.8 (128)	3. (132)
31	1. (1)	3. (2)	2.8 (1)	0. (-1)	4.2 (2)	0. (-1)	2.5 (9)	3.8 (1)	1.1 (2)
32	2.1 (141)	5.8 (496)	54.7 (496)	6.3 (496)	46.7 (524)	21.4 (493)	46.6 (492)	99.1 (261)	17.5 (114)
33	1. (1)	1.9 (26)	2.7 (37)	1.8 (44)	12.2 (37)	42.2 (44)	15.3 (37)	116. (41)	0. (-1)
34	1. (59)	12.9 (72)	2909.3 (71)	88.7 (74)	90.4 (71)	82.9 (71)	73.6 (74)	165.9 (43)	64.6 (74)
35	1. (1)	1. (4)	1.7 (3)	2.1 (8)	2.2 (8)	3.2 (3)	3.3 (8)	215.3 (1)	1.1 (2)
36	1. (1)	1.7 (99)	4. (72)	1.1 (72)	9.5 (102)	18.1 (72)	12.1 (79)	41.4 (99)	1.4 (112)
37	6.2 (420)	11.6 (158)	1223.1 (188)	42.3 (59)	93.1 (188)	84.3 (188)	27.1 (198)	166.7 (20)	2.8 (98)
38	4.1 (689)	172.1 (699)	3059.3 (699)	5.1 (357)	40.5 (574)	58.4 (55)	16.9 (191)	54.2 (187)	51.8 (192)
39	135.9 (517)	2947.4 (1435)	2183.5 (589)	6.9 (940)	90.4 (889)	99.5 (780)	4.5 (2303)	323.4 (2346)	2.4 (2379)

1.4 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 208 tests. Each tests corresponds to one input file.

Table 1.6: Pass/Fail per test file for each CAS

Test #	Rubi		MMA		Maple		Maxima		FriCAS		Sympy		Giac		Mupad		I.A.	
	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail
1	1603	0	1603	0	1532	71	1328	275	1603	0	1148	455	1276	327	1241	362	1016	587
2	2554	0	2554	0	2483	71	2048	506	2535	19	1483	1071	2403	151	1884	670	1469	1085

Continued on next page

Table 1.6 – continued from previous page

Test #	Rubi		MMA		Maple		Maxima		FriCAS		Sympy		Giac		Mupad		I.A.	
	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail
3	48	0	48	0	48	0	38	10	46	2	40	8	41	7	48	0	36	12
4	28	0	28	0	28	0	16	12	28	0	13	15	28	0	4	24	24	4
5	60	0	60	0	60	0	27	33	44	16	14	46	33	27	40	20	58	2
6	697	0	697	0	648	49	632	65	674	23	664	33	616	81	627	70	415	282
7	158	0	158	0	132	26	79	79	141	17	62	96	108	50	65	93	83	75
8	884	0	884	0	856	28	682	202	856	28	516	368	822	62	730	154	550	334
9	30	0	30	0	30	0	22	8	28	2	21	9	26	4	22	8	8	22
10	14	0	14	0	14	0	14	0	14	0	13	1	14	0	14	0	2	12
11	170	0	170	0	170	0	170	0	158	12	136	34	170	0	129	41	88	82
12	2466	0	2451	15	2282	184	2196	270	2370	96	2139	327	1990	476	2092	374	1499	967
13	239	0	239	0	176	63	167	72	213	26	113	126	134	105	168	71	165	74
14	702	0	702	0	522	180	391	311	663	39	301	401	535	167	531	171	515	187
15	12	0	12	0	12	0	12	0	12	0	3	9	11	1	12	0	0	12
16	427	0	427	0	424	3	422	5	341	86	285	142	420	7	425	2	17	410
17	340	0	340	0	291	49	153	187	256	84	114	226	238	102	176	164	208	132
18	228	0	228	0	226	2	212	16	228	0	126	102	213	15	197	31	142	86
19	113	0	113	0	113	0	108	5	113	0	47	66	106	7	113	0	71	42
20	2119	0	2119	0	2097	22	1428	691	2096	23	1013	1106	1678	441	1582	537	1178	941
21	2337	0	2337	0	2337	0	1720	617	2299	38	1153	1184	2153	184	1684	653	1350	987
22	632	0	632	0	631	1	328	304	591	41	174	458	276	356	271	361	486	146
23	118	0	118	0	118	0	67	51	111	7	43	75	90	28	53	65	72	46
24	143	0	142	1	141	2	15	128	69	74	11	132	50	93	19	124	131	12
25	375	0	375	0	375	0	290	85	330	45	141	234	335	40	195	180	208	167
26	858	0	858	0	857	1	688	170	846	12	463	395	799	59	685	173	511	347
27	212	0	212	0	212	0	109	103	206	6	159	53	173	39	178	34	23	189
28	300	0	300	0	300	0	173	127	256	44	106	194	264	36	214	86	149	151
29	106	0	102	4	106	0	83	23	83	23	47	59	100	6	106	0	5	101
30	143	0	143	0	143	0	73	70	115	28	79	64	139	4	143	0	14	129
31	9	0	9	0	9	0	0	9	9	0	0	9	5	4	1	8	9	0
32	549	0	549	0	496	53	303	246	535	14	282	267	412	137	360	189	219	330
33	44	0	44	0	44	0	12	32	42	2	36	8	32	12	44	0	0	44
34	119	0	119	0	119	0	66	53	105	14	69	50	107	12	119	0	10	109
35	8	0	5	3	2	6	2	6	7	1	1	7	4	4	5	3	2	6
36	130	0	129	1	128	2	24	106	129	1	53	77	106	24	72	58	69	61
37	487	3	490	0	489	1	409	81	431	59	430	60	421	69	482	8	12	478
38	705	9	699	15	627	87	375	339	687	27	205	509	474	240	407	307	623	91
39	1587	856	1724	719	1510	933	570	1873	1937	506	547	1896	718	1725	801	1642	2426	17

1.5 Timing

The command `AboluteTiming[]` 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 has 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 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 is not counted in the final statistics.

1.6 Verification

A verification phase was applied on the result of integration for Rubi and Mathematica. 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 produced was correct.

Verification phase has 3 minutes time out. An integral whose result was not verified could still be correct. Further investigation is needed on those integrals which failed verifications. Such integrals are marked in the summary table below and also in each integral separate section so they are easy to identify and locate.

1.7 Important notes about some of the results

1.7.1 Important note about Maxima results

Since these integrals are run in a batch mode, using an automated script, and by using sagemath (SageMath uses Maxima), then any integral where Maxima needs an interactive response from the user to answer a question during evaluation of the integral in order to complete the integration, will fail and is counted as failed.

The exception raised is `ValueError`. Therefore Maxima result below is lower than what could result if Maxima was run directly and each question Maxima asks 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 pecrentage can be higher or lower depending on the specific input test file.

Such integrals can be indentified by looking at the output of the integration in each section for Maxima. The exception message will indicate of the error is due to the interactive question being asked or not.

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 loading of Maxima `abs_integrate` was found to cause some problem. 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.

1.7.2 Important note about FriCAS and Giac/XCAS results

There are Few integrals which failed due to SageMath not able to translate the result back to SageMath syntax and not because these CAS system were not able to do the integrations.

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.

Hopefully the next version of SageMath will have complete translation of FriCAS and XCAS syntax and I will re-run all the tests again when this happens.

1.7.3 Important note about finding leaf size of antiderivative

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

The other CAS systems (SageMath and Sympy) do not have special buildin function for this purpose at this time. Therefore the leaf size for Fricas and Sympy and Giac antiderivatives is 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 is called directly from Python, the following code is 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
```

1.7.4 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.

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.

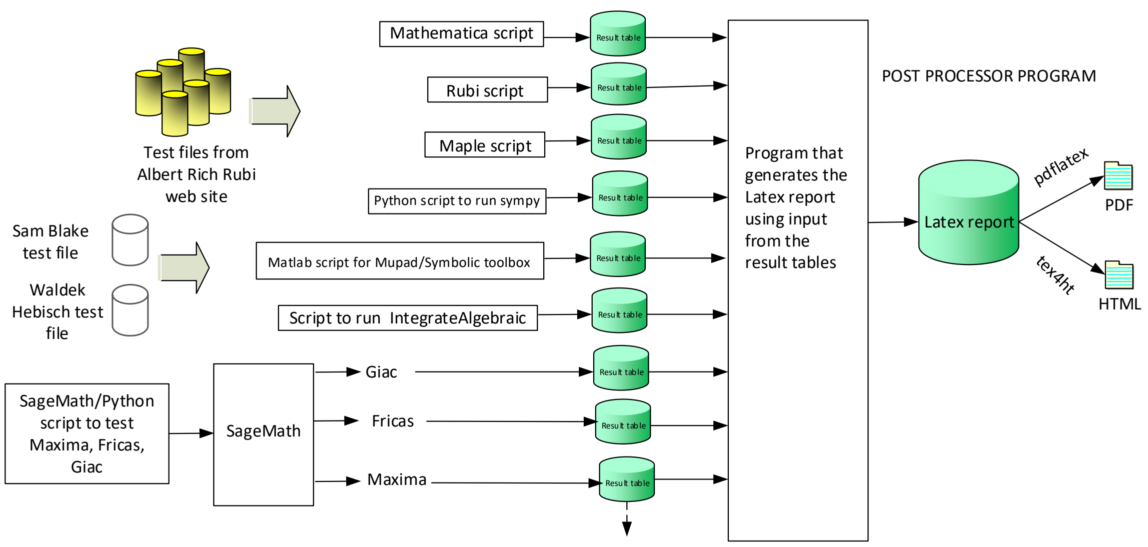
The following is an example of using Matlab's symbolic toolbox (Mupad) to solve an integral

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

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

1.8 Design of the test system

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


Mathematica script
Rubi script
Maple script
Python script to run sympy
Matlab script for Mupad/Symbolic toolbox
Script to run IntegrateAlgebraic
Test files from Albert Rich Rubi web site
Sam Blake test file
Waldek Hebisch test file
SageMath/Python script to test Maxima, Fricas, Giac
SageMath
Giac
Fricas
Maxima
Result table
POST PROCESSOR PROGRAM
Program that generates the Latex report using input from the result tables
Latex report
pdflatex
tex4ht
PDF
HTML

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"

    The following field present only in Rubi and Mathematica Tables

13. integer. 1 if result was verified or 0 if not verified.

    The following fields present only in Rubi Tables

14. integer. Number of rules used.
15. integer. Integrand leaf size.
16. real number. Ratio of field 14 over field 15
17. integer. 1 if result was verified or 0 if not verified.
18. String of form "{n,n,..}" which is list of the rules used by Rubi

High level overview of the CAS independent integration test build system

Nasser M. Abbasi
May 11, 2021

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. `1_Algebraic_functions/1.1_Binomial_products/1.1.1_Linear/1.1.1.2-a+b_x-^m-c+d_x-^n` [1603]
2. `1_Algebraic_functions/1.1_Binomial_products/1.1.1_Linear/1.1.1.3-a+b_x-^m-c+d_x-^n-e+f_x-^p` [2554]
3. `1_Algebraic_functions/1.1_Binomial_products/1.1.1_Linear/1.1.1.4-a+b_x-^m-c+d_x-^n-e+f_x-^p-g+h_x-^q` [48]
4. `1_Algebraic_functions/1.1_Binomial_products/1.1.1_Linear/1.1.1.5_P-x-a+b_x-^m-c+d_x-^n` [28]
5. `1_Algebraic_functions/1.1_Binomial_products/1.1.1_Linear/1.1.1.6_P-x-a+b_x-^m-c+d_x-^n-e+f_x-^p` [60]
6. `1_Algebraic_functions/1.1_Binomial_products/1.1.2_Quadratic/1.1.2.2-c_x-^m-a+b_x^2-^p` [697]
7. `1_Algebraic_functions/1.1_Binomial_products/1.1.2_Quadratic/1.1.2.3-a+b_x^2-^p-c+d_x^2-^q` [158]
8. `1_Algebraic_functions/1.1_Binomial_products/1.1.2_Quadratic/1.1.2.4-e_x-^m-a+b_x^2-^p-c+d_x^2-^q` [884]
9. `1_Algebraic_functions/1.1_Binomial_products/1.1.2_Quadratic/1.1.2.5-a+b_x^2-^p-c+d_x^2-^q-e+f_x^2-^r` [30]

10. `1_Algebraic_functions/1.1_Binomial_products/1.1.2_Quadratic/1.1.2.6-g_x-^m-a+b_x^2-^p-c+d_x^2-^q-e+f_x^2-^r` [14]
11. `1_Algebraic_functions/1.1_Binomial_products/1.1.2_Quadratic/1.1.2.8_P-x-c_x-^m-a+b_x^2-^p` [170]
12. `1_Algebraic_functions/1.1_Binomial_products/1.1.3_General/1.1.3.2-c_x-^m-a+b_x^n-^p` [2466]
13. `1_Algebraic_functions/1.1_Binomial_products/1.1.3_General/1.1.3.3-a+b_x^n-^p-c+d_x^n-^q` [239]
14. `1_Algebraic_functions/1.1_Binomial_products/1.1.3_General/1.1.3.4-e_x-^m-a+b_x^n-^p-c+d_x^n-^q` [702]
15. `1_Algebraic_functions/1.1_Binomial_products/1.1.3_General/1.1.3.6-g_x-^m-a+b_x^n-^p-c+d_x^n-^q-e+f_x^n-^r` [12]
16. `1_Algebraic_functions/1.1_Binomial_products/1.1.3_General/1.1.3.8_P-x-c_x-^m-a+b_x^n-^p` [427]
17. `1_Algebraic_functions/1.1_Binomial_products/1.1.4_Improper/1.1.4.2-c_x-^m-a_x^j+b_x^n-^p` [340]
18. `1_Algebraic_functions/1.1_Binomial_products/1.1.4_Improper/1.1.4.3-e_x-^m-a_x^j+b_x^k-^p-c+d_x^n-^q` [228]
19. `1_Algebraic_functions/1.2_Trinomial_products/1.2.1_Quadratic/1.2.1.1-a+b_x+c_x^2-^p` [113]
20. `1_Algebraic_functions/1.2_Trinomial_products/1.2.1_Quadratic/1.2.1.2-d+e_x-^m-a+b_x+c_x^2-^p` [2119]
21. `1_Algebraic_functions/1.2_Trinomial_products/1.2.1_Quadratic/1.2.1.3-d+e_x-^m-f+g_x-a+b_x+c_x^2-^p` [2337]
22. `1_Algebraic_functions/1.2_Trinomial_products/1.2.1_Quadratic/1.2.1.4-d+e_x-^m-f+g_x-^n-a+b_x+c_x^2-^p` [632]
23. `1_Algebraic_functions/1.2_Trinomial_products/1.2.1_Quadratic/1.2.1.5-a+b_x+c_x^2-^p-d+e_x+f_x^2-^q` [118]
24. `1_Algebraic_functions/1.2_Trinomial_products/1.2.1_Quadratic/1.2.1.6-g+h_x-^m-a+b_x+c_x^2-^p-d+e_x+f_x^2-^q` [143]
25. `1_Algebraic_functions/1.2_Trinomial_products/1.2.1_Quadratic/1.2.1.9_P-x-d+e_x-^m-a+b_x+c_x^2-^p` [375]
26. `1_Algebraic_functions/1.2_Trinomial_products/1.2.2_Quartic/1.2.2.2-d_x-^m-a+b_x^2+c_x^4-^p` [858]
27. `1_Algebraic_functions/1.2_Trinomial_products/1.2.2_Quartic/1.2.2.3-d+e_x^2-^m-a+b_x^2+c_x^4-^p` [212]

28. `1_Algebraic_functions/1.2_Trinomial_products/1.2.2_Quartic/1.2.2.4-f_x-^m-d+e_x^2-^q-a+b_x^2+c_x^4-^p` [300]
29. `1_Algebraic_functions/1.2_Trinomial_products/1.2.2_Quartic/1.2.2.5_P-x-a+b_x^2+c_x^4-^p` [106]
30. `1_Algebraic_functions/1.2_Trinomial_products/1.2.2_Quartic/1.2.2.6_P-x-d_x-^m-a+b_x^2+c_x^4-^p` [143]
31. `1_Algebraic_functions/1.2_Trinomial_products/1.2.2_Quartic/1.2.2.7_P-x-d+e_x^2-^q-a+b_x^2+c_x^4-^p` [9]
32. `1_Algebraic_functions/1.2_Trinomial_products/1.2.3_General/1.2.3.2-d_x-^m-a+b_x^n+c_x^-2_n-^p` [549]
33. `1_Algebraic_functions/1.2_Trinomial_products/1.2.3_General/1.2.3.3-d+e_x^n-^q-a+b_x^n+c_x^-2_n-^p` [44]
34. `1_Algebraic_functions/1.2_Trinomial_products/1.2.3_General/1.2.3.4-f_x-^m-d+e_x^n-^q-a+b_x^n+c_x^-2_n-^p` [119]
35. `1_Algebraic_functions/1.2_Trinomial_products/1.2.3_General/1.2.3.5_P-x-d_x-^m-a+b_x^n+c_x^-2_n-^p` [8]
36. `1_Algebraic_functions/1.2_Trinomial_products/1.2.4_Improper/1.2.4.2-d_x-^m-a_x^q+b_x^n+c_x^-2_n-q-^p` [130]
37. `1_Algebraic_functions/1.3_Miscellaneous/1.3.1_Rational_functions` [490]
38. `1_Algebraic_functions/1.3_Miscellaneous/1.3.2_Algebraic_functions` [714]
39. `9_Blake_problems` [2443]

Chapter 3

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.

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

3.1 Mathematica and Rubi grading function

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

(* ::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_] :=
  If[ExpnType[result]<=ExpnType[optimal],
    If[FreeQ[result,Complex] || Not[FreeQ[optimal,Complex]],
      If[LeafCount[result]<=2*LeafCount[optimal],
        "A",
      "B"],
    "C"],
  If[FreeQ[result,Integrate] && FreeQ[result,Int],
    "C",
  "F"]]


(* ::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]
```

3.2 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

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";
      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";
  end if;


  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;
            #both result and optimal complex
            if leaf_count_result<=2*leaf_count_optimal then
               return "A";
            else
               return "B";
            end if
         else #result contains complex but optimal is not
            if debug then
               print("result contains complex but optimal is not");
            fi;
            return "C";
         end if
      else # result do not contain complex
           # this assumes optimal do not as well
           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";
           end if
      end if
  else #ExpnType(result) > ExpnType(optimal)
      if debug then
         print("ExpnType(result) > ExpnType(optimal)");
      fi;
      return "C";
  end if

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:
```

3.3 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)),ExpnType(op(2,expn)))
    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):

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

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

    if str(result).find("Integral") != -1:
        return "F"

    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:
                    return "A"
                else:
                    return "B"
            else: #result contains complex but optimal is not
                return "C"
        else: # result do not contain complex, this assumes optimal do not as well
            if leaf_count_result <= 2*leaf_count_optimal:
                return "A"
            else:
                return "B"
    else:
        return "C"
```

3.4 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'

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.
    """
    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',
           '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")


    #thanks to answer at https://ask.sagemath.org/question/49179/what-is-sagemath-
    equivalent-to-atomic-type-in-maple/
    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:
        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(expn.operands()[0]),expnType(expn.operands()[1]))
    elif expn.operator() == add_vararg or expn.operator() == mul_vararg: #isinstance(
    expn,Add) or isinstance(expn,Mul)
        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")

    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:
                return "A"
            else:
                return "B"
        else: #result contains complex but optimal is not
            return "C"
    else: # result do not contain complex, this assumes optimal do not as well
        if leaf_count_result <= 2*leaf_count_optimal:
            return "A"
        else:
            return "B"
else:
    return "C"
```