

Computer Algebra Support for the Wright ω function

Robert M. Corless and David J. Jeffrey

This paper describes an implementation of the Wright ω function in the computer algebra system Maple. We describe the implementation of the facilities for symbolic differentiation, integration, series generation, and simplification.

Categories and Subject Descriptors: I.1.1 [Symbolic and Algebraic Manipulation]: Expressions and Representations—*Simplification of expressions*; I.1.2 [Symbolic and Algebraic Manipulation]: Algorithms—*nonalgebraic algorithms*

General Terms: Computer Algebra, Problem Solving Environment, Support for Special Functions over \mathbb{C}

Additional Key Words and Phrases: Discontinuity, branch point, branch cut, unwinding number

“The purpose of symbolic computing is insight, not formulas.”
—with apologies to R. W. Hamming

1. INTRODUCTION AND CONTEXT

We begin with an introduction to the Wright ω function, together with some extracts from a Maple package that implements the theorems and formulae of this paper. Some larger fragments of the package are included in the appendices. The numerical evaluation of this function is studied in a separate paper, [Corless and Jeffrey 2004]. While this paper restricts itself to Maple, the code presented is simple and unadorned, and could easily be translated into another computer algebra language.

The purposes of this paper are threefold. First, the paper may encourage the reader to learn something about the Wright ω function, because it is useful (in section 2 we give some representative applications of this function). Second, the paper will provide an archival record of some design decisions and algorithmic ideas for the support of the Wright ω function in a computer algebra system, which may help others make similar decisions. Finally, it is hoped that the paper will provide some insight into the technical details of computer algebra systems for readers whose main interest is general scientific computing.

Ontario Research Centre for Computer Algebra
and the Department of Applied Mathematics
The University of Western Ontario
London, Ontario, CANADA

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 0098-3500/20YY/1200-0001 \$5.00

There are many good papers and books on symbolic computation (e.g. [von zur Gathen and Gerhard 1999; Geddes et al. 1992]), and without doubt the use of symbolic systems is familiar to the reader through his or her own experience. We will presume on that experience, in fact, and not discuss issues of Maple syntax in this paper; instead the reader is pointed to [Corless 2002], which is an accelerated introduction to Maple for practitioners of general scientific computing (and not significantly outdated for Maple 9.5, which we use here).

However, there are some interesting symbolic computation issues, especially to do with simplification and with the representation of mathematical knowledge, that may not be so familiar to the reader. While we will not concentrate on abstractions and generalizations from the particular function implementation under study here, we will take such opportunities as arise to make a few points that highlight some of the deeper issues. The authors have used Maple, Derive, CAMAL, Reduce, Mathematica, Macsyma, and even the computer algebra system on the HP calculators, both in teaching and in research, for a number of years; thus we have spent some time ‘breaking our heads’ on the differences between symbolic computing, mathematics, and numerical computing. We find that the function under study here, the Wright ω function, provides a good vehicle for discussing some of the lessons we have learned the hard way over the years; the most important of which is that *continuity*, or the lack of it, presents significant obstacles to symbolic computation, as indeed it does to general scientific computing.

The Wright ω function is defined in [Corless and Jeffrey 2002] to be

$$\omega(z) = W_{\mathcal{K}(z)}(e^z), \quad (1)$$

where $\mathcal{K}(z) = \lceil (\text{Im}(z) - \pi)/(2\pi) \rceil$ is the *unwinding number*, which has the property that $z = \ln e^z + 2\pi i \mathcal{K}(z)$, and W_k is the k th branch of the Lambert W function, which satisfies $W_k(z) \exp W_k(z) = z$. See [Corless et al. 1996] for the properties of the Lambert W function. The Maple code to implement conversion from ω notation to W notation is

```
‘convert/LambertW’ := proc( expr )
    eval( expr, Wrightomega = (z -> LambertW(unwindK(z),exp(z))) ) )
end proc;
```

Simple code for the unwinding number is

```
unwindK := proc( z )
    if type(z,function) and op(0,z)=ln then 0
    else ceil( (Im(z)-Pi)/(2*Pi) )
    end if
end proc;
```

This code for the unwinding number does not automatically simplify its answers as much as it could. In particular, detecting symbolic simplifications such as $\lceil (\arg x + \pi)/(2\pi) \rceil = 1$ is difficult in the current Maple environment.

It is probable that the unwinding number is of greater importance for general symbolic computation than the Wright ω function is; we have discussed this function in [Corless and Jeffrey 1996; Jeffrey et al. 1996; Bradford et al. 2002], for example, but at the time of writing no satisfactory implementation (together with

its necessary interaction with commands like `solve`) has been completed. We hope to return to this in a future paper.

What follows is an example of the use of these conversion utilities in a Maple session. Throughout the paper, after we present a theorem or formula, we give code fragments for its implementation and then a section of a Maple session demonstrating the use of the code. We first introduce a shorthand notation, that will be used in all Maple examples in this paper:

```
> alias( omega=Wrightomega );
> convert( omega(x), LambertW ) assuming x::real;
       $\omega$ 
      LambertW( $e^x$ )
```

It is proved in [Corless and Jeffrey 2002] that a consequence of the definition of the Wright ω function is the following relation to W_k , going the other way:

$$W_k(z) = \omega(\ln_k z), \quad (2)$$

where $\ln_k z = \ln z + 2\pi i k$, and now k is any integer. Because of this last equation, the Wright ω function has an important advantage over Lambert W , which is that it transfers multi-valued behaviour from the branches of Lambert W to the argument of the single-valued function Wright ω . Maple code to convert from ω to W is as follows.

```
‘convert/Wrightomega’ := proc( expr )
  local LAMBERTW;
  LAMBERTW := proc( n, x )
    if nargs=1 then
      Wrightomega( ln(n) )
    else
      Wrightomega( ln(x) + 2*Pi*I*n )
    end if;
  end proc;

  eval( expr, LambertW=LAMBERTW );

end proc;
> convert( LambertW(3,x), omega );
```

$$\omega(\ln(x) + 6I\pi)$$

Another advantage of ω is that the Taylor series for ω about $z = a$ is (slightly) simpler than that of W . Let $\omega_a = \omega(a)$ and $q_0(w) = w/(1+w)$. Then

$$\omega(z) = \sum_{n \geq 0} \frac{q_n(\omega_a)}{(1 + \omega_a)^{2n-1}} \frac{(z - a)^n}{n!} \quad (3)$$

where

$$q_n(w) = \sum_{k=0}^{n-1} \left\langle\left\langle \begin{matrix} n-1 \\ k \end{matrix} \right\rangle\right\rangle (-1)^k w^{k+1}, \quad \text{for } n \geq 1 \quad (4)$$

where $\left\langle\left\langle \begin{smallmatrix} n \\ k \end{smallmatrix} \right\rangle\right\rangle$ is a second-order Eulerian number [Graham et al. 1994]. In contrast,

$$W_k(z) = \sum_{n \geq 0} \frac{p_n(W_k(a))}{(1 + W_k(a))^{2n-1}} \frac{(z - a)^n}{n!} \quad (5)$$

and the polynomials p_n are known only by recurrence relation [Corless et al. 1996].

This does not mean, however, that the ‘wrong’ function was implemented first in computer algebra systems, because for some other applications, W is the correct tool to use. For example, in the solution of delay differential equations, and in other applications, W is simpler. So it would be convenient if a computer algebra system could support both, much as we use all three of $\sin x$, $\cos x$ and $\tan x$ even though, strictly speaking, we only need (for example) $\tan x$, because $\sin x = 2 \tan(x/2)/(1 + \tan^2(x/2))$ and $\cos x = (1 - \tan^2(x/2))/(1 + \tan^2(x/2))$.

This paper describes the mathematical basis for a Maple package to support ω symbolically (simplification, differentiation, integration, series computation). For convenience, some published formulae and graphs are repeated, from [Corless et al. 1997] and from [Corless and Jeffrey 2002], but most are described here for the first time. The series (3) will be used heavily in [Corless and Jeffrey 2004], where numerical evaluation of this function is discussed.

1.1 Fundamental property of $\omega(z)$

We have that the solution over \mathbb{C} of the logarithmic polynomial

$$y + \ln y = z$$

is, in the absence of a signed zero,

$$y = \begin{cases} \omega(z) & z \neq t \pm i\pi, t \leq -1 \\ \omega(z), \omega(z - 2\pi i) & z = t + i\pi, t \leq -1 \\ \text{no solution} & z = t - i\pi, t \leq -1. \end{cases} \quad (6)$$

For a proof, see [Corless and Jeffrey 2002].

1.2 Graphical visualization

A graph of $\omega(z)$ for real z can be produced in Maple using, for example, the Maple command ‘`plot([y+ln(y), y, y=0.001..2])`’; but ω is simply monotonically increasing, and the plot is not shown here.

We can visualize ω for complex z in several ways. In figures 1 and 2 we plot the mapping $z \rightarrow \omega(z)$ by showing lines in the z -plane and how they map to the ω -plane [Corless and Jeffrey 2002]. The lines in the z -plane are chosen to be the lines on which ω is discontinuous (also called the doubling line and its reflection). It can be easily seen, for example, that the points a and d , which lie adjacent to each other on opposite sides of the upper line, are mapped to non-adjacent places in the ω -plane.

We can supplement these plots. In Figure 3, we see images of a grid of horizontal and vertical lines in the z -plane. Although we could embed Figure 2 in Figure 3, we have presented both of them for the sake of clarity.

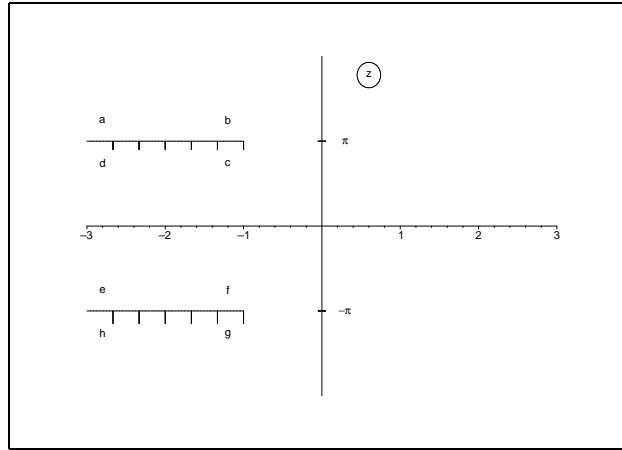


Fig. 1. The z -plane, showing the lines across which ω is discontinuous. The upper line is called the *doubling line*. Along both lines, the closure (indicated by short lines extending down) is taken from below—clockwise around the branch points—to agree with the closure of the unwinding number.

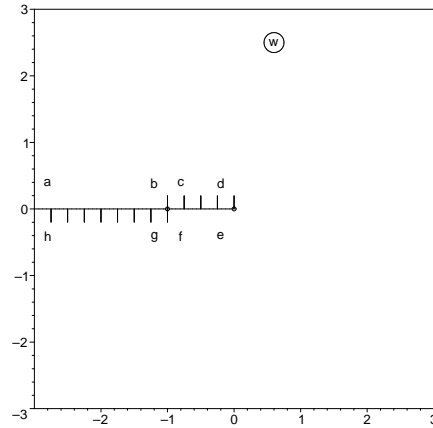


Fig. 2. The ω -plane, showing the images of the doubling line and its reflection. The negative real ω -axis is a branch cut of $\omega + \ln \omega$, which is why that expression is not exactly the inverse function for ω (see equation 14).

1.3 Riemann surface

The Riemann surface for $\omega(z)$ can be plotted by the commands below (see [Corless and Jeffrey 1998] for a general discussion of Riemann surfaces). Note that since ω is single valued, there is only one sheet to the surface.

```
> z := x+I*y; omega := mu + I*nu;
      z := x + y I
      omega := mu + nu I
```

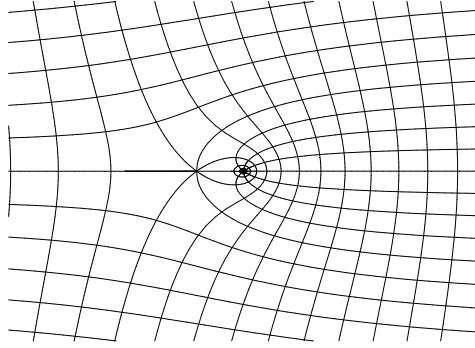


Fig. 3. This plot shows the images of vertical and horizontal lines under the map $z \rightarrow \omega(z)$. The singular point evident in the plot, where the contours intersect is at $\omega = -1$. The closed contours have $\omega = 0$ as their centre.

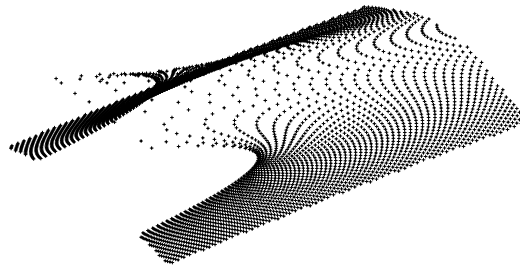


Fig. 4. The Riemann surface for the Wright ω function. The surface is closed on the lower edge of the front discontinuity, and on the upper edge of the back discontinuity.

```
> x := evalc(Re(omega+ln(omega)));
```

$$x := \ln(\sqrt{\mu^2 + \nu^2}) + \mu \quad (7)$$

```
> y := evalc(Im(omega+ln(omega)));
```

$$y := \arctan(\nu, \mu) + \nu \quad (8)$$

```
> plot3d( [x,y,mu], mu=-4..2, nu=-4..4,
> colour=black, axes=NONE, style=PATCHNOGRID,
> labels=["x","y","mu"],
> view=[-2..1, -5..5, -5..3],
> grid=[100,100], orientation=[-114,60],
> style=POINT );
```

The surface is shown in Figure 4, but before we can accept it as representative, we must check that, given the three coordinates x , y , and μ , we can uniquely identify ν , the fourth coordinate. If we can, then there is a bijection between the points on the 3-d surface and the mapping z to ω [Corless and Jeffrey 1998].

Theorem. Given x, y and μ , we can solve (7) – (8) uniquely for ν .

Proof. We start with the case $\nu \neq 0$.

> solve(x=X, nu);

$$\sqrt{-\mu^2 + (e^{X-\mu})^2}, -\sqrt{-\mu^2 + (e^{X-\mu})^2}$$

There are apparently two values of ν for each value of x and μ . However, for $\nu \neq 0$, equation (8) uniquely determines the sign of ν , because y cannot be $\nu + \arctan(\nu, \mu)$ and $-\nu + \arctan(-\nu, \mu)$ simultaneously. (It is simple to show that $y \neq 0$.)

For the case $\nu = 0$, we must check that (x, y, μ) uniquely determine that ν is zero. Putting $\nu = 0$ in (8), we find that y must be zero, if $\mu > 0$, or $y = \pi$, if $\mu < 0$. For $\nu = y = 0$ and $\mu > 0$, the x -equation becomes $X = \mu + \ln \mu$, which has a unique solution by monotonicity.

Finally, we consider the images of the branch cuts. Because of closure, we have

$$\omega(z) = \begin{cases} W_0(-e^x), & \text{where } z = x + \pi i, \\ W_{-1}(-e^x), & \text{where } z = x - \pi i, \end{cases}$$

and in both cases we know that $\nu = 0$.

Remark. We have constructed the Riemann surface using the general technique of working with an inverse function [Corless and Jeffrey 1998], in this case the function was $\omega + \ln(\omega)$. Because $w + \ln w$ has a branch cut for $w < 0$, this function is not exactly the inverse function for ω . We give the exact inverse in equation (14) below. The difference, however, amounts to assigning closure of the surface on the lower edge of the discontinuity on one side, and to the upper edge on the other side. These closures would not be visible in the figure.

2. APPLICATIONS

We give three representative applications.

2.1 Convex conjugate

In [Borwein and Lewis 2000], the “convex conjugate”, namely

$$f^*(x) = \sup_r (rs - f(r))$$

of the function $f(r) = r \ln(r/(1-r)) - r$ is calculated to be $\omega(x)$. Computing the convex conjugate allows, in certain circumstances, global optimization. Since functions like $f(r)$ occur frequently when “information” and entropy are considered, the Wright ω function is quite useful in this area, and not just for this example.

2.2 Fringing fields for a capacitor

In [Valluri et al. 2000] we find the first explicit use of the Wright ω function, though using a different notation. To find the fringing field of a semi-infinite capacitor by conformal mapping, the function

$$\zeta = 1 + z + e^z \tag{9}$$

must be inverted. If we put $y = \exp z$ and hence $\ln y = \ln \exp z = z - 2\pi i \mathcal{K}(z)$, then the results of [Corless and Jeffrey 2002] show that

$$\zeta = 1 + \ln y + 2\pi i \mathcal{K}(z) + y. \tag{10}$$

Table I. Real and complex parts of ω . The macros `unsplit`, `split`, and `stdcx` are internal data structures representing complex numbers that need not be further simplified.

```
macro ( unsplit = 'evalc/unsplit',
       split   = 'evalc/split',
       stdcx   = 'evalc/stdcx');

'evalc/Wrightomega' := proc()
  local a, x, y;
  option 'Copyright (c) 2004 by Robert M. Corless. All rights reserved.';
  a := 'evalc/evalc'(args[nargs]);
  if type(a,'unsplit(algebraic)') then
    unsplit('Wrightomega'(op(a)));
  else
    x := op(1,a);
    y := op(2,a);
    if type(x+I*y,'complex(float)') then
      'evalc/evalc'(Wrightomega(x+I*y));
    elif member(y,{Pi,-Pi}) and is(x<=-1) then
      split( 'LambertW'( 'if'(y=Pi,0,-1),-exp(x)), 0 )
    elif y <> 0 then
      unsplit('Wrightomega'(stdcx(a)));
    else
      # omega(x) real for real x
      split('Wrightomega'(x),0);
    end if;
  end if;
end:
end:
```

Further, we take the region in the z -plane to be between infinite parallel plates: $-\pi \leq \text{Im}(z) \leq \pi$, and hence $\mathcal{K}(z) = 0$ except for the bottom plate when $\mathcal{K}(z) = -1$. Away from the plates, then, we have $y = \omega(\zeta - 1)$. The unique solution on the plates follows on consideration of the branches, as before. We have, using the transformation (10),

$$\begin{aligned} z &= \ln y = \zeta - 1 - y \\ &= \zeta - 1 - \omega(\zeta - 1). \end{aligned} \tag{11}$$

2.3 Borda's mouthpiece

In [Kahan and Darcy 1998] we find the following problem, similar to the capacitor problem above, used as an example of why a signed floating-point zero is useful. Define

$$g(z) = z^2 + z\sqrt{z^2 + 1} \tag{12}$$

and

$$F(z) = 1 + g(z) + \ln g(z). \tag{13}$$

If we can solve a two-dimensional fluids or electrostatics problem in the z -plane, say with a sink at the origin and walls along some portion of the imaginary axis, then by the principle of conformal mapping we have also solved fluids or electrostatics problems in the g -plane and the F -plane, giving different geometries. The geometry

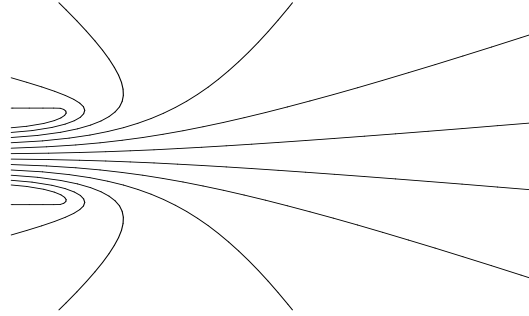


Fig. 5. Borda's mouthpiece: images of rays $z = r \exp(i\theta)$ under $F(z) = 1 + g(z) + \ln g(z)$ where $g(z) = z^2 + z\sqrt{z^2 + 1}$. Support for signed floating-point zeros facilitates drawing this picture. This figure was drawn in Maple, using a natural syntax.

of the F -plane corresponds to fluids entering a semi-infinite channel with walls at $\pm i\pi$. See Figure 5.

We do not need to evaluate ω to draw this picture. However, to identify the potential at any point in the F -plane, we must solve for the corresponding value of z . It is clear that $g(F) = \omega(F - 1)$, and simple to show that

$$z = \pm g(F) / \sqrt{2g(F) + 1} .$$

3. ALGEBRAIC PROPERTIES

We first consider the simplification of ω for special values of its argument. After these are established, there remain questions concerning their implementation. Wright ω is the same as all other special functions in posing the following questions to computer algebra developers. The first question is whether to simplify at all. For example, an expansion in a series of Legendre polynomials looks clearer when all functions are left as Legendre polynomials, even though the early Legendre functions can be replaced by low-degree polynomials in a monomial basis, and therefore it may be convenient not to 'simplify'.

A deeper technical reason is that simplification is, in general, undecidable. For example, even recognizing 0 amongst arbitrary expressions built up from constants such as one symbol x representing an unknown but real quantity, $\ln 2$, π , e , and i with the elementary operations of addition, subtraction, multiplication, division, exponentiation and composition, is recursively unsolvable [Richardson 1968]. Even without a symbolic parameter in the input, recognizing zero is very difficult, and solveable for algebraic numbers only if Schanuel's conjecture is true [Richardson 1992]. The notion that reliably detecting 0 is difficult is familiar to researchers in scientific computing, who are very familiar with the question of error tolerances; but the fact that it is technically undecidable in exact computation is somewhat surprising. Nonetheless, heuristics for pragmatic simplification are in great demand, in part to keep down unnecessary expression swell and thus keeping computational costs as much under control as possible.

A second question is how much effort should be put into searching for simplifications. For example, although it is true that

$$\omega\left(\ln\left[\frac{528309}{168976}\ln\left(\frac{9}{4}\right)3^{\frac{7451}{10561}}2^{\frac{3110}{10561}}\right]\right)=\frac{19567}{10561}\ln\frac{9}{4},$$

a computer algebra system must decide how much effort to devote to finding this. [Our current code does not find this simplification.]

The third question is the perennial one for computer algebra systems: ‘what is a simplification?’ (see e.g. [Carette 2004; Moses 1971]). Here we do not discuss these questions, but concentrate on what transformations are possible in this particular case. We begin with the basic code for `Wrightomega` (see Table II). Using this code, we see, for instance, the following automatic transformation:

```
> omega( 3 + ln(3) );
```

3

No transformations are done if no simplification is possible.

```
> omega( 3 );
```

$\omega(3)$

Floating-point input automatically triggers floating-point evaluation of the function (see [Corless and Jeffrey 2004] for the details of the numerical method invoked). For example, at 20 digits, we have

```
> omega( 3.0 );
```

2.2079400315693229986

We may verify that $\omega(3.0) + \ln\omega(3.0) = 3.0$. [The “%” symbol below refers to the previous result.]

```
> % + ln(%);
```

3.00000000000000000000

If the input may simplify, but the simplification is not obvious, then the basic code in Table II does not make any transformation (note that $\sin(-3)$ becomes $-\sin 3$, though, from a built-in Maple automatic simplification).

```
> omega( sin(-3) + ln(sin(-3)) );
```

$\omega(-\sin(3) + \ln(-\sin(3)))$

3.1 Inverse transformation

It is shown in [Corless and Jeffrey 2002] that the inverse function ω^{-1} obeys

$$\omega^{-1}(y) = \begin{cases} y + \ln(y) - 2\pi i & -\infty < y < -1 \\ -1 \pm i\pi & y = -1 \\ y + \ln(y) & \text{otherwise.} \end{cases} \quad (14)$$

From this, we deduce the rules

$$\omega(z + \ln z) = z \quad \text{for all } z \in \mathbb{C} \quad (15)$$

$$\omega(x + \ln x - 2\pi i) = x \quad \text{for all } x \leq -1 \quad (16)$$

$$\omega(x + \ln |x| - \pi i) = x \quad \text{for all } x \leq -1 \quad (17)$$

Note that (16) is not a special case of (15) because $x + \ln x - 2\pi i$ is not in the range of $z + \ln z$. Table III gives some representative special values, including examples

Table II. Fundamental code for ω

```

Wrightomega := proc( z )
  local logs, r;
  if z=1 then
    1
  elif z=-1+I*Pi or z=-1-I*Pi then
    -1
  elif z=-infinity then
    0
  elif z=infinity then
    infinity
  elif type( z, complex(float) ) then
    'evalf/Wrightomega'( z )
  elif type( z, '+' ) and has( z, ln ) then
    # Check for simple rational values
    logs := select(has,[op(z)],ln);
    if nops(logs)=1 and type( logs[1], specfunc('rational',ln) ) then
      r := op(1,logs[1]);
      if z=r+ln(r) then
        r
      elif z=-r+ln(r)-I*Pi and r >= 1 then
        -r
      else
        'procname'(args)
      end if;
    elif nops(logs)=1 and type( -logs[1], specfunc('rational',ln) ) then
      r := op(1,-logs[1]);
      if z= 1/r - ln(r) then
        1/r
      elif z=-1/r - ln(r) + I*Pi and r >= 1 then
        -1/r
      else
        'procname'(args)
      end if;
    else
      'procname'(args)
    end if;
  else
    'procname'(args)
  end if;
end proc;

```

of these simplifications.

To implement these simplifications in Maple, we must detect the pattern $z + \ln z$ in the argument A when ω is called. We must also decide if the inequality $x \leq -1$ is true, which, as alluded to previously, is undecidable if x is formed from sufficiently complicated exact inputs. This undecidability is confined to the Maple `assume` facility [Weibel and Gonnet 1993] and the command `is`, which uses powerful heuristics and a specialized form of interval arithmetic implemented in the `evalr` command, but, of course, cannot be algorithmic in the presence of expressions with symbolic parameters.

Table III. Some special values of $\omega(z)$.

z	$\omega(z)$
$-\infty$	0
0	$\omega(0) = 0.5671\dots$
1	1
$2 + \ln 2$	2
$-1/3 + \ln(1/3) + i\pi$	$-1/3$
$-1 + i\pi$	-1
$-1 - i\pi$	-1
$-2 + \ln 2 - i\pi$	-2
∞	∞

As described above, Maple has two levels of simplification, ‘automatic’ and ‘on demand’. The present simplifications are placed in ‘demand’ simplifications. The Maple `match` command would not succeed in detecting the $z + \ln z$ pattern except in simple cases, and therefore is not used. A specific example of where we need more is $z = \ln 3$, when the pattern becomes $\ln 3 + \ln \ln 3$ and a simple match can assign either term to be the logarithm term in the pattern. The present code, therefore, searches for the logarithm having the longest argument and then uses that argument to test for the pattern. This is a heuristic, of course, but it has proved useful. The search is implemented using a special-purpose matching function `lnplus?` which returns true if A is a sum of the form $b + \ln c$. Each term in A is examined to see whether it is a logarithm (using `select`) and its argument added to a list; then the list is sorted using the Maple `length` command and the longest element is assigned to the name c , and then $b = A - \ln c$. This allows the following simplifications. Using the code in Appendix 4, we see the following examples of simplification on demand.

```

> omega( sin(3) + ln(sin(3)) );
      omega(sin(3) + ln(sin(3)))
> simplify(%);
      sin(3)
> omega( ln(3) + ln(ln(3)) );
      omega(ln(3) + ln(ln(3)))
> simplify( % );
      ln(3)
> omega( x + ln(x) );
      omega(x + ln(x))
> simplify( % );
      x
> omega( x + ln(x) - I*2*Pi );
      omega(x + ln(x) - 2 I pi)
> simplify(%) assuming x <= -1;
      x

```

```

> omega( x + ln(abs(x)) - Pi*I );
      omega(x + ln(|x|) - pi I)
> simplify(%) assuming x <= -1;
      x
> omega( 1-sqrt(5) + ln( sqrt(5)-1 ) - Pi*I );
      omega(1 - sqrt(5) + ln(sqrt(5) - 1) - pi I)
> simplify(%);
      1 - sqrt(5)

```

3.2 Transformations for log-exp

Because $\ln e^z \neq z$, separate theorems are needed for arguments containing exponentials.

Theorem. Let $m, n \in \mathbb{Z}$. For $n > 0$, let $-\frac{3}{4} < r \leq \frac{1}{4}$, or for $n < 0$ let $-\frac{1}{4} < r \leq \frac{3}{4}$ or for $n = 0$ let $0 < |r| \leq \frac{1}{4}$, then

$$\omega\left(\ln_n\left(2\pi i(n+r)e^{2\pi i(m+r)}\right)\right) = 2\pi i(n+r).$$

Proof. Note $e^{2\pi i(m+r)} = e^{2\pi ir}$. Take first the case $n > 0$. Then $0 < n - \frac{3}{4} < n+r \leq n + \frac{1}{4}$ and $\ln_n(2\pi i(n+r)e^{2\pi ir}) = \ln_n(2\pi(n+r)e^{2\pi i(r+1/4)})$. Now $-\frac{1}{2} < r + \frac{1}{4} \leq \frac{1}{2}$ and hence $-\pi < 2\pi(r + \frac{1}{4}) \leq \pi$ and so $\ln_n[2\pi(n+r)e^{2\pi i(r+1/4)}] = \ln(2\pi(n+r)) + 2\pi i(r + \frac{1}{4} + n) = \ln(2\pi i(n+r)) + 2\pi i(n+r)$. Finally, we see that $\omega(\ln(2\pi i(n+r)) + 2\pi i(n+r)) = 2\pi i(n+r)$ because $2\pi i(n+r) \neq t \pm i\pi$ for any $t \leq -1$. The proof for $n < 0$ is similar except we use $-\frac{1}{2} < r - \frac{1}{4} \leq \frac{1}{2}$ and $\ln(-2\pi(n+r))$ because $n+r < 0$. The case $n = 0$ has two subcases $-\frac{1}{2} < r < 0$ and $0 < r \leq \frac{1}{4}$. First note

$$\ln\left(2\pi ire^{2\pi i(m+r)}\right) = \ln\left(\pm 2\pi re^{2\pi i(r \pm 1/4)}\right).$$

Now $0 < r \leq \frac{1}{4} \Rightarrow 0 < \frac{1}{4} < r + \frac{1}{4} \leq \frac{1}{2}$ and likewise $-\frac{1}{4} < r < 0 \Rightarrow -\frac{1}{2} < r - \frac{1}{4} < -\frac{1}{4}$. In the first case $0 < \pi/2 < 2\pi(r + \frac{1}{4}) \leq \pi$ and in the second $-\pi < 2\pi(r - \frac{1}{4}) < -\pi/2 < 0$, and so in either case

$$\begin{aligned} \ln\left(2\pi ire^{2\pi i(r+1/4)}\right) &= \ln(2\pi r) + 2\pi i(r + \frac{1}{4}) \\ &= \ln(2\pi ir) + 2\pi ir, \quad r > 0 \end{aligned}$$

$$\begin{aligned} \ln\left(-2\pi re^{2\pi i(r-1/4)}\right) &= \ln(-2\pi r) + 2\pi i(r - \frac{1}{4}) \\ &= \ln(2\pi ir) + 2\pi ir, \quad r < 0 \end{aligned}$$

and $\omega(\ln(2\pi ire^{2\pi ir})) = 2\pi ir$ in either case, because $2\pi ir \neq t \pm i\pi$ for any $t \leq -1$. □

Remark. The case $n = 0, r = 0$ is special, in that

$$\omega(\varepsilon e^{i\theta} + \ln \varepsilon e^{i\theta}) = \varepsilon e^{i\theta}$$

for $-\pi < \theta \leq \pi$ and $\varepsilon > 0$. Hence in a limiting sense $\omega(\ln(2\pi ire^{2\pi ir})) = 2\pi ir$ even if $r = 0$.

Corollary: Under the same conditions, including $n = r = 0$,

$$W_n \left(2\pi i(n+r)e^{2\pi i(m+r)} \right) = 2\pi i(n+r).$$

If the branch index does not match the n in the $2\pi i(n+r)$, this simplification does not apply (and is not true because $2\pi i(n+r)$ is not in range W_k unless $k = n$). The proof of the corollary is just a consequence of $W_n(z) = \omega(\ln_n z)$.

Remark. $\exp(2\pi ir)$ will automatically simplify in Maple for some short rationals r (for example, $r = 1/4$, whence $\exp(2\pi ir) = i$). This makes recognition of the pattern more difficult. Our code explicitly checks for such rationals, once a partial match has been made and the branch n identified.

Remark. Numerical verification of the edge conditions is difficult. Consider for example $n = 0$ and $r = -\frac{1}{4}$. Then computation of $2\pi ir e^{2\pi ir} = -\frac{2\pi i}{4} e^{-2\pi i/4} = -\frac{\pi i}{2} \cdot e^{-\pi i/2} = \frac{\pi}{2}(-i) \cdot (-i) = -\pi/2$ in exact arithmetic; but $2\pi i \cdot (-0.25) \cdot e^{2\pi i \cdot (-0.25)}$ invokes numerical evaluation of $\exp(-\pi i \cdot (0.5))$ in most language models; at, say, 20 digits this results in $2\pi ir e^{2\pi ir} \doteq -1.570 - 4.199 \cdot 10^{-20}i$, which is very close to $-\pi/2$ but has a small negative imaginary part; subsequent correct calculation gives

$$W_0(-\pi/2) = +i\pi/2$$

in agreement with the theorem, but in the numerical case

$$W_0(-\pi/2 - i\varepsilon) = -i\pi/2 + O(\varepsilon)$$

which is grossly different from $+i\pi/2$.

The danger we see here is inherent in the problem of evaluating discontinuous functions, and motivates exact computation and simplification whenever possible. Notice that logarithm has this problem on the negative real axis, and that the Wright ω function has this problem on the rays $t \pm i\pi$, $t \leq -1$.

Using the code in Appendix 4, we see the following example of simplification on demand for examples of the form of the theorem in this section, one for which the exponential automatically simplifies on input, making pattern matching difficult.

```
> m := 5: n := 3: r := 21/87:
> zz := ln( 2*Pi*I*(n+r)*exp(2*Pi*I*(m+r))) + 2*Pi*I*n ;
```

$$zz := \ln\left(\frac{188}{29} I \pi e^{(\frac{14}{29} I \pi)}\right) + 6 I \pi$$

```
> omega( zz ) ;
```

$$\omega\left(\ln\left(\frac{188}{29} I \pi e^{(\frac{14}{29} I \pi)}\right) + 6 I \pi\right)$$

```
> simplify( omega( zz ), omega );
```

$$\frac{188}{29} I \pi$$

3.3 Further simplifications

There are a number of simplifications of the type $\omega(\ln(a \ln b)) = c \ln b$, for $a, b, c \in \mathbb{N}$. Such simplifications have a number of uses, for example in equation solving. Let $b, c \in \mathbb{R}^+$, then the equation

$$xb^x = c$$

has a solution $x = \omega(\ln[c \ln b]) / \ln b$.

A simple and interesting strategy for detecting such symbolic simplifications uses floating-point computation. We compute $\omega(\ln[a \ln b]) / \ln b$ as a floating point number and then round it to an integer. This gives a candidate for a symbolic simplification, which can be verified directly using the definition of ω .

Example: To simplify $\omega(\ln(896 \ln 2)) = \omega(7 \ln 2 + \ln 7 + \ln \ln 2)$, we compute $\omega(\ln(896 \ln 2)) / \ln 2 = 7.00000$. It is then a simple matter to verify that $\omega = 7 \ln 2$ satisfies $\omega + \ln \omega = \ln[896 \ln 2]$. Our current code does this through the simplify command:

```
> omega( 7 ln(2) + ln(7) + ln(ln(3)) );
      omega(7 ln(2) + ln(7) + ln(ln(3)))
> simplify(%);
      7 ln 2
```

It is obvious that there are many generalizations of the domain of the parameters, and of the domain of the solution(s). A decision on which transformations should be implemented in a given computer algebra system is typically made on an *ad hoc* basis, and a methodology for such decisions has not been the subject of a general discussion in the literature.

3.4 Derivatives and integrals

The implicitly elementary Lambert W function and the Wright ω function are not Liouvillian; that is, neither are expressible as finite combinations of exponentials, logarithms, root extractions, or indefinite integrals of simpler functions. For a proof, see [Bronstein et al.]. For a discussion of algorithms, including the Risch algorithm, for the analytic integration of elementary functions, see [Bronstein 1997].

Notwithstanding the fact that ω is not Liouvillian, many integrals involving ω at the lowest level may be integrated by the simple substitution $x = \omega + \ln \omega$, because the derivative of ω is rational in ω . A more general set of heuristics is given in [Bronstein 2004], and also in the upcoming 2nd edition of [Bronstein 1997], in press. A genuine extension of the Risch algorithm to include the Lambert W function and the Wright ω function is under way (Manuel Bronstein, private communication).

We now quote results from [Corless and Jeffrey 2002]:

$$\frac{d\omega}{dz} = \frac{\omega}{1 + \omega}$$

$$\int \omega^n dz = \begin{cases} \frac{\omega^{n+1}-1}{n+1} + \omega^n/n & \text{if } n \neq -1 \\ \ln \omega - 1/\omega & \text{if } n = -1 \end{cases}$$

The derivative formula is valid except on the doubling line and its reflection, when it is valid as a derivative in the real direction only. The integrals can be verified directly by differentiation of both sides. The addition of the constant term $-1/(n+1)$ to the integral of ω^n is a trick due, in the case $\int x^n dx$, to W. Kahan. Using this trick, the formula for limiting case $n = -1$ is a simple limit of the formula for $n \neq -1$.

Table IV. Maple code to integrate functions containing ω

```

'int/Wrightomega' := proc (f) local gx, h, inds, u;
  inds := map(proc (x) if op(0,x) = ('Wrightomega') then x end if end proc,
    indets(f,function));
  if nops(inds) <> 1 then RETURN(FAIL) end if;
  inds := inds[1];
  if nops(inds) = 1 then
    gx := op(inds)
  else
    RETURN(FAIL)
  end if;
  if not type(gx,linear(_X)) then RETURN(FAIL) end if;
  h := subs(inds = u, _X = (u+ln(u)-coeff(gx,_X,0))/coeff(gx,_X),f);
  h := h*(1+1/u)/coeff(gx,_X);
  h := int(h,u);
  if has(h,int) then RETURN(FAIL) end if;
  subs(ln(u) = gx-u,u = inds,h)
end proc:

```

Differentiation is easily implemented in Maple:

```

'diff/Wrightomega' := proc( expr, var );
  diff(expr,var)*(Wrightomega(expr)/(1+Wrightomega(expr)))
end proc:

```

Notice that we must encode the chain rule, but once this procedure is loaded into Maple, arbitrary orders of derivatives (and hence series at arbitrary regular points) become available.

```
> diff( omega(x), x );
```

$$\frac{\omega(x)}{1 + \omega(x)}$$

Now the third derivative:

```
> diff( omega(x), x$3 );
```

$$\frac{\omega(x)}{(1 + \omega(x))^3} - \frac{4\omega(x)^2}{(1 + \omega(x))^4} + \frac{3\omega(x)^3}{(1 + \omega(x))^5}$$

Notice that (by default) the derivative is returned in an un-normalized form. On demand, we may put it into normal form, as follows:

```
> normal(%);
```

$$-\frac{\omega(x)(-1 + 2\omega(x))}{(1 + \omega(x))^5}$$

```
> series( omega(x), x=1 );
```

$$1 + \frac{1}{2}(x-1) + \frac{1}{16}(x-1)^2 - \frac{1}{192}(x-1)^3 - \frac{1}{3072}(x-1)^4 + \frac{13}{61440}(x-1)^5 + O((x-1)^6)$$

Integration is more complicated. The code in Table IV implements the substitution $x = w + \ln w$, which transforms many integrals containing ω into integrals of elementary functions. When this strategy of making the substitution $z = w + \ln w$ is programmed in Maple as in Table IV, the following integrals are examples of

what is possible and what is not.

```
> Int( log( omega(z) ), z )=
> int( log( omega(z) ), z ) + C;
```

$$\int \ln(\omega(z)) dz = \omega(z) \ln(\omega(z)) - \omega(z) + \frac{1}{2} \ln(\omega(z))^2 + C$$

```
> Int((omega(z)+1)/(2*omega(z)^2+omega(z)+1), z)=
> int((omega(z)+1)/(2*omega(z)^2+omega(z)+1), z)+C;
```

$$\int \frac{\omega(z) + 1}{2\omega(z)^2 + \omega(z) + 1} dz = -\frac{1}{4} \ln(2\omega(z)^2 + \omega(z) + 1) + \frac{5}{14} \sqrt{7} \arctan\left(\frac{1}{7} (4\omega(z) + 1) \sqrt{7}\right) + \ln(\omega(z)) + C$$

```
> Int( sin(omega(z)), z ) =
> int( sin(omega(z)), z ) + C;
```

$$\int \sin(\omega(z)) dz = -\cos(\omega(z)) + \text{Si}(\omega(z)) + C$$

In what follows, we turn on some diagnostics for integration, that allow us to track the steps used in the attempt to integrate.

```
> infolevel[int]:=5:
> Int( 1/(2+sin(omega(z))), z )=
> int( 1/(2+sin(omega(z))), z );
```

```
int/indef1: first-stage indefinite integration
int/indef2: second-stage indefinite integration
int/indef2: applying derivative-divides
int/indef1: first-stage indefinite integration
int/indef2: second-stage indefinite integration
int/trigon: case of integrand containing trigs
int/prptrig: case ratpoly*trig(arg)
int/rischnorm: enter Risch-Norman integrator
int/rischnorm: exit Risch-Norman integrator
int/risch: enter Risch integration
```

$$\int \frac{1}{2 + \sin(\omega(z))} dz = \int \frac{1}{2 + \sin(\omega(z))} dz$$

The above computation shows that the above integrand cannot be transformed just by the substitution $z = w + \ln w$ to an integrand that has an elementary antiderivative. The use of the ordinary Risch algorithm in that context does not show that the integrand cannot be expressed in terms of an extension containing ω ; for that, we need the extension promised by Manuel Bronstein.

3.5 Laplace transform of $\omega(t)$

We have, by hand-guided computation with Maple,

$$\mathcal{L}(\omega)(s) := \int_0^\infty e^{-st} \omega(t) dt = s^{s-2} \Gamma(1-s, s\omega(0)) + \frac{\omega(0)}{s} \quad (18)$$

if $\operatorname{Re}(s) > 0$, where the incomplete gamma function takes s in both its arguments.

3.6 Series about regular points

Once the derivative has been coded, the Maple built-in routine `series/function` can compute series about regular points. For example, about $x = 0$, we have

```
> series( omega(x), x );
```

$$\begin{aligned} & \omega(0) + \frac{\omega(0)}{1 + \omega(0)} x + \frac{1}{2} \frac{\omega(0)}{(1 + \omega(0))^3} x^2 - \frac{1}{6} \frac{\omega(0)(-1 + 2\omega(0))}{(1 + \omega(0))^5} x^3 + \\ & \frac{1}{24} \frac{\omega(0)(1 - 8\omega(0) + 6\omega(0)^2)}{(1 + \omega(0))^7} x^4 - \\ & \frac{1}{120} \frac{\omega(0)(-1 + 22\omega(0) - 58\omega(0)^2 + 24\omega(0)^3)}{(1 + \omega(0))^9} x^5 + O(x^6) \end{aligned}$$

This, of course, is the first few terms of the series (3), for $a = 0$. The series about $x = 1$ we have already seen; here are the first few terms of the series about $x = 2 + \ln 2$, where $\omega(a) = 2$:

```
> series( omega(x), x=2+ln(2) );
```

$$\begin{aligned} & 2 + \frac{2}{3}(x - 2 - \ln(2)) + \frac{1}{27}(x - 2 - \ln(2))^2 - \frac{1}{243}(x - 2 - \ln(2))^3 + \frac{1}{2916} \\ & (x - 2 - \ln(2))^4 - \frac{1}{393660}(x - 2 - \ln(2))^5 + O((x - 2 - \ln(2))^6) \end{aligned}$$

3.7 Series about ∞

The series is (translating results of [Comtet 1974; de Bruijn 1961])

$$\omega(z) = z - \ln z + \sum_{\ell \geq 0} \sum_{m \geq 1} c_{\ell m} \frac{\ln^m z}{z^{\ell+m}}, \quad (19)$$

where $c_{\ell m} = (-1)^\ell \left[\begin{smallmatrix} \ell+m \\ \ell+1 \end{smallmatrix} \right] / m!$ is defined in terms of Stirling cycle numbers [Graham et al. 1994]. This series converges for large enough z , outside the region bounded by the doubling line and its reflection. The proof is unpublished.

```
> asympt( omega(x), x );
```

$$\begin{aligned} & x - \ln(x) + \frac{\ln(x)}{x} + \frac{-\ln(x) + \frac{1}{2} \ln(x)^2}{x^2} + \frac{\ln(x) - \frac{3}{2} \ln(x)^2 + \frac{1}{3} \ln(x)^3}{x^3} \\ & + \frac{-\ln(x) + 3 \ln(x)^2 - \frac{11}{6} \ln(x)^3 + \frac{1}{4} \ln(x)^4}{x^4} \\ & + \frac{\ln(x) - 5 \ln(x)^2 + \frac{35}{6} \ln(x)^3 - \frac{25}{12} \ln(x)^4 + \frac{1}{5} \ln(x)^5}{x^5} \end{aligned}$$

> asympt(omega(x)^2, x);

$$\begin{aligned}
 & x^2 - 2 \ln(x) x + 2 \ln(x) + \ln(x)^2 + \frac{-\ln(x)^2 - 2 \ln(x)}{x} \\
 & + \frac{2 \ln(x) - 2 \ln(x)^2 + \frac{2}{3} \ln(x)^3 - 2 \ln(x) \%1}{x^2} + \\
 & \frac{-2 \ln(x) + 6 \ln(x)^2 - \frac{11}{3} \ln(x)^3 + \frac{1}{2} \ln(x)^4 - 2 \%2 \ln(x) + 2 \ln(x) \%1}{x^3} \\
 & + (-2 \ln(x) (-\ln(x) + 3 \ln(x)^2 - \frac{11}{6} \ln(x)^3 + \frac{1}{4} \ln(x)^4) + 2 \%2 \ln(x) \\
 & + 2 \ln(x) - 10 \ln(x)^2 + \frac{35}{3} \ln(x)^3 - \frac{25}{6} \ln(x)^4 + \frac{2}{5} \ln(x)^5 + \%1^2) / x^4 \\
 & + (-2 \ln(x) (\ln(x) - 5 \ln(x)^2 + \frac{35}{6} \ln(x)^3 - \frac{25}{12} \ln(x)^4 + \frac{1}{5} \ln(x)^5) \\
 & + 2 \%2 \%1 + 2 \ln(x) (-\ln(x) + 3 \ln(x)^2 - \frac{11}{6} \ln(x)^3 + \frac{1}{4} \ln(x)^4)) / x^5 \\
 & + O(\frac{1}{x^6}) \\
 & \%1 := -\ln(x) + \frac{1}{2} \ln(x)^2 \\
 & \%2 := \ln(x) - \frac{3}{2} \ln(x)^2 + \frac{1}{3} \ln(x)^3
 \end{aligned}$$

3.8 Series about $-\infty$

From the series $W(z) = \sum_{n \geq 1} (-n)^{n-1} z^n / n!$, which is valid for $|z| < \exp(-1)$, we have

$$\omega(z) = \sum_{n \geq 1} \frac{(-n)^{n-1}}{n!} e^{nz} . \tag{20}$$

The region of convergence of this series is $|\exp(z)| < \exp(-1)$, or, if $z = x + iy$, the half-plane $x < -1$. If, further, $-\pi < y \leq \pi$, i.e. z lies between the doubling line and its reflection, then the series converges to $\omega(z)$. The proof is unpublished.

> asympt(omega(-x), x);

$$e^{(-x)} - (e^{(-x)})^2 + \frac{3}{2} (e^{(-x)})^3 - \frac{8}{3} (e^{(-x)})^4 + \frac{125}{24} (e^{(-x)})^5 + O((e^{(-x)})^6)$$

> asympt(omega(-x^2), x);

$$e^{(-x^2)} - (e^{(-x^2)})^2 + \frac{3}{2} (e^{(-x^2)})^3 - \frac{8}{3} (e^{(-x^2)})^4 + \frac{125}{24} (e^{(-x^2)})^5 + O((e^{(-x^2)})^6)$$

3.9 Branch point series for $\omega(z)$

The Wright ω function has branch points at $z = -1 \pm i\pi$. The branch point series are awkward to write down because of the closure properties of ω . Once it is granted that Lambert W should be counter-clockwise continuous, and that ω

has the definition (1), then at the branch points, ω must be clockwise continuous. However, at the branch point, the behaviour is that of square root, which is counter-clockwise continuous. Therefore, in order to write down branch-point series for Wright ω , we must first create a square root with clockwise closure. As a result, near $z = -1 + i\pi$, we have

$$\omega(z) = - \sum_{n \geq 0} a_n \left(i \sqrt{\overline{\overline{2(z+1-i\pi)}}} \right)^n \quad (21)$$

where the double conjugation gives us the correct closure from below on $t + i\pi$ for $t \leq -1$. Near $z = -1 - i\pi$,

$$\omega(z) = - \sum_{n \geq 0} a_n \left(-i \sqrt{\overline{\overline{2(z+1+i\pi)}}} \right)^n. \quad (22)$$

In both cases a_n is given by the recurrence relation [Marsaglia and Marsaglia 1990]

$$\begin{aligned} a_0 &= a_1 = 1 \\ a_k &= \frac{1}{(k+1)a_1} \left(a_{k-1} - \sum_{i=2}^{k-1} i a_i a_{k+1-i} \right). \end{aligned} \quad (23)$$

The derivation of these series from the results of [Marsaglia and Marsaglia 1990] is straightforward, except for the use of $\sqrt{\overline{\overline{z}}}$. This construction, verified in [Corless and Jeffrey 2002], is one of a family of transformations used by G. Batchelor, and gives us the correct closure.

Remark. The use of $\sqrt{\overline{\overline{(z-a)}}}$ to represent a square root function with a closure different from the CCC closure, as explained by Kahan [Kahan 1986], is a useful tool in a computer algebra setting. However, it relies on the designers to be sophisticated enough to provide symbolic means of representing (and not over-simplifying) these series, and the users to be sophisticated enough to know that $\sqrt{\overline{\overline{z}}} \neq \sqrt{z}$ on the branch cut.

Using the code in Appendix A, we find (for example)

```
> series( omega(x), x=-1-I*Pi );
```

```
Warning, The series has wrong signs if argument to Wrightomega is on a
branch cut Re <=-1, Im = +/-Pi
```

$$\begin{aligned} & \left(-\frac{1}{3} - \sqrt{2} \sqrt{x+1+\pi I} I + \frac{2x}{3} + \frac{2}{3} I \pi + \frac{1}{18} I \sqrt{2} (x+1+\pi I)^{(3/2)} \right. \\ & \left. + \frac{2(x+1+\pi I)^2}{135} - \frac{1}{1080} I \sqrt{2} (x+1+\pi I)^{(5/2)} \right) \end{aligned}$$

That last series is *incorrect* on the branch cut. At present, Maple does not have a mechanism for representing the conjugate-squareroot-conjugate construct that we need for a correct series, in a way that allows interaction with the system (of course we could simply program the basic series with conjugates—the problem is that Maple cannot then use that series for compositions such as $\omega(\ln(-\exp(-1-t^2/2)))$). In fact, the following two computations are exactly wrong when $t > 1$ —they should

be interchanged—also in part because Maple simplifies `series(sqrt(t^2),t)` to t .

```
> a1 := series( omega( -1+I*Pi - t^2 ), t, 12 );
```

Warning, The series has wrong signs if argument to Wrightomega is on a branch cut $\text{Re} \leq -1$, $\text{Im} = \pm\pi$

$$a1 := -1 - \sqrt{2}t - \frac{2}{3}t^2 - \frac{\sqrt{2}}{18}t^3 + \frac{2}{135}t^4 - \frac{\sqrt{2}}{1080}t^5 - \frac{4}{8505}t^6 + \frac{139\sqrt{2}}{680400}t^7 - \frac{2}{25515}t^8 + \frac{571\sqrt{2}}{146966400}t^9 + \frac{1124}{189448875}t^{10} - \frac{163879\sqrt{2}}{67898476800}t^{11}$$

```
> a2 := series( omega( -1-I*Pi - t^2 ), t, 12 );
```

Warning, The series has wrong signs if argument to Wrightomega is on a branch cut $\text{Re} \leq -1$, $\text{Im} = \pm\pi$

$$a2 := -1 + \sqrt{2}t - \frac{2}{3}t^2 + \frac{\sqrt{2}}{18}t^3 + \frac{2}{135}t^4 + \frac{\sqrt{2}}{1080}t^5 - \frac{4}{8505}t^6 - \frac{139\sqrt{2}}{680400}t^7 - \frac{2}{25515}t^8 - \frac{571\sqrt{2}}{146966400}t^9 + \frac{1124}{189448875}t^{10} + \frac{163879\sqrt{2}}{67898476800}t^{11}$$

The series are correct for any value of $t \in \mathbb{C} \setminus [1, \infty)$, including values of t that have signed zeros. However, for $t > 1$, they should be switched.

4. CONCLUDING REMARKS

A package has been written in Maple to support the Wright ω function and its description given here. The features it provides include conversion tools to convert answers containing Lambert W to equivalent answers containing ω , and vice versa. Simplification, differentiation, analytical integration, and series generation are all present. We also implemented a prototype of the unwinding number, which may have further utility in symbolic simplification.

Acknowledgements

Many people were very helpful with discussions during the writing of this paper and its accompanying code, especially Michael Cherkassoff, Dave Hare, Edgardo Cheb-Terrab, and Jürgen Gerhard. Some testing of the efficiency of higher order codes was carried out by Hui Ding. Jon Borwein and Bill Gosper provided motivation to look at ω .

REFERENCES

- BORWEIN, J. M. AND LEWIS, A. S. 2000. *Convex Analysis and Nonlinear Optimization*. Advanced Books in Mathematics. Canadian Mathematical Society.
- BRADFORD, R. J., CORLESS, R. M., DAVENPORT, J. H., JEFFREY, D. J., AND WATT, S. M. 2002. Reasoning about the elementary functions of complex analysis. *Annals of Mathematics and Artificial Intelligence* 36, 303–318.
- BRONSTEIN, M. 1997. *Symbolic integration I: transcendental functions*. Springer-Verlag New York, Inc.
- BRONSTEIN, M. 2004. Structure theorems for parallel integration. *submitted*.
- BRONSTEIN, M., CORLESS, R. M., DAVENPORT, J. H., AND JEFFREY, D. J. Algebraic properties of the Lambert W function. *submitted*.

- CARETTE, J. 2004. Understanding expression simplification. In *ACM International Symposium on Symbolic and Algebraic Computation*. ACM Press, 72–79.
- COMTET, L. 1974. *Advanced Combinatorics*. Reidel.
- CORLESS, R. M. 2002. *Essential Maple 7*, 2nd ed. Springer-Verlag.
- CORLESS, R. M., GONNET, G. H., HARE, D. E. G., JEFFREY, D. J., AND KNUTH, D. E. 1996. On the Lambert W function. *Advances in Computational Mathematics* 5, 329–359.
- CORLESS, R. M. AND JEFFREY, D. J. 1996. The unwinding number. *SIGSAM BULLETIN: Communications in Computer Algebra* 30, 2:116, 28–35.
- CORLESS, R. M. AND JEFFREY, D. J. 1998. Elementary Riemann surfaces. *SIGSAM Bulletin* 32, 1 (March), 11–17.
- CORLESS, R. M. AND JEFFREY, D. J. 2002. *On the Wright ω function*. LNAI, vol. 2385. Springer, Marseille, 76–89.
- CORLESS, R. M. AND JEFFREY, D. J. 2004. Complex numerical values of the Wright ω function. *submitted*.
- CORLESS, R. M., JEFFREY, D. J., AND KNUTH, D. E. 1997. A sequence of series for the Lambert W function. In *ACM International Symposium on Symbolic and Algebraic Computation*. 195–203.
- DE BRUIJN, N. G. 1961. *Asymptotic Methods in Analysis*. North-Holland.
- GEDDES, K. O., CZAPOR, S. R., AND LABAHN, G. 1992. *Algorithms for Computer Algebra*. Kluwer.
- GRAHAM, R. L., KNUTH, D. E., AND PATASHNIK, O. 1994. *Concrete Mathematics*. Addison-Wesley.
- JEFFREY, D. J., HARE, D. E. G., AND CORLESS, R. M. 1996. Unwinding the branches of the Lambert W function. *Mathematical Scientist* 21, 1–7.
- KAHAN, W. AND DARCY, J. 1998. How Java’s floating-point hurts everyone everywhere. <http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf>.
- KAHAN, W. M. 1986. Branch cuts for complex elementary functions, or, much ado about nothing’s sign bit. In *The state of the art in numerical analysis: Proceedings of the Joint IMA/SIAM Conference*, M. J. D. Powell and A. Iserles, Eds. Oxford University Press.
- MARSAGLIA, G. AND MARSAGLIA, J. C. 1990. A new derivation of Stirling’s approximation to $n!$. *American Mathematical Monthly* 97, 826–829.
- MOSES, J. 1971. Algebraic simplification: a guide for the perplexed. *Commun. ACM* 14, 8, 527–537.
- RICHARDSON, D. 1968. Some undecidable problems involving elementary functions of a real variable. *The Journal of Symbolic Logic* 33, 4 (December), 514–520.
- RICHARDSON, D. 1992. The elementary constant problem. In *ACM International Symposium on Symbolic and Algebraic Computation*. ACM Press, 108–116.
- VALLURI, S. R., CORLESS, R. M., AND JEFFREY, D. J. 2000. Some applications of the Lambert W function to physics. *Canadian Journal of Physics* 78, 823–831.
- VON ZUR GATHEN, J. AND GERHARD, J. 1999. *Modern Computer Algebra*. Cambridge University Press.
- WEIBEL, T. AND GONNET, G. H. 1993. An assume facility for CAS, with a sample implementation for Maple. In *Proceedings of the International Symposium on Design and Implementation of Symbolic Computation Systems*. Springer-Verlag, 95–103.

Appendix A: Code for series

```

‘series/Wrightomega’ := proc()
  local a, ans, c, f, i, k, lc, ld, lt, p, s, stirlingcycle, t, x;
  option ‘Copyright (c) 2004 by Robert M. Corless. All rights reserved.’;

  f := args[1]; # typically, f(x) = a + x
  x := args[2];

  try
    t := series( f, x, Order+1 ); # Guard order may be needed
  
```

```

if t=0 then
  return series( Wrightomega(0), x )
elif not type(t,'series') then
  return FAIL;
end if;

# The series may be disguised zero
t := map( Normalizer, t );
while (t<>0) and (op(2,t)<0) and (op(1,t)<>0(1)) and
  Testzero(op(1,t)) do
  t:=subsop(1=0,t);
end do;

if t=0 then
  return series( Wrightomega(0), x )
else
  # Compose series for t with that for omega
  # 4 different cases: regular points, branch points, -infinity, +/complex infinity)
  ld := op(2,t); # leading order of 1st term
  lc := op(1,t); # leading coefficient
  lt := lc*x^ld;
  if ld < 0 then
    # Either series at infinity or series at -infinity
    if lc < 0 then
      # series at -infinity (by convention x -> 0+)
      # omega( f(x) ) = sum( (-n)^(n-1)*exp(n*t)/n!, n=1..infinity )
      s := exp(t);
      ans := add( (-i)^(i-1)/i!*s^i, i=1..Order-1 ) + O( s^Order );
      return ans
    else
      # series at +infinity (lc <> 0 by prior computation)
      # Use Comtet's form with Stirling cycle number coefficients
      stirlingcycle := (n,m) -> (-1)^(n-m)*combinat[stirling1](n,m);
      c := (ell,m) -> (-1)^ell*stirlingcycle(ell+m,ell+1)/m!;
      ans := series( t - ln(t), x );
      for i to Order-1 do
        ans := series( ans + add( c(i-m,m)*ln(t)^m, m=1..i )/t^i, x );
      end do;
      return ans
    end if;
  end if;
else
  # Either regular point or branch point
  if ld=0 and member( Normalizer(lc), {-1+I*Pi,-1-I*Pi} ) then
    # branch point
    s := Normalizer( (lc+1)/I/Pi ); # +1 or -1
    # wrong on branch cut, but series/fracpower incapable yet
    # truly, p := -s*I*conjugate( sqrt( 2*conjugate( t-lc ) ) );
    #p := map(q->evalc(conjugate(q)),
    #      series( -s*I*sqrt(2*(map(q->evalc(evalc(conjugate(q))), t-lc))), x ) );
    p := series( -s*I*sqrt(2*(t-lc)), x );
    WARNING(sprintf("The series has wrong signs if argument to "
    "Wrightomega is on a branch cut Re <=-1, Im = +/-Pi ", f) );
    a := array(0..Order-1);
    a[0] := 1;
    a[1] := 1;
    # Marsaglia and Marsaglia series

```

```

        for k from 2 to Order-1 do
            a[k] := (a[k-1] - add( i*a[i]*a[k+1-i], i=2..k-1 ))/(k+1)/a[1];
        end do;
        ans := series( add( -a[k]*p^k, k=0..Order-1 ) , x );
        return ans
    else
        # regular point
        return 'series/function'( Wrightomega(t), x )
    end if;
end if;
end if;
catch :
    return FAIL
end try;
end:

```

Appendix B: Code for simplification

```

'simplify/Wrightomega' := proc(sinput)
    local Wterm, a, ans, b, c, fact, fact_list, w_facts,
        rest_facts, rest_prod, result, match_function, matches,
        k, z, zi, s, branch, Z, r, m, rng, rats, i, j, 'SureNotOnBranch?', 'lnplus?', djj_a, djj_b, djj_s;
    option remember, system,
        'Copyright (c) 2004 by Robert M. Corless and David J. Jeffrey. All rights reserved.';

    'SureNotOnBranch?' := proc( x ) local z1, z2;
        Normalizer := 'if'(type(x,complex(numeric)),evalf,t->t);
        # assume names are real, as does evalc
        if is(x::real) or evalb(evalc(Re(x))=x) then
            return true
        end;
        try
            z1 := Normalizer( x - I*Pi );
            z2 := Normalizer( x + I*Pi );
            # is( -3 + 0.0*I < -1 ) returns FAIL, which we don't want
            return evalb( not (is(z1::real) and is( Re(z1) <= -1 )) and
                not (is(z2::real) and is( Re(z2) <= -1 )) )
        catch :
            return false
        end try
    end proc;

    # Convert all LambertW to Wrightomegas
    s := convert( sinput, 'Wrightomega' );

    if not has(s,'Wrightomega') or type(s,'name') then
        return s;

    elif type(s,'function') and op(0,s) = 'ln' then
        if type(op(1,s),'function') and op([1,0],s) = 'Wrightomega'
            and 'SureNotOnBranch?'(op([1,1],s)) then
            # Apply the simplification
            # ln( omega(x) ) = x - omega(x)
            return op(1,op(1,s)) - op(1,s);
        elif type(op(1,s),'*') then
            # Apply the simplification

```

```

#   ln( a*Wrightomega(x) ) =
#   ln(a)+ln(Wrightomega(x)) - 2*Pi*I*unwindK(%)
Wterm := map(proc(f) if type(f,'function') and
              op(0,f) = 'Wrightomega' then
                f else 1 fi end, op(1,s));
a := op(1,s)/Wterm;
if type(Wterm,'function') then
    ans := ln(a) + 'simplify/Wrightomega'( ln(Wterm) );
return ans - 2*Pi*I*unwindK(ans)
fi;

fi;

elif type(s,'function') and op(0,s) = 'Wrightomega' then
a := op(1,s);
# s = Wrightomega(a) was input.
# Look for a = b + ln(c)
# note that b and c may themselves contain ln
#
'lnplus?' := proc( expr, b, c )
local eops, z, logs, 'type/lnfun';
'type/lnfun' := proc( e )
    evalb( type(e,function) and op(0,e)=ln )
end proc;
if not has(expr,ln) then
    return false
elif 'type/lnfun'(expr) then
    b := 0;
    c := op(1,expr);
    return true
elif not type(expr, '+') then
    return false
else
    eops := op(expr);
    ASSERT( expr = add( e, e in [eops] ) );
    logs := select( t->'type/lnfun'(t), {eops} );
    if logs = {} then
        return false
    end if;
    logs := convert( logs, list );
    # sort log terms in increasing length order
    logs := sort( logs, (a,b)->evalb(length(a)<length(b)) );
    c := op(1,logs[nops(logs)] );
    b := expr - ln(eval(c));
    return true
end if
end proc;

if 'lnplus?(a,'b','c') then
ASSERT( s = Wrightomega(a) );
ASSERT( a = b + ln(c) );
# b might be zero, or a = b + ln(something), or
# it might be more complicated.
# Equations (15), (16) and (17) from the paper:
# eq (15) b = x
# eq (16) b = x-2*Pi*I

```

```

# eq (17) b = x - Pi*I
if b<> 0 then
  if is(a=b+ln(b)) then
    return b
  elif is(a=b+ln(b+2*Pi*I) and b+2*Pi*I <=-1) then
    return b+2*Pi*I
  elif is(a=b+ln(abs(b+Pi*I)) and b+Pi*I <=-1) then
    return b+Pi*I
  fi;
fi;
# Added by DJJ Monday 6th
userinfo(5,'Wrightomega',a,b,c);
# Implement integer simplification search
if b=0 then # we are looking for a=ln(x ln y), i.e. c= djj_a* ln(djj_b)
  if op(0,c)='*' then
    if typematch(op(1,c),ln(djj_a::integer ), 'djj_b') then
      djj_a:=op(1,c);
      djj_s:= round(evalf( Wrightomega(a)/djj_a) );
      userinfo(5,'Wrightomega',djj_a,djj_b,djj_s);
    elif typematch(op(2,c),ln(djj_a::integer),'djj_b') then
      djj_a:=op(2,c);
      djj_s:= round(evalf( Wrightomega(a)/djj_a));
      userinfo(5,'Wrightomega',djj_a,djj_b,djj_s);
    fi;
    djj_b:=djj_s*djj_a;
    userinfo(5,'Wrightomega',djj_b);
    if is(a=djj_b + ln(djj_b)) then return djj_b; fi;
  fi;
else # b>0 and we are looking for a=ln x + ln(ln(y)), i.e. b=ln(x), c= ln(y)
  if typematch(c,ln(djj_a::integer),'djj_b') then
    djj_a:= c;
    djj_s:= round(evalf( Wrightomega(a)/djj_a) );
    djj_b:=djj_s*djj_a;
    userinfo(5,'Wrightomega',djj_a,djj_b,djj_s);
    if is(a=djj_b + ln(djj_b)) then return djj_b; fi;
  fi;
fi;

# Apply the complex simplifications
# from the Theorem in Section 3.2
# omega(ln[n](2*Pi*I*(n+r)*exp(2*Pi*I*(m+r))))
# -> 2*Pi*I*(n+r) if m::integer, n::integer, and r is in the range
# -3/4 < r <= 1/4 if n >0, -1/4 < r <= 1/4 if n=0, and
# -1/4 < r <= 3/4 if n < 0 (Corless & Jeffrey 2004)
# Check if b = 2*Pi*I*branch for some integer branch
if is( b/2/Pi/I, integer ) then
  branch := eval(b/2/Pi/I);
  # Look for c=2*Pi*I*(branch+r)*exp(2*Pi*I*(m+r))
  b := 'b'; z := 'z'; matches := 'matches';
  if match( eval(c,Pi=Z/I)=Z*b*exp(Z*z), Z, matches) then
    r := eval( b/2, matches )-branch;
    # Use "signum" to decide if branch is positive, zero, or negative
    # if this fails, return the range (0,0)
    # so the "is" test for r will fail unless r=0.

```

```

# Note that the user may have been perverse enough to
# set _Envsignum0, so we must reset it (we need 0 )
_Envsignum0 := 0;
rng := 'if'( signum(branch)= 1, RealRange(Open(-3/4),1/4),
             'if'( signum(branch)=-1, RealRange(Open(-1/4),3/4),
             'if'( signum(branch)= 0, RealRange(Open(-1/4),1/4),
             RealRange(0,0) ) ) );

if is( r, rng ) then
  m := eval( z/2, matches ) - r;
  if is( m, integer ) then
    return 2*Pi*I*(branch+r);
  end if;
end if;

# RMC December 2003 Same as above but in the case of small
# rational r when exp has disappeared

# The list of "small known rationals" is built anew
# each time the routine hits here; but because exp
# has option remember, this is likely not too onerous.
# Building the list automatically ensures that additions
# such as 2*Pi*I/5 (not known to Maple at the time of
# writing, but potentially known) will not be excluded.
# Probably generating this list at load time would be best...

rats := { seq( seq( i/j, i=-j-1..j-1 ), j=2..48 ) };
rats := select( t->not(has(exp(2*Pi*I*t),exp)), rats );
_Envsignum0 := 0;
if signum(branch) = 1 then
  rats := select( t->is(t,RealRange(Open(-3/4),1/4)), rats );
elif signum(branch) = -1 then
  rats := select( t->is(t,RealRange(Open(-1/4),3/4)), rats );
elif signum(branch)=0 then
  rats := select( t->is(t,RealRange(Open(-1/4),1/4)), rats );
else
  # signum of branch not known. Don't try the simplification
  # unless r = 0 (since branch is integral, this works)
  rats := {0};
end if;

for r in rats do
  if simplify(radnormal(a-ln(2*Pi*I*(branch+r))*exp(2*Pi*I*r))-2*Pi*I*branch ) = 0 then
    return 2*Pi*I*(branch+r)
  end if;
end do;
end if;
end if;
end if;

map(procname,args);
end:

```