Using the ordinal calculator

For ordCalc_0.3.2

Paul Budnik
Mountain Math Software
paul@mtnmath.com

Contents

About this manual

This manual duplicates the online documentation. In this formatted version of the documentation, computer input and output is in `tty font`. For a complete description of the program and the mathematics on which it is based see the paper "A Computational Approach to the Ordinal Numbers"[3]. That document includes this manual as an appendix.

1 Introduction

The Ordinal Calculator is an interactive tool for understanding the hierarchies of recursive and countable ordinals[7, 6, 5]. It is also a research tool to help to expand these hierarchies. Its motivating goal is ultimately to expand the foundations of mathematics by using computer technology to manage the combinatorial explosion in complexity that comes with explicitly defining the recursive ordinals implicitly defined by the axioms of Zermelo-Frankel set theory[4, 1]. The underlying philosophy focuses on what formal systems tell us about physically realizable combinatorial processes[2].

The source code and documentation is licensed for use and distribution under the Gnu General Public License, Version 2, June 1991 and subsequent versions. A copy of this license must be distributed with the program. It is also at:

`http://www.gnu.org/licenses/gpl-2.0.html`. The ordinal calculator source code, documentation and some executables can be downloaded from: `http://www.mtnmath.com/ord` or `https://sourceforge.net/projects/ord`.

Most of this manual is automatically extracted from the online documentation.

This is a command line interactive interface to a program for exploring the ordinals. It supports recursive ordinals through and beyond the Veblen hierarchy[7]. It defines notations for the Church-Kleene ordinal (the ordinal of the recursive ordinals) and some larger countable ordinals. They are used in a form of ordinal collapsing to define large recursive ordinals.

1.1 Command line options

The standard name for the ordinal calculator is `ord`. Typing `ord` (or `./ord`) ENTER will start `ord` in command line mode on most Unix or Linux based systems. The command line options are mostly for validating or documenting `ord`. They are:

'`cmd`' — Read specified command file and enter command line mode.
'`cmdTex`' — Read specified command file and enter TeX command line mode.
'`version`' — Print program version.
'`help`' — Describe command line options.
'`cmdDoc`' — Write manual for command line mode in TeX format.
'`tex`' — Output TeX documentation files.
'`psi`' — Do tests of Veblen hierarchy.
'`base`' — Do tests of base class Ordinal.
'`try`' — Do tests of class FiniteFuncOrdinal.
'`gamma`' — Test for consistent use of gamma and epsilon.
'`iter`' — Do tests of class IterFuncOrdinal.
'`admis`' — Do tests of class AdmisLevOrdinal.
'`admis2`' — Do additional tests of class AdmisLevOrdinal.
'`play`' — Do integrating tests.
'`descend`' — Test descending trees.
'`collapse`' — Ordinal collapsing tests.
'`nested`' — Ordinal nested collapsing tests.
'`nested2`' — Ordinal nested collapsing tests 2.
'`nested3`' — Ordinal nested collapsing tests 3.
'`exitCode`' — LimitElement exit code base test.
'`exitCode2`' — LimitElement exit code base test 2.
'`limitEltExitCode`' — Admissible level LimitElement exit code test 0.
'`limitEltExitCode1`' — Admissible level limitElement exit code test 1.
'`limitEltExitCode2`' — Admissible level limitElement exit code test 2.
'`limitEltExitCode3`' — Admissible level limitElement exit code test 3.
'`limitOrdExitCode`' — Admissible level limitOrd exit code test.
'`limitOrdExitCode1`' — Admissible level limitOrd exit code test.
'`limitOrdExitCode2`' — Admissible level limitOrd exit code test.
'`limitOrdExitCode3`' — Admissible level limitOrd exit code test.
'`admisLimitElementExitCode`' — Admissible level limitElement exit code test.
'`admisDrillDownLimitExitCode`' — Admissible level drillDownLimitElement exit code test.
'`admisDrillDownLimitComExitCode`' — Admissible level drillDownLimitElementCom exit code test.
'`admisLimitElementComExitCode`' — Admissible level exit code test for limitElmentCom.
'`admisExamples`' — Admissible level tests of examples.
'`nestedLimitEltExitCode`' — Nested level limitElement exit code test.
'`nestedLimitEltExitCode2`' — Nested level limitElement exit code test 2.
'`nestedBaseLimitEltExitCode`' — Nested level base class limit exit code test.
'`nestedLimitOrdExitCode`' — Nested level limitOrd exit code test.

‘`nestedCmpExitCode`’ — Nested level compare exit code test.
‘`nestedEmbedNextLeast`’ — Nested embed next least test.
‘`transition`’ — Admissible level transition test.
‘`cmpExitCode`’ — Admissible level compare exit code test.
‘`drillDownExitCode`’ — Admissible level compare exit code test.
‘`embedExitCode`’ — Admissible level compare exit code test.
‘`fixedPoint`’ — test fixed point detection.
‘`nestedEmbed`’ — basic nested embed tests.
‘`nestedEmbedPaperTables`’ — test tables in ordarith paper.
‘`nestedEmbedPaperTables2`’ — test tables in ordarith paper.
‘`nestedEmbedPaperTables3`’ — test tables in ordarith paper.
‘`infoLimitTypeExamp`’ — test inforLimitTypeExamples.
‘`cppTest`’ — test interactive C++ code generation.
‘`cppTestTeX`’ — test interactive C++ code that generates TeX.
‘`test`’ — a stub at the end of validate.cpp used to degug code.
‘`helpTex`’ — TeX document command line options.
‘`paper`’ — generate tables for paper on the calculator.

1.2 Help topics

Following are topics with more information. Type ‘`help` topic’ to access them.
‘`cmds`’ – lists commands.
‘`defined`’ – list predefined ordinal variables.
‘`compare`’ – describes comparison operators.
‘`members`’ – describes member functions.
‘`ordinal`’ – describes available ordinal notations.
‘`ordlist`’ – describes ordinal lists and their use.
‘`purpose`’ – describes the purpose and philosophy of this project.
‘`syntax`’ – describes syntax.
‘`version`’ – displays program version.

This program supports line editing and history.

2 Ordinals

Ordinals are displayed in plain text (the default) and/or LaTeX format. (Enter ‘`help opts`’ to control this.) The finite ordinals are the nonnegative integers. The ordinal operators are +, * and ^ for addition, multiplication and exponentiation. Exponentiation has the highest precedence. Parenthesis can be used to group subexpressions.

The ordinal of the integers, `omega`, is also represented by the single lowercase letter: ‘`w`’. The Veblen function is specified as ‘`psi(p1,p2,...,pn)`’ where n is any integer > 0. Special notations are displayed in some cases. Specifically `psi(x)` is displayed as `w^x`. `psi(1,x)` is displayed as `epsilon(x)`. `psi(1,0,x)` is displayed as `gamma(x)`. In all cases the displayed version can be entered as input.

Larger ordinals in the Veblen hierarchy are specified as `psi_{1}(a1,a2,...,an)`. The first parameter is enclosed in {brackets} not parenthesis. `psi_{1}` is defined as the union of `w, epsilon(0), gamma(0), psi(1, 0, 0, 0), psi(1, 0, 0, 0, 0), psi(1, 0, 0, 0, 0, 0), psi(1, 0, 0, 0, 0, 0, 0), ...` You can access the sequence whose union is a specific ordinal using member functions. Type `help members` to learn more about this. Larger notations beyond the recursive ordinals are also available. The format for the countable admissible ordinals and their countable limits is `omega_{k}`. To partially fill the gaps introduced with these notations, the following additional notations are used:
`omega_{k,g}(a1,a2,...,an)`, `omega_{k}[e]`,
`[[d1/s1,d2/s2,..,dn/sn]]omega_{k,g}(a1,a1,...,an)`,
`[[d1/s1,d2/s2,..,dn/sn]]omega_{k}[e]`,
`[[d1/s1,d2/s2,..,dn/sn]]omega_{k}[[e]]`.
See the documentation 'A Computational Approach to the Ordinal Numbers'[3] to learn more about these notations.

There are several predefined ordinals. '`w`' and '`omega`' can be be used interchangeably for the ordinal of the integers. '`eps0`' and '`omega1CK`' are also predefined. Type '`help defined`' to learn more.

3 Predefined ordinals

The predefined ordinal variables are:
`omega` $= \omega$
`w` $= \omega$
`omega1CK` $= \omega_1$
`w1` $= \omega_1$
`w1CK` $= \omega_1$
`eps0` $= \varepsilon_0$

4 Syntax

The syntax is that of restricted arithmetic expressions and assignment statements. The tokens are variable names, nonnegative integers and the operators: +, `*` and `^` (addition, multiplication and exponentiation). Comparison operators are also supported. Type '`help comparison`' to learn about them. The letter '`w`' is predefined as omega the ordinal of the integers. Type '`help defined`' for a list of all predefined variables. To learn more about ordinals type '`help ordinal`'. C++ style member functions are supported with a '.' separating the variable name (or expression enclosed in parenthesis) from the member function name. Enter ‘`help members`’ for the list of member functions.

An assignment statement or ordinal expression can be entered and it will be evaluated and displayed in normal form. Typing ‘`help opts`’ lists the display options. Assignment statements are stored. They can be listed (command ‘`list`’) and their value can be used in subsequent expressions. All statements end at the end of a line unless the last character is ‘\’. Lines can be continued indefinitely. Comments must be preceded by either ‘`%`’ or ‘`//`’.

Commands can be entered as one or more names separated by white space. File names should be enclosed in double quotes (`"`) if they contain any non alphanumeric characters such as dot, ‘.’. Command names can be used as variables. Enter ‘`help cmds`’ to get a list of commands and their functions.

5 Ordinal lists

Lists are a sequence of ordinals. An assignment statement can name a single ordinal or a list of them separated by commas. In most circumstances only the first element in the list is used, but some functions (such as member function ‘`limitOrdLst`’) use the full list. Type ‘`help members`’ to learn more about ‘`limitOrdLst`’.

6 Commands

6.1 All commands

The following commands are available:
‘`cmpCheck`’ – toggle comparison checking for debugging.
‘`cppList`’ – list the C++ code for assignments or write to an optional file.
‘`examples`’ – shows examples.
‘`exit`’ – exits the program.
‘`exportTeX`’ – exports assignments statements in TeX format.
‘`help`’ – displays information on various topics.
‘`list`’ – lists assignment statements.
‘`log`’ – write log file (ord.log default) (see help logopt).
‘`listTeX`’ – lists assignment statements in TeX format.
‘`logopt`’ – control log file (see help log).
‘`name`’ – toggle assignment of names to expressions.
‘`opts`’ – controls display format and other options.
‘`prompt`’ – prompts for ENTER with optional string argument.
‘`quit`’ – exits the program.
‘`quitf`’ – exits only if not started in interactive mode.
‘`read`’ – read “input file” (ord_calc.ord default).
‘`readall`’ – same as read but no ’wait for ENTER‘ prompt.
‘`save`’ – saves assignment statements to a file (ord_calc.ord default).

‘`setDbg`’ – set debugging options.
‘`tabList`’ – lists assignment values as C++ code to generate a LaTeX table.
‘`yydebug`’ – enables parser debugging (off option).

6.2 Commands with options

Following are the commands with options.

Command ‘`examples`’ – shows examples.
It has one parameter with the following options.
‘`arith`’ – demonstrates ordinal arithmetic.
‘`compare`’ – shows compare examples.
‘`display`’ – shows how display options work.
‘`member`’ – demonstrates member functions.
‘`VeblenFinite`’ – demonstrates Veblen functions of a finite number of ordinals.
‘`VeblenExtend`’ – demonstrates Veblen functions iterated up to a recursive ordinal.
‘`admissible`’ – demonstrates admissible level ordinal notations.
‘`admissibleDrillDown`’ – demonstrates admissible notations dropping down one level.
‘`admissibleContext`’ – demonstrates extended admissible ordinal parameters.
‘`list`’ – shows how lists work.
‘`desLimitOrdLst`’ – shows how to construct a list of descending trees.

Command ‘`logopt`’ – control log file (see help log).
It has one parameter with the following options.
‘`flush`’ – flush log file.
‘`stop`’ – stop logging.

Command ‘`opts`’ – controls display format and other options.
It has one parameter with the following options.
‘`both`’ – display ordinals in both plain text and TeX formats.
‘`tex`’ – display ordinals in TeX format only.
‘`text`’ – display ordinals in plain text format only (default).
‘`psi`’ – additionally display ordinals in Psi format (turned off by the above options).
‘`promptLimit`’ – lines to display before pausing, < 4 disables pause.

Command ‘`setDbg`’ – set debugging options.
It has one parameter with the following options.
‘`all`’ – turn on all debugging.
‘`arith`’ – debug ordinal arithmetic.
‘`clear`’ – turn off all debugging.
‘`compare`’ – debug compare.
‘`exp`’ – debug exponential.

'`limArith`' – limited debugging of arithmetic.
'`limit`' – debug limit element functions.
'`construct`' – debug constructors.

7 Member functions

Every ordinal (except 0) is the union of smaller ordinals. Every limit ordinal is the union of an infinite sequence of smaller ordinals. Member functions allow access to to these smaller ordinals. One can specify how many elements of this sequence to display or get the value of a specific instance of the sequence. For a limit ordinal, the sequence displayed, were it extended to infinity and its union taken, that union would equal the original ordinal.

The syntax for a member function begins with either an ordinal name (from an assignment statement) or an ordinal expression enclosed in parenthesis. This is followed by a dot (.) and then the member function name and its parameters enclosed in parenthesis. The format is '`ordinal_name.memberFunction(p)`' where `p` may be optional. Functions '`limitOrdLst`' and '`desLimitOrdLst`' return a list. All other member functions return a scalar value. Unless specified otherwise, the returned value is that of the ordinal the function was called from.

The member functions are:

'`cpp`' – output C++ code to define this ordinal.
'`descend`' – (n,m) iteratively (up to m) take nth limit element.
'`descendFull`' – (n,m,k) iteratively (up to m) take n limit elements with root k.
'`desLimitOrdLst`' – (depth, list) does limitOrdLst iteratively on all outputs depth times.
'`ek`' – effective kappa (or indexCK) for debugging.
'`getCompareIx`' – display admissible compare index.
'`isValidLimitOrdParam`' – return true or false.
'`iv`' – alias for isValidLimitOrdParam.
'`le`' – evaluates to specified finite limit element.
'`lec`' – alias to return limitExitCode (for debugging).
'`limitElement`' – an alias for 'le'.
'`limitExitCode`' – return limitExitCode (for debugging).
'`listLimitElts`' – lists specified (default 10) limit elements.
'`listElts`' – alias for listLimitElts.
'`limitOrd`' – evaluates to specified (may be infinite) limit element.
'`limitOrdLst`' – apply each input from list to limitOrd and return that list.
'`lo`' – alias for limitOrd.
'`limitType`' – return limitType.
'`maxLimitType`' – return maxLimitType.
'`maxParameter`' – return maxParameter (for debugging).
'`blt`' – return baseLimitTypeEquiv if defined for first term (for debugging).

8 Comparison operators

Any two ordinals or ordinal expressions can be compared using the operators: `<, <=, >, >=` and `==`. The result of the comparison is the text either `TRUE` or `FALSE`. Comparison operators have lower precedence than ordinal operators.

9 Examples

In the examples a line that begins with the standard prompt '`ordCalc> `' contains user input. All other lines contain program output

To select an examples type '`examples`' followed by one of the following options.
'`arith`' – demonstrates ordinal arithmetic.
'`compare`' – shows compare examples.
'`display`' – shows how display options work.
'`member`' – demonstrates member functions.
'`VeblenFinite`' – demonstrates Veblen functions of a finite number of ordinals.
'`VeblenExtend`' – demonstrates Veblen functions iterated up to a recursive ordinal.
'`admissible`' – demonstrates admissible level ordinal notations.
'`admissibleDrillDown`' – demonstrates admissible notations dropping down one level.
'`admissibleContext`' – demonstrates extended admissible ordinal parameters.
'`list`' – shows how lists work.
'`desLimitOrdLst`' – shows how to construct a list of descending trees.

9.1 Simple ordinal arithmetic

The following demonstrates ordinal arithmetic.

```
ordCalc> a=w^w
Assigning ( w^w ) to 'a'.
ordCalc> b=w*w
Assigning ( w^2 ) to 'b'.
ordCalc> c=a+b
Assigning ( w^w ) + ( w^2 ) to 'c'.
ordCalc> d=b+a
Assigning ( w^w ) to 'd'.
```

9.2 Comparison operators

The following shows compare examples.

```
ordCalc> psi(1,0,0) == gamma(0)
gamma( 0 ) == gamma( 0 ) ::  TRUE
```

```
ordCalc> psi(1,w) == epsilon(w)
epsilon( w) == epsilon( w) ::  TRUE
ordCalc> w^w < psi(1)
( w^w ) < w ::  FALSE
ordCalc> psi(1)
Normal form:  w
```

9.3 Display options

The following shows how display options work.

```
ordCalc> a=w^(w^w)
Assigning ( w^( w^w ) ) to ‘a’.
ordCalc> b=epsilon(a)
Assigning epsilon( ( w^( w^w ) )) to ‘b’.
ordCalc> c=gamma(b)
Assigning gamma( epsilon( ( w^( w^w ) )) ) to ‘c’.
ordCalc> list
a = ( w^( w^w ) )
b = epsilon( ( w^( w^w ) ))
c = gamma( epsilon( ( w^( w^w ) )) )
%Total 3 variables listed.
ordCalc> opts tex
ordCalc> list
a = \omega{}^{\omega{}^{\omega{}}}
b = \varepsilon_{\omega{}^{\omega{}^{\omega{}}}}
c = \Gamma_{\varepsilon_{\omega{}^{\omega{}^{\omega{}}}}}
%Total 3 variables listed.
ordCalc> opts both
ordCalc> list
a = ( w^( w^w ) )
a = \omega{}^{\omega{}^{\omega{}}}
b = epsilon( ( w^( w^w ) ))
b = \varepsilon_{\omega{}^{\omega{}^{\omega{}}}}
c = gamma( epsilon( ( w^( w^w ) )) )
c = \Gamma_{\varepsilon_{\omega{}^{\omega{}^{\omega{}}}}}
%Total 3 variables listed.
```

9.4 Member functions

The following demonstrates member functions.

```
ordCalc> a=psi(1,0,0,0,0)
```

```
Assigning psi( 1, 0, 0, 0, 0 ) to 'a'.
ordCalc> a.listElts(3)

3 limitElements for psi( 1, 0, 0, 0, 0 )
le(1) = psi( 1, 0, 0, 0 )
le(2) = psi( psi( 1, 0, 0, 0 ) + 1, 0, 0, 0 )
le(3) = psi( psi( psi( 1, 0, 0, 0 ) + 1, 0, 0, 0 ) + 1, 0, 0, 0 )
End limitElements

Normal form:  psi( 1, 0, 0, 0, 0 )
ordCalc> b=a.le(6)
Assigning psi( psi( psi( psi( psi( psi( 1, 0, 0, 0 ) + 1, 0, 0, 0 ) + 1, 0, 0,
0 ) + 1, 0, 0, 0 ) + 1, 0, 0, 0 ) + 1, 0, 0, 0 ) to 'b'.
```

9.5 Veblen function of N ordinals

The following demonstrates Veblen functions of a finite number of ordinals.

The Veblen function with a finite number of parameters, `psi(x1,x2,...xn)` is built up from the function `omega^x`. `psi(x) = omega^x`. `psi(1,x)` enumerates the fixed points of `omega^x`. This is `epsilon(x)` which equals `psi(1,x)`. Each additional variable diagonalizes the functions definable with existing variables. These functions can have any finite number of parameters.

```
ordCalc> a=psi(w,w)
Assigning psi( w, w ) to 'a'.
ordCalc> b=psi(a,3,1)
Assigning psi( psi( w, w ), 3, 1 ) to 'b'.
ordCalc> b.listElts(3)

3 limitElements for psi( psi( w, w ), 3, 1 )
le(1) = psi( psi( w, w ), 3, 0 )
le(2) = psi( psi( w, w ), 2, psi( psi( w, w ), 3, 0 ) + 1 )
le(3) = psi( psi( w, w ), 2, psi( psi( w, w ), 2, psi( psi( w, w ), 3, 0 ) + 1
) + 1 )
End limitElements

Normal form:  psi( psi( w, w ), 3, 1 )
ordCalc> c=psi(a,a,b,1,3)
Assigning psi( psi( w, w ), psi( w, w ), psi( psi( w, w ), 3, 1 ), 1, 3 ) to 'c'.
ordCalc> c.listElts(3)

3 limitElements for psi( psi( w, w ), psi( w, w ), psi( psi( w, w ), 3, 1 ), 1,
3 )
```

```
le(1) = psi( psi( w, w ), psi( w, w ), psi( psi( w, w ), 3, 1 ), 1, 2 )
le(2) = psi( psi( w, w ), psi( w, w ), psi( psi( w, w ), 3, 1 ), 0, psi( psi(
w, w ), psi( w, w ), psi( psi( w, w ), 3, 1 ), 1, 2 ) + 1 )
le(3) = psi( psi( w, w ), psi( w, w ), psi( psi( w, w ), 3, 1 ), 0, psi( psi(
w, w ), psi( w, w ), psi( psi( w, w ), 3, 1 ), 0, psi( psi( w, w ), psi( w, w
), psi( psi( w, w ), 3, 1 ), 1, 2 ) + 1 ) + 1 )
End limitElements

Normal form:  psi( psi( w, w ), psi( w, w ), psi( psi( w, w ), 3, 1 ), 1, 3 )
```

9.6 Extended Veblen function

The following demonstrates Veblen functions iterated up to a recursive ordinal.

The extended Veblen function, `psi_{a}(x1,x2,...,xn)`, iterates the idea of the Veblen function up to any recursive ordinal. The first parameter is the recursive ordinal of this iteration.

```
ordCalc> a=psi_{1}(1)
Assigning psi_{ 1}(1) to ‘a’.
ordCalc> a.listElts(4)

4 limitElements for psi_{ 1}(1)
le(1) = psi_{ 1} + 1
le(2) = psi( psi_{ 1} + 1, 0 )
le(3) = psi( psi_{ 1} + 1, 0, 0 )
le(4) = psi( psi_{ 1} + 1, 0, 0, 0 )
End limitElements

Normal form:  psi_{ 1}(1)
ordCalc> b=psi_{w+1}(3)
Assigning psi_{ w + 1}(3) to ‘b’.
ordCalc> b.listElts(4)

4 limitElements for psi_{ w + 1}(3)
le(1) = psi_{ w + 1}(2) + 1
le(2) = psi_{ w}(psi_{ w + 1}(2) + 1, 0)
le(3) = psi_{ w}(psi_{ w + 1}(2) + 1, 0, 0)
le(4) = psi_{ w}(psi_{ w + 1}(2) + 1, 0, 0, 0)
End limitElements

Normal form:  psi_{ w + 1}(3)
```

9.7 Admissible ordinal notations

The following demonstrates admissible level ordinal notations.

Countable admissible ordinal notations and their limits are `w_{k}`. The gaps introduced by these notations are partially filled by using notations and ideas from the Veblen hierarchy. The notation for this is `omega_{k,g}(x1,x2,x3,..,xn)`. The first parameter is the admissible level. `omega_{1}` is the Church-Kleene ordinal, the ordinal of the recursive ordinals. The remaining parameters are similar to those defined in the Veblen hierarchy.

```
ordCalc> a=w_{1}(1)
Assigning omega_{ 1}(1) to 'a'.
ordCalc> a.listElts(4)

4 limitElements for omega_{ 1}(1)
le(1) = omega_{ 1}
le(2) = psi_{ omega_{ 1} + 1}
le(3) = psi_{ psi_{ omega_{ 1} + 1} + 1}
le(4) = psi_{ psi_{ psi_{ omega_{ 1} + 1} + 1} + 1}
End limitElements

Normal form:  omega_{ 1}(1)
```

9.8 Admissible notations drop down parameter

The following demonstrates admissible notations dropping down one level.

Admissible level ordinals have the a limit sequence defined in terms of lower levels. The lowest admissible level is that of recursive ordinals. To implement this definition of limit sequence, a trailing parameter in square brackets is used. This parameter (if present) defines an ordinal at one admissible level lower than indicated by other parameters.

```
ordCalc> a=w_{1}[1]
Assigning omega_{ 1}[ 1] to 'a'.
ordCalc> a.listElts(4)

4 limitElements for omega_{ 1}[ 1]
le(1) = w
le(2) = psi_{ w}
le(3) = psi_{ psi_{ w} + 1}
le(4) = psi_{ psi_{ psi_{ w} + 1} + 1}
End limitElements

Normal form:  omega_{ 1}[ 1]
ordCalc> b=w_{1}
```

```
Assigning omega_{ 1} to ‘b’.
ordCalc> c=b.limitOrd(w^3)
Assigning omega_{ 1}[ ( w^3 )] to ‘c’.
ordCalc> c.listElts(4)

4 limitElements for omega_{ 1}[ ( w^3 )]
le(1) = omega_{ 1}[ ( w^2 )]
le(2) = omega_{ 1}[ (( w^2 )*2 )]
le(3) = omega_{ 1}[ (( w^2 )*3 )]
le(4) = omega_{ 1}[ (( w^2 )*4 )]
End limitElements

Normal form:  omega_{ 1}[ ( w^3 )]
ordCalc> d=w_{5,c}(3,0)
Assigning omega_{ 5, omega_{ 1}[ ( w^3 )]}(3, 0) to ‘d’.
ordCalc> d.listElts(4)

4 limitElements for omega_{ 5, omega_{ 1}[ ( w^3 )]}(3, 0)
le(1) = omega_{ 5, omega_{ 1}[ ( w^3 )]}(2, 1)
le(2) = omega_{ 5, omega_{ 1}[ ( w^3 )]}(2, omega_{ 5, omega_{ 1}[ ( w^3 )]}(2,
1) + 1)
le(3) = omega_{ 5, omega_{ 1}[ ( w^3 )]}(2, omega_{ 5, omega_{ 1}[ ( w^3 )]}(2,
omega_{ 5, omega_{ 1}[ ( w^3 )]}(2, 1) + 1) + 1)
le(4) = omega_{ 5, omega_{ 1}[ ( w^3 )]}(2, omega_{ 5, omega_{ 1}[ ( w^3 )]}(2,
omega_{ 5, omega_{ 1}[ ( w^3 )]}(2, omega_{ 5, omega_{ 1}[ ( w^3 )]}(2, 1) + 1)
+ 1) + 1)
End limitElements

Normal form:  omega_{ 5, omega_{ 1}[ ( w^3 )]}(3, 0)
```

9.9 Admissble notation parameters

The following demonstrates extended admissible ordinal parameters.

The context parameter in admissible level ordinals allows one to use any notation at any admissible level to define notations at any lower admissible level or to define recursive ordinals.

```
ordCalc> a=w_{1}(4)
Assigning omega_{ 1}(4) to ‘a’.
ordCalc> a.listElts(4)

4 limitElements for omega_{ 1}(4)
le(1) = omega_{ 1}(3)
le(2) = psi_{ omega_{ 1}(3) + 1}(3)
le(3) = psi_{ psi_{ omega_{ 1}(3) + 1}(3) + 1}(3)
le(4) = psi_{ psi_{ psi_{ omega_{ 1}(3) + 1}(3) + 1}(3) + 1}(3)
End limitElements

Normal form:  omega_{ 1}(4)
ordCalc> b=w_{1}(1)
Assigning omega_{ 1}(1) to ‘b’.
ordCalc> b.listElts(4)

4 limitElements for omega_{ 1}(1)
le(1) = omega_{ 1}
le(2) = psi_{ omega_{ 1} + 1}
le(3) = psi_{ psi_{ omega_{ 1} + 1} + 1}
le(4) = psi_{ psi_{ psi_{ omega_{ 1} + 1} + 1} + 1}
End limitElements

Normal form:  omega_{ 1}(1)
ordCalc> c=w_{1,7}
Assigning omega_{ 1, 7} to ‘c’.
ordCalc> c.listElts(4)

4 limitElements for omega_{ 1, 7}
le(1) = omega_{ 1, 6}(omega_{ 1, 6} + 1)
le(2) = omega_{ 1, 6}(omega_{ 1, 6} + 1, 0)
le(3) = omega_{ 1, 6}(omega_{ 1, 6} + 1, 0, 0)
le(4) = omega_{ 1, 6}(omega_{ 1, 6} + 1, 0, 0, 0)
End limitElements

Normal form:  omega_{ 1, 7}
ordCalc> r=[[1]]w_{1}
Assigning [[1]]omega_{ 1} to ‘r’.
ordCalc> r.listElts(4)

4 limitElements for [[1]]omega_{ 1}
le(1) = [[1]]omega_{ 1}[[ 1]]
le(2) = [[1]]omega_{ 1}[[ 2]]
le(3) = [[1]]omega_{ 1}[[ 3]]
le(4) = [[1]]omega_{ 1}[[ 4]]
End limitElements

Normal form:  [[1]]omega_{ 1}
```

9.10 Lists of ordinals

The following shows how lists work.

Lists are a sequence of ordinals (including integers). A list can be assigned to a variable just as a single ordinal can be. In most circumstances lists are evaluated as the first ordinal in the list. In '`limitOrdLst`' all of the list entries are used. These member functions return a list with an input list

```
ordCalc> lst = 1, 12, w, gamma(w^w), w1
Assigning 1, 12, w, gamma( ( w^w ) ), omega_{ 1} to 'lst'.
ordCalc> a=w1.limitOrdLst(lst)
( omega_{ 1} ).limitOrd( 12 ) = omega_{ 1}[ 12]
( omega_{ 1} ).limitOrd( w ) = omega_{ 1}[ w]
( omega_{ 1} ).limitOrd( gamma( ( w^w ) ) ) = omega_{ 1}[ gamma( ( w^w ) )]
Assigning omega_{ 1}[ 12], omega_{ 1}[ w], omega_{ 1}[ gamma( ( w^w ) )] to 'a'.
ordCalc> bg = w_{w+33}
Assigning omega_{ w + 33} to 'bg'.
ordCalc> c=bg.limitOrdLst(lst)
( omega_{ w + 33} ).limitOrd( 12 ) = omega_{ w + 33}[ 12]
( omega_{ w + 33} ).limitOrd( w ) = omega_{ w + 33}[ w]
( omega_{ w + 33} ).limitOrd( gamma( ( w^w ) ) ) = omega_{ w + 33}[ gamma( ( w^w
) )]
( omega_{ w + 33} ).limitOrd( omega_{ 1} ) = omega_{ w + 33}[ omega_{ 1}]
Assigning omega_{ w + 33}[ 12], omega_{ w + 33}[ w], omega_{ w + 33}[ gamma( (
w^w ) )], omega_{ w + 33}[ omega_{ 1}] to 'c'.
```

9.11 List of descending trees

The following shows how to construct a list of descending trees.

'`desLimitOrdLst`' iterates '`limitOrdLst`' to a specified '`depth`'. The first parameter is the integer depth of iteration and the second is the list of parameters to be used. This routine first takes '`limitOrd`' of each element in the second parameter creating a list of outputs. It then takes this list and evaluates '`limitOrd`' for each of these values at each entry in the original parameter list. All of these results are combined in a new list and the process is iterated '`depth`' times. The number of results grows exponentially with '`depth`'.

```
ordCalc> lst = 1, 5, w, psi(2,3)
Assigning 1, 5, w, psi( 2, 3 ) to 'lst'.
ordCalc> bg = w_{3}
Assigning omega_{ 3} to 'bg'.
ordCalc> d= bg.desLimitOrdLst(2,lst)
( omega_{ 3} ).limitOrd( 1 ) = omega_{ 3}[ 1]
( omega_{ 3} ).limitOrd( 5 ) = omega_{ 3}[ 5]
( omega_{ 3} ).limitOrd( w ) = omega_{ 3}[ w]
( omega_{ 3} ).limitOrd( psi( 2, 3 ) ) = omega_{ 3}[ psi( 2, 3 )]
Descending to 1 for omega_{ 3}
( omega_{ 3}[ 1] ).limitOrd( 1 ) = omega_{ 2}
( omega_{ 3}[ 1] ).limitOrd( 5 ) = omega_{ 2, omega_{ 2, omega_{ 2, omega_{ 2,
omega_{ 2} + 1} + 1} + 1} + 1}
( omega_{ 3}[ 5] ).limitOrd( 1 ) = omega_{ 3}[ 4]
( omega_{ 3}[ 5] ).limitOrd( 5 ) = omega_{ 2, omega_{ 2, omega_{ 2, omega_{ 2,
omega_{ 3}[ 4] + 1} + 1} + 1} + 1}
( omega_{ 3}[ w] ).limitOrd( 1 ) = omega_{ 3}[ 1]
( omega_{ 3}[ w] ).limitOrd( 5 ) = omega_{ 3}[ 5]
( omega_{ 3}[ psi( 2, 3 )] ).limitOrd( 1 ) = omega_{ 3}[ psi( 2, 2 )]
( omega_{ 3}[ psi( 2, 3 )] ).limitOrd( 5 ) = omega_{ 3}[ epsilon( epsilon( epsilon(
epsilon( psi( 2, 2 ) + 1) + 1) + 1) + 1)]
Assigning omega_{ 3}[ 1], omega_{ 3}[ 5], omega_{ 3}[ w], omega_{ 3}[ psi( 2, 3
)], omega_{ 2}, omega_{ 2, omega_{ 2, omega_{ 2, omega_{ 2, omega_{ 2} + 1} + 1}
+ 1} + 1}, omega_{ 3}[ 4], omega_{ 2, omega_{ 2, omega_{ 2, omega_{ 2, omega_{ 3}[
4] + 1} + 1} + 1} + 1}, omega_{ 3}[ 1], omega_{ 3}[ 5], omega_{ 3}[ psi( 2, 2 )],
omega_{ 3}[ epsilon( epsilon( epsilon( epsilon( psi( 2, 2 ) + 1) + 1) + 1) + 1)]
to ‘d’.
```

References

[1] Paul Budnik. *What is and what will be: Integrating spirituality and science.* Mountain Math Software, Los Gatos, CA, 2006. 2

[2] Paul Budnik. What is Mathematics About? In Paul Ernest, Brian Greer, and Bharath Sriraman, editors, *Critical Issues in Mathematics Education*, pages 283–292. Information Age Publishing, Charlotte, North Carolina, 2009. 2

[3] Paul Budnik. A Computational Approach to the Ordinal Numbers: Documents ordCalc 0.3.2, `www.mtnmath.com/ord/ordinal.pdf`. August 2012. 2, 5

[4] Paul J. Cohen. *Set Theory and the Continuum Hypothesis.* W. A. Benjamin Inc., New York, Amsterdam, 1966. 2

[5] Jean H. Gallier. What's so Special About Kruskal's Theorem And The Ordinal Γ_0? A Survey Of Some Results In Proof Theory. *Annals of Pure and Applied Logic*, 53(2):199–260, 1991. 2

[6] Larry W. Miller. Normal functions and constructive ordinal notations. *The Journal of Symbolic Logic*, 41(2):439–459, 1976. 2

[7] Oswald Veblen. Continuous increasing functions of finite and transfinite ordinals. *Transactions of the American Mathematical Society*, 9(3):280–292, 1908. 2

Index

The defining reference for a phrase, if it exists, has the page *number* in *italics*.

The following index is semiautomated with multiple entries created for some phrases and subitems automatically detected. Hand editing would improve things, but is not always practical.

Formatted: August 2, 2012