A Maple Package for Posets

Version 2.4[*]

18 March 2009

John R. Stembridge[1]
Department of Mathematics
University of Michigan
Ann Arbor, Michigan 48109–1043 USA

email: `jrs@umich.edu`
`www.math.lsa.umich.edu/~jrs`

Contents

1. Overview

This article provides an introduction to Version 2.4 of `posets`, a package of 41 Maple programs that provide an environment for computations involving partially ordered sets and related structures. The package is particularly useful for visualizing partial orders, for isomorphism testing, and for computing various poset invariants, such as Möbius functions and order polynomials. Also included is a library containing all 19,449 posets with ≤ 8 vertices and all 7,372 lattices with ≤ 10 vertices—this is convenient for investigating questions of the form "Which posets in class X satisfy property Y?"

New in Version 2.4:

- A `strongcomps` function, for extracting (a) the strongly connected components of a directed graph, and (b) the associated acyclic digraph on those components.
- Improved implementations of the functions related to isomorphism testing and automorphisms. For example, detecting an isomorphism between two randomly labeled copies of a 7-cube (a graph with 128 vertices and a transitive automorphism group)

[*] Version 2.4 is the Aruban Flag Day Edition.

[1] This project has been supported by grants from the NSF and NSA.

is about 2.5 times faster, and finding generators for the automorphism group of the lattice of partitions of a 7-set (a poset with 877 vertices) is about 12 times faster.

For those who are behind on their upgrades, the new features in Version 2.3 include

- New functions: `chains`, `ideals`, `eulerian`, `f_poly`, `h_poly`, `flag_f`, and `flag_h`.
- The function `plot_poset` has two new options: a `levels` option that provides greater control over the layout of the Hasse diagram of a poset, and a `dot` option that writes a description of the Hasse diagram in the `dot` language. The cross-platform GraphViz package ⟨`www.graphviz.org`⟩ converts dot-descriptions of graphs into many output formats, including `gif`, `png`, and PostScript, and also provides graph-viewing tools.
- Many of the existing functions, particularly `covers`, `filter`, `lattice`, `J`, `omega`, `subposet`, `W`, and `zeta`, are now faster and more space efficient. In some cases, the improvements are dramatic. For example, computing `J(chain(6) &* chain(7))` and `W(chain(6) &* chain(7),1,t)` is now about 75–80 times faster than Version 2.2.

For a more detailed account of changes and new functionality, see §7.

2. System Requirements

First, you must have some version of Maple ⟨`www.maplesoft.com`⟩.

This version of `posets` has been developed and tested on various Linux machines running various flavors of Maple from Maple V Release 3 to Maple 11. I anticipate no problems running this on Maple 12, but have not tested it. In theory, it should work even on Maple V Release 2, and everything except `plot_poset` and the integrated help files should work on Maple V Release 1. My congratulations are offered to anyone out there who has a working copy of Maple V Release 1.

Version 2.4 is packaged in two formats: one is a Unix `tar` file (the "Unix Edition"), and the second is a collection of plain text files that comprise the Vanilla Edition. Both editions are available for download from

⟨`www.math.lsa.umich.edu/~jrs/maple.html`⟩.

The minor functional differences between these two editions are discussed in §3. Users of Maple on Unix platforms have the choice of installing either edition of `posets`. For everyone else, the Vanilla Edition is the only choice.

3. Getting Started

Special Notes for the Unix Edition. The first step is to unpack the `tar` file and follow the installation instructions in the provided `READ_ME` file. If a system administrator has done the installation for you, and you aren't already a user of one of my packages, then you will need to find out the name of the directory where the package has been installed and edit an initialization file.

For the purposes of this discussion, let's assume that the `posets` package has been installed in the directory `/usr/local/maple/packages`. You can check this by verifying that `/usr/local/maple/packages` has a subdirectory named `posets`. Next, you will need to create a file named `.mapleinit` in your home directory. Or add to it, if you already have one. In this file you should insert the following two lines:

```
HomeLib:=`/usr/local/maple/packages`:
libname:=libname, HomeLib:
```

Be careful to correctly type both of the left quote-marks (‘). Each time Maple is invoked, these statements will define where the package is located.

You are now ready to run the package. Simply enter the command `with(posets)` during a Maple session. To access documentation about an individual function, use `?posets[`*function*`]` or `?posets,`*function*. The source code for `posets` is located in

`/usr/local/maple/packages/posets/src`,

the poset library is located in

`/usr/local/maple/packages/posets/lib`,

and the raw text for the on-line help is located in

`/usr/local/maple/packages/help/posets`.

Additional files that provide applications and extensions of the package may be found in

`/usr/local/maple/packages/posets/examples`.

Special Notes for the Vanilla Edition. The Vanilla Edition consists primarily of a single text file that may be loaded into a Maple session via the `read` command. It loads all of the package functions and the entire contents of the poset library (in a compressed form) into the workspace, whereas the Unix Edition keeps clutter to a minimum by loading functions and library elements at the moment they are first used.

When the Vanilla Edition of `posets` is first loaded, the package functions may be used only via the long form

`posets[`*function*`](`*arguments*`)`.

Although `posets` is not recognized by the Maple `with` command, the Vanilla Edition provides a special command `withposets()` that enables the use of the short forms

function`(`*arguments*`)`.

Similarly, `withposets(`*arguments*`)` enables the short form for some subset of the package, modeling the expected behavior of `with(posets,`*arguments*`)`.

Documentation for the individual functions is not loaded when the package is loaded. Instead there is a separate HTML file available at

⟨`http://www.math.lsa.umich.edu/~jrs/software/posetshelp.html`⟩

that provides all of the help texts cross-referenced with hyperlinks. Similarly, the suite of examples is provided as a directory of plain text files available for download at

⟨`www.math.lsa.umich.edu/~jrs/software/posetsexamples`⟩.

The remainder of this article presents an overview of the workings of the package, explaining the basic definitions and fundamental data structures. A one-line description of the purpose of each procedure and example is provided in §6. For more detailed information about the individual commands, consult the on-line help.

4. Some Definitions

We first summarize some of the basic definitions in the theory of finite partially ordered sets and lattices. Standard references include [**1**], [**5**], and Chapter 3 of [**7**]. We refer the reader to these sources for the terminology that we have not explicitly defined here or in the on-line help.

A *partially ordered set* (or poset) is a pair $P = (X, <)$ consisting of a vertex set X, and a binary relation $<$ on X such that for all $x, y, z \in X$, we have

1. (Anti-reflexivity) $x \not< x$.
2. (Transitivity) $x < y$ and $y < z$ implies $x < z$.

The notation $x \le y$ is an abbreviation for '$x < y$ or $x = y$'.

The poset P is a *lattice* if every pair $x, y \in X$ has a least upper bound (i.e., there is an element z such that $x \le z$ and $y \le z$, and for every z' such that $x \le z'$ and $y \le z'$, we have $z \le z'$) and a greatest lower bound. In such cases, the least upper bound of x and y is also known as the *join*, and the greatest lower bound is also known as the *meet*. A lattice is *distributive* if the join and meet satisfy the same distributive laws enjoyed by union and intersection for sets.

A vertex y is said to *cover* vertex x (in P) if $x < y$ and there is no $z \in X$ such that $x < z < y$. Note that the partial order is completely determined by its covering relation via transitive closure.

It is sometimes useful to think of P as a directed graph having vertex set X and arcs $x \leftarrow y$ for every related pair $x < y$. The *Hasse diagram* of P is a planar embedding of the digraph corresponding to the covering relation of P (with edges allowed to cross each other) such that whenever $x < y$, the vertical coordinate of x is less than the vertical coordinate of y.

We define a poset P to be *ranked* if there is an integer-valued function $r : X \to \mathbf{Z}$ (the rank function) such that $r(y) - r(x) = 1$ whenever y covers x. Note that in [**7**], a poset is defined to be *graded* if all maximal chains (i.e., maximal totally ordered subsets of the vertices) have the same length. It is not hard to show that graded posets are ranked, and also that the converse is false. However, among *bounded* posets (i.e., posets that have a maximum and minimum element), the properties of being graded or ranked are equivalent.

An *order ideal* of P is a subset I of X such that $y \in I$ and $x < y$ implies $x \in I$. The set of all order ideals of P, denoted $J(P)$, is partially ordered by inclusion; this poset is a distributive lattice, and conversely, all distributive lattices arise this way.

A *linear extension* of P is a total ordering $w = (w_1, \ldots, w_n)$ of the vertex set of P with the property that $w_i < w_j$ (as elements of P) implies $i < j$ (as integers).

Although it does not seem to be a standard notion in the literature, another important concept for the `posets` package is the *filtration* of a poset. This is an ordered partition $(F_1, F_2, \ldots, F_r)$ of the vertex set X, defined so that F_1 is the set of minimal elements of X, F_2 is the set of minimal elements of $X - F_1$, F_3 is the set of minimal elements of $X - (F_1 \cup F_2)$, and so on. One may similarly define the filtration of an acyclic digraph, replacing "minimal elements" with "sinks" (vertices of outdegree 0).

Two functions in the package, `omega` and `W`, make use of the notion of a *labeling* of a poset $P = (X, <)$ as in [**6**]. By this we mean an injective map $\lambda : X \to \mathbf{Z}$ that assigns integer labels to the vertices. Two labelings of the poset P, say λ and λ', are said to be *equivalent* if

$$\lambda(x) > \lambda(y) \Leftrightarrow \lambda'(x) > \lambda'(y)$$

for all $x, y \in X$ such that y covers x. The equivalence class of a labeling λ is therefore determined by the set of pairs (x, y) such that y covers x and $\lambda(x) > \lambda(y)$. A labeling is *natural* if this set is empty; i.e., $x < y$ implies $\lambda(x) < \lambda(y)$.

A (P, λ)-*partition* is a mapping $f : X \to \{1, 2, \ldots\}$ such that for all $x < y$ in P, we have (1) $f(x) \geq f(y)$, and (2) $\lambda(x) > \lambda(y) \Rightarrow f(x) > f(y)$.

The *order polynomial* of the pair (P, λ) is the unique polynomial $\Omega(P, \lambda; t)$ with the property that for all integers $m > 0$, $\Omega(P, \lambda; m)$ equals the number of (P, λ)-partitions f such that $f(x) \leq m$ for all $x \in X$. It can be shown that the order polynomial of (P, λ) depends only on the equivalence class of λ (and on P). Note also that if λ is a natural labeling, then $\Omega(P, \lambda; m)$ equals the number of order-reversing (or equivalently, order-preserving) maps from P to an m-element chain.

Now suppose that the poset P has n vertices. Given any linear extension w of P, the *descent set* of w relative to λ is defined to be

$$D(\lambda, w) := \{1 \leq i \leq n : \lambda(w_i) > \lambda(w_{i+1})\},$$

following the convention that $\lambda(w_{n+1}) = -\infty$. In these terms, the descent set necessarily includes n, except in the degenerate case of a poset with 0 vertices.

We define the W-*polynomial* of the pair (P, λ) to be the generating series

$$W(P, \lambda; q, t) = \sum_{w \in L(P)} q^{S(D(\lambda, w))} t^{|D(\lambda, w)|},$$

where $L(P)$ is the set of linear extensions of P and $S(D(\lambda, w))$ denotes the sum of the members of $D(\lambda, w)$. Note that the W-polynomial in [**7**] is univariate, and corresponds in our notation to $q^{-n} W(q, 1)$.

It can be shown that

$$\frac{W(P, \lambda; q, t)}{(1 - qt)(1 - q^2 t) \cdots (1 - q^n t)} = \sum_f q^{S(f)} t^{\max(f)},$$

where $S(f) = \sum_{x \in X} f(x)$ and the sum ranges over all (P, λ)-partitions f. It follows that the order polynomial and W-polynomial are related as follows:

$$\frac{W(P, \lambda; 1, t)}{(1 - t)^{n+1}} = \sum_{m \geq 0} \Omega(P, \lambda; m) t^m.$$

It is worth remarking that one of the original motivations that led me to develop the `posets` package was to test the Neggers-Stanley Conjecture [**3**]; i.e., the conjecture that for every labeling λ of every poset P, all zeroes of the polynomial $W(P, \lambda; 1, t)$ are real. Using the package, we verified the conjecture for all labelings of posets with ≤ 8 vertices. Later, we developed software to search over isomorphism classes of larger posets (see the `traverse` function in the `examples` directory), and verified the conjecture for the natural labelings of the 183,231 posets with 9 vertices and the 2,567,284 posets with 10 vertices.

More recently, Petter Brändén found an unnaturally labeled counterexample to the Neggers-Stanley Conjecture with 22 vertices [**2**]. This suggested areas to narrow the search, and allowed us to find a naturally labeled counterexample with 17 vertices (disproving Neggers original conjecture) and determine that the smallest (unnatural) counterexamples have 10 vertices (see [**9**] and ⟨`www.math.lsa.umich.edu/~jrs/data/pocon/`⟩).

The package was also crucial in the development of the 3+1 Conjecture in [**8**, §5].

5. Some Data Structures

The `posets` package has several special data structures.

Posets. In the package, a poset is represented by the set of ordered vertex pairs `[x,y]` such that `y` covers `x`.[2] This is a reasonable compromise among space, speed, and convenience.

For example, the Maple expression `{[a,b],[b,c],[a,d]}` represents a poset on four vertices $\{a, b, c, d\}$ in which the covering relations are $a < b$, $b < c$, and $a < d$. Note that transitivity implies $a < c$, but the pair `[a,c]` is not listed in the expression, since it is not part of the covering relation.

One may recover all of the relations implied by transitivity via the `closure` command in the `posets` package. Conversely, given an acyclic relation that is transitive, or merely partially transitive, one may recover the covering relation via the `covers` command.

The vertices of a poset must be integers or names.

Poset Structures. There is an apparent defect in the way we have chosen to represent posets. In the above example, how did we know that the vertex set is $\{a, b, c, d\}$? What if the vertex set was intended to be $\{a, b, c, d, e\}$, with the vertex e being unrelated to any of the other vertices? Indeed, given the possibility of isolated vertices, there is no certain way to recover the vertex set of a poset from its set of covering pairs.

To cope with these situations, the `posets` package has the following more general data structure. Suppose that `P` is a Maple expression representing the set of covering pairs of a poset P, as before. A *poset structure* for P is a Maple expression sequence of one of the following three forms:

1. `P`
2. `P, X` (if `X` is the vertex set of `P`)
3. `P, n` (if `{1,2,...,n}` is the vertex set of `P`)

The first form may be used only when the poset has no isolated vertices. Thus

```
{[a,b],[b,c],[a,d]}
{[a,b],[b,c],[a,d]}, {a,b,c,d,e}
{[2,3],[2,4],[3,5],[4,5]}, 6
```

are all valid examples of poset structures.

All posets in the package must have non-empty vertex sets. Thus `{}` and `{},{}` are not valid poset structures, but `{},1` and `{},{a,b}` are valid. (Using a poset with 0 vertices should not cause any fatal errors; we simply make no promises that the output returned will conform to your expectation of the "correct" answer in such cases.)

[2] This is a slight lie, as we shall see in a moment.

Every procedure in the `posets` package that accepts a poset as part of its input will also accept any poset structure.[3] For example, consider the function `antichains`, whose purpose is to produce a list of all antichains (sets of totally unrelated vertices) in a poset. Any of the calling sequences

```
antichains(P);
antichains(P,X);
antichains(P,n);
```

may be used to produce the list of antichains in P. Of course, the first form may be used only if P has no isolated vertices.

Similarly, some of the procedures that produce posets as output, notably `chain`, `subinterval`, and the operators `&u`, `&+`, and `&*` will return poset structures of the form `P,X` or `P,n` when there are isolated vertices.

Directed graphs. Some of the procedures in the `posets` package may be applied to arbitrary directed graphs, not just the covering relations of posets. These include

```
autgroup,  canon,  connected,  dual,  isom,  rm_isom,  strongcomps.
```

Here, the function `connected` is applied to the underlying *undirected* graph. Several more procedures may applied to *acyclic* digraphs, including

```
closure,  covers,  extensions,  filter,  plot_poset,  subinterval.
```

A directed graph D is represented in the package as a set of ordered pairs. For example, the Maple expression `{[a,b],[b,c],[a,c]}` represents the (acyclic) digraph with vertex set $\{a, b, c\}$, and arcs $a \leftarrow b$, $b \leftarrow c$, $a \leftarrow c$. As with posets, the above procedures also accept what could be called "directed graph structures" of the form `D,X` or `D,n`, specifying that the vertex set of `D` is `X` or `{1,2,...,n}`, respectively.

It is useful to be aware that for a general digraph D (possibly with cycles), the function `filter` returns the filtration of the (acyclic) induced subgraph formed by deleting all vertices of D that can reach a cycle by following a directed path. Thus the expression

```
nops(map(op,filter(D,X))) = nops(X)
```

provides a simple test for whether digraph `D` is acyclic.

Labelings of posets. (This is relevant only for the functions `omega` and `W`.) In the package, one specifies a labeling λ for a poset `P` (recall the discussion in §4) by means of the subset `S` of `P` consisting of those covering pairs `[x,y]` such that $\lambda(\mathtt{x}) > \lambda(\mathtt{y})$; recall that this set determines the labeling up to equivalence. In particular, to compute the order polynomial or W-polynomial, one uses the calling sequences

```
omega(P,t,S);
W(P,q,t,S);
```

If the set `S` is omitted, then the default value is `S={}`, which corresponds to a natural labeling. If there is no labeling of `P` whose corresponding subset is `S`, then an error will be signaled when computing `W` or `omega`.

[3] A quasi-exception to this rule is the procedure `rm_isom`, which accepts lists or sets of covering relations of posets, but not individual posets.

6. Short Synopses of the Procedures

Here is a brief indication of the purpose of each procedure in the `posets` package. For the full details, including syntax, definitions and examples, consult the on-line help via the commands `?posets,`*function* or `?posets[`*function*`]`, or use the HTML document provided in the Vanilla Edition.

`antichains`	list/count antichains in a partially ordered set
`atomic`	test whether a lattice is atomic
`autgroup`	automorphism group of a poset (or directed graph)
`canon`	canonically relabel a poset (or directed graph)
`chain`	total order of specified length
`chains`	list/count chains in a partially ordered set
`char_poly`	characteristic polynomial of a (graded) poset
`closure`	transitive closure of an acyclic digraph
`connected`	connected components and poset connectivity test
`covers`	covering relation of an acyclic digraph
`distributive`	test distributivity of lattices
`dual`	dual of a poset (or directed graph)
`eulerian`	test whether a poset is Eulerian
`extensions`	linear extensions of a poset (or acyclic digraph)
`filter`	filtration of a poset (or directed graph)
`flag_f`	flag f-polynomial of a ranked, bounded poset
`flag_h`	flag h-polynomial of a ranked, bounded poset
`f_poly`	f-polynomial of a bounded poset
`h_poly`	h-polynomial of a bounded poset
`ideals`	list/count order ideals in a partially ordered set
`isom`	test posets (or directed graphs) for isomorphism
`J`	lattice of order ideals of a poset
`lattice`	test whether a poset is a lattice or semi-lattice
`Lattices`	list nonisomorphic lattices
`meet`	compute meets in lattices, maximal lower bounds in posets
`mobius`	Möbius function of a poset
`modular`	test modularity and semi-modularity of lattices
`omega`	order polynomial of a poset
`plot_poset`	plot posets (or acyclic directed graphs)
`Posets`	list nonisomorphic posets
`rand_poset`	random poset generator
`ranked`	test whether a poset is ranked
`rm_isom`	remove isomorphic copies from a list of posets
`strongcomps`	strongly connected components of a digraph
`subinterval`	extract a subinterval from a poset
`subposet`	extract an induced subposet from a poset
`W`	W-polynomial of a poset
`zeta`	zeta polynomial of a poset

Also, when the package is loaded, the following neutral operators are defined:

`&u`	disjoint union of posets	(for help, see `posets[disj_union]`)
`&+`	ordinal sum of posets	(for help, see `posets[ord_sum]`)
`&*`	direct product of posets	(for help, see `posets[product]`)

These are binary in-fix operators—thus `P &+ Q` will return the ordinal sum of P and Q, and `P &* Q &* R` will return the direct product of P, Q and R.

Here is a short summary of the contents of the `examples` subdirectory. These files must be explicitly loaded by the user and contain both documentation and source code.

`bigraphs`	list non-isomorphic (simple) bipartite graphs
`bruhat_order`	Bruhat ordering of a finite Coxeter group*
`divisor`	lattice of divisors of an integer
`dominance`	dominance ordering of partitions of an integer*
`graphs`	list non-isomorphic (simple, undirected) graphs
`par_lattice`	lattice of partitions of a set
`traverse`	iterate over isomorphism classes of posets on n points
`weak_order`	weak ordering (and Cayley graph) of a finite Coxeter group*
`young_diag`	Young diagram of a (skew) partition
`young_lattice`	Young's lattice*

The examples `bruhat_order` and `weak_order` require the `coxeter` package. Likewise, the examples `dominance` and `young_lattice` require the `SF` package. Both packages are available at ⟨`www.math.lsa.umich.edu/~jrs/maple.html`⟩.

7. Changelog

Version 2.4. The changes are:

- A new `strongcomps` function, for extracting (a) the strongly connected components of a directed graph, and (b) the associated acyclic digraph on those components.
- Improved implementations of the functions related to isomorphism testing and automorphisms. For example, detecting an isomorphism between two randomly labeled copies of a 7-cube (a graph with 128 vertices and a transitive automorphism group) is about 2.5 times faster, and finding generators for the automorphism group of the lattice of partitions of a 7-set (a poset with 877 vertices) is about 12 times faster.
- Random number generation has changed significantly in Maple 10. In particular, the procedure `rand` no longer produces the same stream of pseudo-random numbers. For this reason, we have added an option to pass any random number generating procedure as an argument to `rand_poset`, overriding the default use of `rand`.

Version 2.3. The significant changes and additions are:

- Seven new functions have been added: `chains`, `ideals`, `eulerian`, `f_poly`, `h_poly`, `flag_f`, and `flag_h`.
- The function `antichains` now (by default) returns a lexicographically sorted list of all antichains in a poset, and has options for specifying a range of antichain sizes, as well as for efficient computation of the generating series for antichains. (The new `chains` and `ideals` functions have similar features.) The old version returned a *set* of antichains, had fewer options, and these options used an incompatible syntax.

- The numbering of vertices by the `J`-operator is now linked to the lexicographic ordering used by the `ideals` function. Thus, the i-th vertex of `J(P,X)` is the i-th order ideal listed by `ideals(P,X)`.
- The `ranked` option in `plot_poset` has been replaced by the more flexible `levels`.
- The function `plot_poset` now has the option of writing a description of the Hasse diagram of a poset (or acyclic digraph) in the `dot` language, for external processing by the GraphViz package (see ⟨`www.graphviz.org`⟩).
- Many of the existing functions, particularly `covers`, `filter`, `lattice`, `J`, `omega`, `subposet`, `W`, and `zeta`, are now faster and more space efficient.
- Fast counting of linear extensions of a poset is now possible by evaluating the W-polynomial at $(q,t)=(1,1)$. For example, it takes about 0.2 GHz-sec[4] to determine that the product of two chains of length 7 has 475073684264389879228560 linear extensions, using the command `W(chain(7) &* chain(7),1,1)`.
- The option of installing a small version of the library has been deleted.

Version 2.2. In addition to the debut of the Vanilla Edition, the significant changes and new features in this version of the `posets` package are:

- A new function `canon` has been introduced for generating canonical relabelings of posets. Two posets (or directed graphs) (P,X) and (Q,Y) are isomorphic if and only if `canon(P,X)=canon(Q,Y)`. (However, isomorphism testing is faster with `isom`.)
- The function `automorphisms` has been replaced with the more powerful `autgroup`.
- The old isomorphism tester would provide a complexity estimate when called with arguments specifying a single poset structure. The new `isom` is now so much faster and effective that this functionality has been dropped.
- Three new examples, `bigraphs`, `graphs` and `traverse` have been added.
- The order in which posets are listed in the library has changed, and the particular vertex labelings of these posets have changed as well.

Version 2.1. Aside from improvements in the documentation, the main changes in Version 2.1 of the `posets` package are the following:

- The procedure `plot_poset` now passes on any unrecognized optional arguments to Maple's `plots[display]` command. This allows the user to take advantage of the new options available for Maple 2D graphics that were added in Releases 3 and 4.
- The operators `&+` and `&*` for ordinal sum and direct product have been rewritten so that they do not conflict with Maple's internal use of these operators. For example, after loading the `posets` package, one may still use `evalm(A &* B)` to multiply matrices `A` and `B`.
- The examples `bruhat_order` and `weak_order` have been rewritten to take advantage of new features in the latest version of the `coxeter` package.

Version 2.0. The main changes affecting users upgrading to Version 2.0 are:

- There are 13 new procedures: `atomic`, `automorphisms`, `connected`, `distributive`, `&u` (disjoint union), `lattice`, `meet`, `modular`, `plot_poset`, `rand_poset`, `ranked`, `subinterval`, and `subposet`.

[4] A GHz-sec means one second of time for a machine with a 1 GHz Pentium III CPU.

- Many of the existing procedures are significantly faster and more space-efficient. Several, including `antichains`, `isom`, `Lattices`, `omega`, `Posets`, and `W`, have new features.
- Every procedure is consistent in its use of poset structures. Also, a poset structure is now an expression sequence, not a list.
- Filtering digraphs with cycles will no longer cause Maple to enter an infinite loop.
- The library now has the 16,999 posets with 8 vertices and the 5,994 lattices with 10 vertices. The storage method (and ordering) of posets within the library is new.
- The `examples` directory has been completely redesigned.
- The procedures `permfit` and `invariants` have been renamed `` `posets/permfit` `` and `` `posets/invariants` `` and are now internal to the package.
- The procedure `height` has been removed from the package. A functional replacement is `proc() nops(filter(args)) end`.
- The `W` function now computes a two-variable polynomial attached to a (labeled) poset. The old `W` function is recovered by taking $q = 1$.
- `rm_isom` now returns a sublist (or subset) of the input.
- `filter` no longer accepts the optional flag `'ranked'`. This option is subsumed by the new `ranked` function.

8. Miscellany

Isomorphism testing. The algorithms we use for isomorphism and the associated automorphism and canonical labeling problems are closely related to the ones used by Brendan McKay for Nauty (see [**4**] and ⟨`cs.anu.edu.au:80/people/bdm/nauty/`⟩).

The first point is that one should replace isomorphism of directed graphs with a slightly more general problem: isomorphism of vertex-colored digraphs. Thus we consider isomorphisms between triples (D, X, π), where X is a vertex set, D is a subset of $X \times X$, and π is a mapping that assigns a color $\pi(x)$ to each vertex $x \in X$. Of course, isomorphisms are required to preserve colors. If we are interested only in normal (uncolored) digraph isomorphism, we assign every vertex the same color.

The second point is that we need a reasonably fast operation f that takes a given colored digraph (D, X, π), and produces a new coloring of (D, X), say σ, with the property that (D, X, π) and (D', X', π') are isomorphic if and only if (D, X, σ) and (D', X', σ') are isomorphic, where $\sigma = f(D, X, \pi)$ and $\sigma' = f(D', X', \pi')$. Furthermore, the operation f should always produce a coloring that is at least as discriminating as the previous one—if two vertices have different colors in π, then they should have different colors in $f(D, X, \pi)$. We call any operation f with these properties a *discriminant.*

Leaving aside the details of how we might choose our discriminant, the algorithm for isomorphism testing functions as follows. Given two colored digraphs (D, X, π) and (D', X', π'), we first check to see that both graphs have the same number of vertices of each color, and then check to see whether a randomly chosen color-preserving bijection between X and X' is an isomorphism. If all vertices have different colors, then there is only one possible choice for the isomorphism and the algorithm is finished. If not, then we use the discriminant f to recolor both graphs. If the new colorings distinguish more vertices than the old ones, then progress has been made, and we recursively apply the same algorithm. Otherwise, if no progress has occurred, then we fix a vertex $x \in X$ from the smallest color class with more than one vertex, and for each $x' \in X'$ of the same color, we test whether there is a colored digraph isomorphism that maps $x \to x'$. If no such x' has this property, then the original graphs are not isomorphic.

By inventing a new color for x and x', one sees that testing for an isomorphism that maps $x \to x'$ is another (recursive) instance of colored digraph isomorphism.

One possibility for the discriminant is trivial: $f(D, X, \pi) = \pi$. As you would expect, this functions abysmally; it yields an algorithm equivalent to exhaustive search. At the other extreme, the best discriminant we could hope for would be one that assigns the same color to two vertices if and only if they belong to the same orbit under the automorphism group of (D, X, π). If this could be computed in polynomial time, we would have a polynomial time algorithm for graph isomorphism: once we reached the last stage of the algorithm, the first vertex x' we try to match with the chosen x must either lead to an isomorphism, or the graphs cannot be isomorphic.

In practice, a discriminant that comes close to perfect separation of the vertex orbits of colored digraphs will function reasonably well. The discriminant used in the `posets` package (we call it the "fast discriminant") is computed as follows. Given (D, X, π), we first compute a weaker discriminant g whose colors are vectors: for each $x \in X$, the color of x in $\tau = g(D, X, \pi)$ is defined to be the vector $(\pi(x), a_1, b_1, a_2, b_2, \ldots)$, where a_i (respectively, b_i) denotes the number of edges $(x, y) \in D$ (respectively $(y, x) \in D$) in which y has the i-th π-color. The fast discriminant f is computed by iterating g until it stabilizes (i.e., fails to distinguish any additional vertices).

To give you some indication of how well the fast discriminant performs, it takes `isom` about 2 GHz-sec to find an isomorphism between two randomly labeled copies of the lattice of partitions of a 6-set, a poset with 203 vertices and an automorphism group isomorphic to S_6 (see `par_lattice` in the `examples` directory).

Using the library. The library is accessed via the functions `Posets` and `Lattices`. The list of posets on 8 vertices (and to a lesser extent, the list of lattices on 10 vertices) is large enough that using it incautiously may cause Maple to allocate large amounts of memory. For example, even an operation as simple as `map(nops,Posets(8))` requires the allocation of about 4MB of memory in Maple 9. Also, one should avoid constructing `do`-loops involving lists as large as `Posets(8)`. For example, a computation of the form

```
keepers:=NULL;
for P in list_of_posets do
  if is_interesting(P) then keepers:=keepers,P fi
od:
keepers:=[keepers]:
```

has a running time that is quadratic in the number of items in `list_of_posets`, and is virtually unusable for lists whose sizes are in the range of `Posets(8)`. On the other hand, the above computation could be rewritten using `map` or `seq` so that the running time is linear. To facilitate such practices, the `Posets` and `Lattices` commands optionally accept a Boolean procedure for selecting sublists from the library satisfying a given property. Thus an efficient way to select the "interesting" posets on 8 points would be

```
keepers:=Posets(8,is_interesting):
```

To conduct space-efficient searches of even larger classes of posets, see `traverse` in the `examples` directory.

Plotting posets. After some experimentation, I've settled on `red` and `black` as the best defaults for the edge and vertex colors used by `plot_poset`.[5] Your mileage may vary, so the default color choices can be changed, either during a Maple session, or permanently. They can also be passed as arguments to `plot_poset`. To change the default colors during a Maple session, use (for example) the assignments

```
‘posets/default‘[ecolor]:=magenta;
‘posets/default‘[vcolor]:=yellow;
```

after loading the `posets` package via the `with` command. To change the default colors permanently, edit the file `src/posets` and then recompile it.

If you have the GraphViz package and suitable viewing software installed on the host machine, it is possible to process dot plots from within a Maple session using the `system` command. For example, if the host OS is some form of Unix with X11 graphics, and the ImageMagick viewer `display` is available, then the Maple commands

```
plot_poset(P, labels, dot=‘/tmp/myplot.dot‘);
system(‘dot -Tps /tmp/myplot.dot | display - &‘);
```

would generate a temporary PostScript file containing the output of the dotplot of `P` and display the image at the terminal.

The dangers of `&+` *and* `&*`. In Maple V Releases 3, 4 and 5, the neutral operators `&+` and `&*` are protected. There are a few commands in Release 3 (e.g., `sqrt`), and a few more in Release 4 and 5 (e.g., `solve`) that make undocumented use of these operators as inert place holders. When the `posets` package is loaded, it "unprotects" these operators, and assigns to them the procedures for ordinal sum and direct product. Having these assignments in place does not break `sqrt` or `solve`, since the procedures are designed to return unevaluated when they are called with anything other than a sequence of poset structures. This allows the continued use of `&+` and `&*` as inert place holders.

But there may be further undocumented uses of `&+` and `&*` that I am unaware of. If you are paranoid, you can choose alternative names such as `&s` and `&p` for ordinal sum and direct product and restore `&+` and `&*` to their original state:

```
with(posets);
‘&s‘:=op(‘&+‘); ‘&p‘:=op(‘&*‘);
unassign(‘&+‘,‘&*‘);
```

If you are *really* paranoid, you can make these changes permanent in the source code and recompile it. I have not made these changes part of the package, since this would have broken upward compatibility.

Bugs. Please send reports of reproducible bugs to my email address on page 1. Be sure to include an example, preferably as small as possible. Note that the package at present does only a small amount of testing for valid input.

Compensation. If the `posets` package proves to be useful in your research, I would appreciate an acknowledgment of it in publications derived from that research.

[5] In Maple V R2 and R3, plots have a black background, so `white` is used as the default vertex color. However, beware that `white` is almost certainly not the vertex color you would want for a dot plot.

Mailing list. I maintain a low-traffic, private mailing list for users of `posets` and my other packages `SF`, `coxeter`, and `weyl`. If you would like to be kept informed of new versions, new packages, or (gasp) bugs, please send me an email message.

9. Copyleft

10. References

[**1**] G. Birkhoff, *Lattice Theory,* 3rd ed., Amer. Math. Soc., Providence, 1967.

[**2**] P. Brändén, Counterexamples to the Neggers-Stanley conjecture, *Electron. Res. Announc. Amer. Math. Soc.* **10** (2004), 155–158.

[**3**] F. Brenti, Unimodal, log-concave and Polya frequency sequences in combinatorics, *Mem. Amer. Math. Soc.* (1989), no. 413.

[**4**] B. D. McKay, Practical graph isomorphism, *Congr. Numer.* **30** (1981), 45–87.

[**5**] I. Rival, *Ordered Sets,* Reidel, Dordrecht-Boston, 1982.

[**6**] R. P. Stanley, Ordered structures and partitions, *Mem. Amer. Math. Soc.* (1972), no. 119.

[**7**] R. P. Stanley, *Enumerative Combinatorics, Vol. I,* Wadsworth & Brooks/Cole, Monterey, CA, 1986.

[**8**] R. P. Stanley and J. R. Stembridge, On immanants of Jacobi-Trudi matrices and permutations with restricted position, *J. Combin. Theory Ser. A* **62** (1993), 261–279.

[**9**] J. R. Stembridge, Counterexamples to the poset conjectures of Neggers, Stanley, and Stembridge, *Trans. Amer. Math. Soc.* **359** (2007), 1115–1128.