# Lazy Completion of a Partial Order to the Smallest Lattice

Karell Bertet[1], Michel Morvan[1], Lhouari Nourine[2]

[1] LITP/IBP - Université Denis Diderot
Paris 7 Case 7014 2, place Jussieu 75256 Paris Cedex 05 France.
email: (bertet,morvan)@litp.ibp.fr.
[2] Département d'Informatique Fondamentale
LIRMM, Université Montpellier II — CNRS UMR C09928
161 rue Ada, 34392 Montpellier Cedex 5, France.
email: `nourine@lirmm.fr`.

**Abstract.** Lattice structures are often used in knowledge processing, but starting from a partial order, completion into a lattice poses efficiency problems. We use some recent results on lattice theory to propose an on-line algorithm for efficiently computing of the smallest lattice containing a given partial order, called the Dedekind-MacNeille completion. By on-line we mean that the elements are added if necessary when we compute the greatest lower bound or the least upper bound of two elements that do not already exist. This result can be used in knowledge processing for maintaining the hierarchy of types in an on-line fashion.

**Keywords:** algorithm, partial order, lattice, Dedekind-MacNeille completion, encoding.

## 1 Introduction and Motivations

Lattices are an important class of partial orders because of their structural properties and their interest in many areas as knowledges processing with the hierarchy of types [4,6,1,13,7]. The problem of minimally extending a partial order to the smallest lattice appears in many applications.

Such an application is the hierarchy of types which is a directed acyclic graph where interpretation depends on applications (inherit, subsume, unification,...). Maintaining the graph of types is crucial in knowledge processing. This graph is usually a lattice, but it is realized dynamically, ie. while processing new knowledge. Most of the time, the application supposes that only a partial graph is known and that this graph is completed when necessary by adding a new type. That is, we are given some types and we want to compute the generalization or the specialization. But computing the set of all general types is not always practical, since some types may be not not significant for this application.

Formally, given a partial order $P = (X, E)$, we have to compute the Dedekind-MacNeille completion of $P$, denoted by $DM(P)$, which is the smallest lattice which embeds $P$ as a suborder. For more details on Dedekind-MacNeille completion see Bouchet [3] and Davey and Priestley [5]. Figure 1 shows an example of such completion. From Bouchet and Wille [3,13]we know that DM(P) is isomorphic to the lattice of maximal antichains of the bipartite order $(X, X, \not\leq)$, and also is isomorphic to the Galois lattice of $(X, X, \leq)$. So a way to compute $DM(P)$ is simply to use algorithms for computing the lattice of maximal antichains or the Galois lattice of a given partial order. There are two efficient algorithms to do this [2,9].

In this paper, we present an algorithm for on-line computing $DM(P)$, namely the lazy Dedekind-MacNeille completion: We dynamically compute $DM(P)$ by adding only elements when we need them as the lower or upper bound of two given elements. In Section 3 we show the main constraint to respect in order to insure a correct minimal construction. To do so, we propose an encoding of $P$ which gives a way which adds such an element in an efficient way based on a nice result [1,11] in lattice theory. Finally, we present an algorithm which adds such an element with the use of a tree.

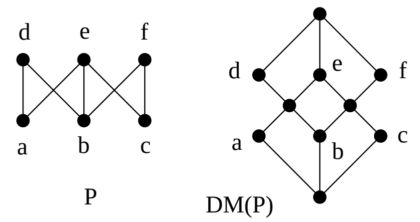



**Fig. 1.** Dedekind-MacNeille completion of a poset

## 2 Definitions and Notations

A *partial order* $P = (X, \leq_P)$ is a reflexive, antisymmetric and transitive binary relation on a set $X$. Two distinct elements $x$ and $y$ are said to be *comparable* if $x \leq_P y$ or $y \leq_P x$. Otherwise, they are said *incomparable*, denoted by $x||_P y$. $P$ is said a bipartite order when $X = X_1 \cup X_2$ with $X_1 \cap X_2 = \emptyset$ and $x \leq_P y$ implies $x \in X_1$ and $y \in X_2$. A bipartite order is often denoted by $P = (X_1, X_2, \leq_P)$.

We define the following sets for an element $x$ of $X$ : $Pred(x) = \{y \in X | y <_P x\}$ the set of *predecessors* of $x$, $Succ(x) = \{y \in X | x <_P y\}$ the set of *successors* of $x$.

A partial order $Q = (X', \leq_Q)$ is a *suborder* of $P$ iff $X' \subseteq X$ and the relation in $Q$ is the same as the relation in $P$ on the elements of $X'$.

An element $z \in X$ is an *upper bound* of $x, y \in X$ if $x \leq_P z$ and $y \leq_P z$. Let $z$ be an upper bound of $x$ and $y$, $z$ is called the least upper bound or the *join* of $x$ and $y$ if $z \leq_P t$, for all upper bounds $t$ of $x$ and $y$. The *greatest lower bound*, or *meet* is defined dually. We denote by $LUB(x,y)$ (resp $GLB(x,y)$) the least upper bound (resp the greatest lower bound) of $x$ and $y$.

A non-empty partial order $P$ is called a *lattice* if $LUB(x,y)$ and $GLB(x,y)$ exist for all $x,y \in X$. It is clear that a finite lattice has one minimal element and one maximal element denoted respectively by $\top$ and $\perp$. Since we are dealing with algorithms the lattices we consider are supposed to be finite.

Let $L = (X, \leq_L)$ be a lattice. An element $x \in X$ is said to be *join-irreducible* if $x \neq \perp$ and if $x = LUB(y,z)$ implies $x = y$ or $x = z$ for all $y,z \in X$. *Meet-irreducible* elements are defined dually. We denote the set of join-irreducible elements of $L$ by $J(L)$ and the set of meet-irreducible elements by $M(L)$. We denote by $2^X$ the set of all subsets of $X$.

**Definition 1.** [10] Let $P = (X, \leq_P)$ be a partial order. The Dedekind-MacNeille completion of $P$, denoted by $DM(P)$, is the smallest lattice containing $P$ as a suborder.

## 3 Dedekind-MacNeille Completion of a Partial Order

Since $P$ is a suborder of $DM(P)$, we have to add elements in $P$ in order to compute $DM(P)$. A naive algorithm consists in adding an element in $P$ when the $GLB$ of two elements $x$ and $y$ does not exist in $P$, with $x$ and $y$ as successors and $Pred(x) \cap Pred(y)$ as predecessors . But this algorithm does not compute the Dedekind-MacNeille completion as the following example illustrates it.

*Example 2.* Let $P$ be the partial order in Figure 2(a). Suppose that the application asks for the $GLB$ of $e$ and $f$. The naive algorithm will add an element below $e$ and $f$ as in (b) because $GLB(e,f)$ does not already exist. Likewise, if the application asks for the $GLB$ of $f$ and $g$, an element will be added below $f$ and $g$ as in (c). In this way, these two elements added are not in the Dedekind-MacNeille completion of $P$ shown in (d). Clearly this algorithm cannot be used.

Let $L$ be a lattice and $\xi(L)$ the set of all partial orders having $L$ as Dedekind-MacNeille completion. We know that the partial order induced by $J(L) \cup M(L)$ is a suborder of each order of $\xi(L)$ [5]. Consider the set of elements of $L$ which are not irreducible elements. Each subset of this set corresponds to a set of elements that we have to add to $J(L) \cup M(L)$ to obtain a partial order of $\xi(L)$. Since all the subsets of this set ordered by inclusion forms a boolean lattice, we obtain an isomorphic boolean lattice with the set $\xi(L)$ ordered by suborder (ie. $P \leq_\xi P'$ iff $P$ is a suborder of $P'$). Clearly, the $\top$ element of $\xi(L)$ is $L$ and the $\perp$ is the partial order induced by $J(L) \cup M(L)$. Figure 3 shows the set $\xi(L)$ for a lattice $L$.

In $\xi(DM(P))$, any path from a partial order $P$ to $\top$ of $DM(P)$ corresponds to a way of incrementally computing $DM(P)$.

The choice of such a way can be taken relatively to the application, for example adding elements from $\perp$ to $\top$ one by level at each step. In the following, we give an algorithm that follows a path according to questions asked by the application. The elements are added if necessary when we compute a $GLB$ or a $LUB$ that does not already exist.

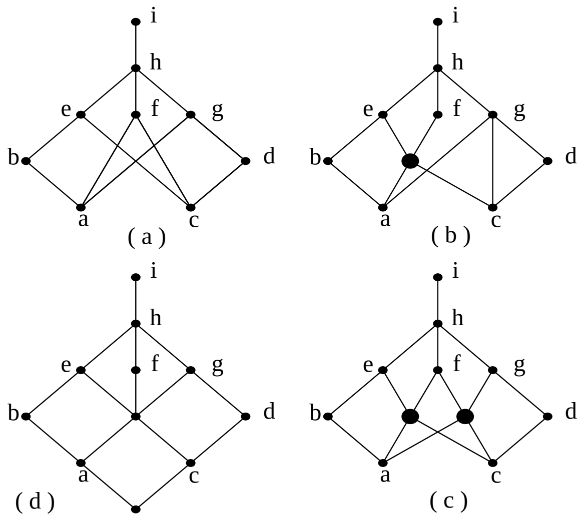



**Fig. 2.** Creation of dummy vertices

## 4 Encoding Preserving LUB and GLB

We now recall an encoding of a partial order by sets, which allows us to avoid the creation of extra elements as shown in Figure 2. This encoding is based on a result of Markowsky [11] on lattices, which appears implicitly in Aït-Kaci *et al*[1] paper on lattices encoding.

**Definition 3.** Let $T$ and $L$ be lattices. Let $x$ and $y$ be two elements of $T$.
A function $\varphi : T \to L$ is said to be:

1. **Meet-preserving** if $\varphi(GLB(x, y)) = GLB(\varphi(x), \varphi(y))$.
2. **Join-preserving** if $\varphi(LUB(x, y)) = LUB(\varphi(x), \varphi(y))$.

**Theorem 4.** *[11] Let $L = (X, \leq_L)$ be a lattice. Then the mapping*

1. *$\varphi : X \to 2^{J(L)}$ with $\varphi(x) = \{j \in J(L)$ such that $j \leq_L x\}$ is meet-preserving, ie. $\varphi(GLB(x, y)) = \varphi(x) \cap \varphi(y)$.*
2. *$\phi : X \to 2^{M(L)}$ with $\phi(x) = \{m \in M(L)$ such that $m \geq_L x\}$ is join-preserving, ie. $\phi(LUB(x, y)) = \phi(x) \cap \phi(y)$.*

*These two encodings are optimal (ie. the sizes of $\varphi$ and $\phi$ are minimal).*

Notice that for a distributive lattice $\varphi$ and $\phi$ are both join-preserving and meet-preserving[11].

**Lemma 5.** *Let $L = (X, \leq_L)$ be a lattice. Then any element of $L$ but the $\perp$ (resp. $\top$), is the $LUB$ (resp.$GLB$) of join-irreducible (resp. meet-irreducible) elements.*

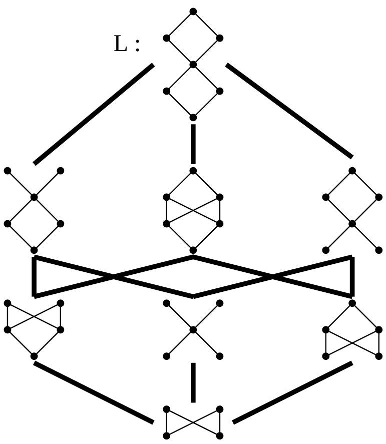



**Fig. 3.** Set $\xi(L)$ for a lattice $L$

In the following, we show how to compute the join-irreducible and meet-irreducible elements of $DM(P)$ denoted by $J(P)$ and $M(P)$ from the partial order $P$. This can be done without computing $DM(P)$. For that, we need to introduce the following notions:

Let $P = (X, \leq_P)$ be a partial order. We compute the following bipartite order $B = (X, X', \leq_B)$, with $X'$ be a copy of $X$ and $x \leq_B y$ iff $x$ is in $X$, $y$ is in $X'$ and $x \not\leq_P y$.

**Definition 6.** [12] A bipartite order $B = (X, X', \leq_P)$ is said to be reduced if:

$C_0$ : For all $x \in X$, for all $y \in X'$, we have $Succ_B(x) \neq \emptyset$, and $Pred_B(y) \neq \emptyset$
$C_1$ : For all $x \in X$, for all $x_1, \ldots, x_k \in X \backslash \{x\}$, we have $Succ_B(x) \neq \bigcup_{i=1}^{k} Succ_B(x_i)$
$C_2$ : For all $y \in X'$, for all $y_1, \ldots, y_k \in X' \backslash \{y\}$, we have $Pred_B(y) \neq \bigcup_{i=1}^{k} Pred_B(y_i)$

Condition $C_1$ is not respected in the following cases:

a) There exist $x$ and $x'$ in $X$ such that $x \neq x'$ and $Succ_B(x) = Succ_B(x')$
b) There exist $x$ in $X$ and $x_1, \ldots, x_k$ in $X \backslash \{x\}$ with $k > 1$ such that $Succ_B(x) = \bigcup_{i=1}^{k} Succ_B(x_i)$

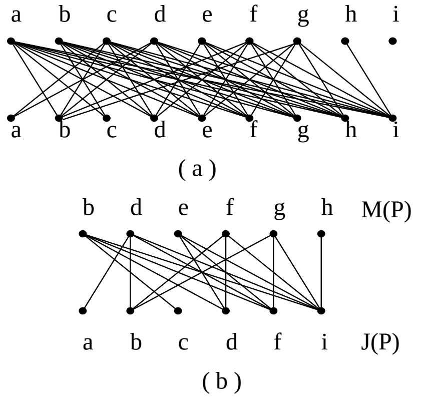
The symmetrical remark can be made for $C_2$:

a') There exist $y$ and $y'$ in $X'$ such that $y \neq y'$ and $Pred_B(y) = Pred_B(y')$
b') There exist $y$ in $X'$ and $y_1, \ldots, y_k$ in $X' \backslash \{y\}$ with $k > 1$ such that $Pred_B(y) = \bigcup_{i=1}^{k} Pred_B(y_i)$

So we can transform a bipartite order $B = (X, X', \leq_B)$ in a reduced bipartite order $Bip(P) = (J, M, \leq_B)$ by deleting singletons ($C_0$), deleting one of the two elements $x$ or $x'$ (arbitrarily) in case $a$), deleting $x$ in case $b$), deleting one of the two elements $y$ or $y'$ (arbitrarily) in case $a'$), deleting $y$ in case $b'$).

It can be shown that $J$ (resp. $M$) is the set of join-irreducible (resp. meet-irreducible) elements of $DM(P)$.

*Example 7.* Figure 4 illustrates the encodings for the partial order of Figure 2: $(a)$ the bipartite order $(X, X', \not\leq_P)$, $(b)$ its reduction $Bip(P) = (J, M, \not\leq_P)$ where $a$ is deleted because $Pred(a) = Pred(f) \cup Pred(b)$ and $i$ is deleted because $Pred(i) = \emptyset$, $(c)$ the encodings $\varphi$ and $\phi$ of the elements in $P$.




| | $\varphi$ | $\phi$ |
|---|---|---|
| a | a | befgh |
| b | ab | beh |
| c | c | defgh |
| d | cd | dgh |
| e | abc | eh |
| f | acf | fh |
| g | acd | gh |
| h | abcdf | h |
| i | abcdfi | $\emptyset$ |

(c)

**Fig. 4.** Encodings $\varphi$ and $\phi$

Now we can apply Theorem 4 to have the join-preserving and meet-preserving encoding for $DM(P)$ using $J(P)$ and $M(P)$. Since $P$ is a suborder of $DM(P)$ then we encode only elements for $P$ and the others can be obtained using $GLB$ or $LUB$ asked by the application. The following corollary is a direct consequence of Theorem 1.

**Corollary 8.** *The encodings $\varphi$ and $\phi$ computed by Algorithm 1 are respectively meet-preserving and join-preserving.*

We are now going to show how to compute the code of the $GLB$ of two given elements of $P$ and then how to decode this information to obtain the encoded element(ie. to describe $\varphi^{-1}$). We discuss only $GLB$, for $LUB$ can be obtained dually.

**Lemma 9.** *Let $x$ and $y$ be two elements of a partial order $P$.*
*Then $\varphi(GLB(x, y)) = \varphi(x) \cap \varphi(y)$, furthermore $GLB(x, y) = \varphi^{-1}(\varphi(x) \cap \varphi(y))$.*

The complexity of the algorithm computing $GLB(x, y)$ depends on the data structure used for storing the code $\varphi$. To have good complexity, we use the tree data structure introduced in [8] for an optimal representation of a distributive lattice. This tree allows us to have a best time complexity to compute $\varphi^{-1}$.

**Algorithm 1:** Join-preserving and meet-preserving encodings

**Data** : A partial order $P = (X, \leq_P)$

**Result** : The meet-preserving encoding $\varphi : P \to 2^{J(P)}$. the join-preserving encoding $\phi : P \to 2^{M(P)}$.

**begin**

Compute $B = (X, X', \not\leq_P)$;

Compute the reduction $Bip(P) = (J, M, \leq_B)$ of $B$.

{Encoding all elements in $P$}

**for** *all* $x \in X$ **do**

$\varphi(x) = \bigcup_{j \in J,\ j \leq_P x} \{j\}$;

$\phi(x) = \bigcup_{m \in M,\ m \geq_P x} \{m\}$;

**end**

## 5 Algorithm

In this section, we give an algorithm based on a tree data structure to compute $GLB(x, y)$. To do so, consider a partial order $P = (X, \leq_P)$ and the encoding $\varphi$ of $P$ computed previously. For each element $x \in P$, we suppose that $\varphi(x)$ is sorted according to a topological sorting $\sigma$ of $J(P)$.

Let us define the tree data structure, denoted by $T_\varphi$ for the encoding $\varphi$.

- Let $Root_\varphi$ be the root of $T_\varphi$ (this corresponds to the maximal element in $DM(P)$).
- Each node of the tree corresponds to an element of $DM(P)$.
- Every edge of the tree is labeled by elements of $J(P)$, such that the union of labels from any node $x$ to $Root_\varphi$ corresponds $\varphi(x)$ sorted according to $\sigma$.
- The sons of each internal node are sorted according to $\sigma^{-1}$

Note that for a distributive lattice, every edge is labeled by one element of $J(P)$. Most operations on this tree are described in [8]. We use this tree to store the code $\varphi$: for a code $\varphi(x)$ of an element $x$, we have to search the tree in a recursive way to verify if $x$ is in the tree. If not, we can easily add it.

*Example 10.* Figure 5 shows $T_\varphi$ for the encoding of the partial order of Figure 2, with $\sigma_\varphi = a < b < c < d < f < i$.

Let us now describe the computing of $GLB(x, y)$ using the tree.

*Example 11.* Consider the partial order in Figure 2. If the application asks for $GLB(e, f)$, Algorithm 2 computes $\varphi(GLB(e, f)) = \varphi(e) \cap \varphi(f) = ac$ which is added in the tree of the encoding $\varphi$ as Figure 6(a). Then, a new element appears in $P$ (not explicitelly added in $P$ but represented by its code) as in Figure 6(b). Now, if the application asks for $GLB(f, g)$, Algorithm 2 computes $\varphi(GLB(f, g)) = \varphi(f) \cap \varphi(g) = ac$ which already exists in $T\varphi$.

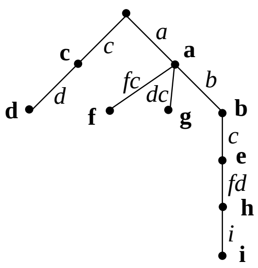



**Fig. 5.** Tree of the encoding $\varphi$

---

**Algorithm 2:** $GLB(x, y)$

---

**Data** : A partial order $P = (X, \leq_P)$;
Two elements $x$ and $y$ of $P$;
The root $T\varphi$;

**Result** : $GLB(x, y)$

**begin**

  *code* $= \varphi(x) \cap \varphi(y)$ ;

  **if** *there exist* $z$ *in* $T\varphi$ *with* $\varphi(z) = code$ **then**

    Return( $z$);

  **else**

    $GLB(x, y)$ do not exist

    Here the program asks the user if $GLB(x, y)$ has a signification

    If yes we add a new element $z$ in $T\varphi$.

**end**

---

**Corollary 12.** *Let $P = (X, \leq_P)$ be a partial order.*

- *The initial computation of the encodings $\varphi$ and $T\varphi$ requires $O(|J(P)| * |X|)$ time.*
- *The computation of $GLB$ requires $O(|J(P)|)$.*

Of course, it is easy to modify the algorithm to explicitely add each new element to $P$. This naturally increases the complexity of the $GLB$ computation to $O(|J(P)| + |X|)$

The $LUB$ operation can be done dually. For simultaneous computation of $GLB$ and $LUB$ we must do some update on $T\varphi$ when adding a new element after $LUB$ and vice versa.

## 6 Conclusion

In this paper we have described a new algorithm to compute the Dedekind MacNeille completion of a partial order, based on $GLB$ and $LUB$ operations. This seems to be interesting for applications in knowledges processing.

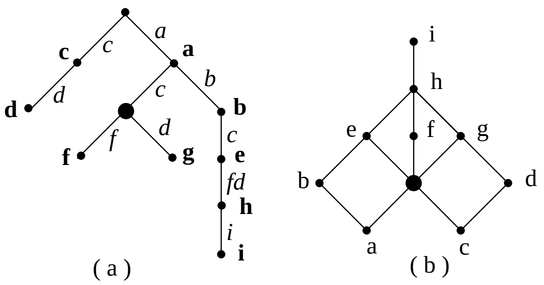



**Fig. 6.** Computation of $GLB(e, f)$

Furthermore our algorithm can be used to compute the Galois lattice or the concept lattice from Wille terminologies. Our arguments for this are based on equivalences between Dedekind MacNeille Completion, Galois lattice and the lattice of Maximal antichains [12].

*Remark 13.* This algorithm was implemented under Maple language (http://www-litp.ibp.fr:/~bertet/).

## References


1. H. Aït-Kaci, R. Boyer, P. Lincoln, and R. Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Langages and Systems*, 11(1):115–146, january 1989.
2. J.P. Bordat. Calcul pratique du treillis de gallois d'une correspondance. In *Math. Sci. Hum, 96*, pages 31–47, 1986.
3. A. Bouchet. Codages et dimensions de relations binaires. *Annals of Discrete Mathematics 23, Ordres: Description and Roles, (M. Pouzet, D. Richard eds)*, 1984.
4. Yves Caseau. Efficient handling of multiple inheritance hierarchies. In *OOPSLA'93*, pages 271–287, 1993.
5. B. A. Davey and H. A. Priestley. *Introduction to lattices and orders.* Cambridge University Press, second edition, 1991.
6. R. Godin and H. Mili. Building and maintening analysis-level class hierarchies using galois lattices. In *OOPSLA'93*, pages 394–410, 1993.
7. M. Habib and L. Nourine. Bit-vector encoding for partially ordered sets. In Proc. of International Workshop ORDAL'94, editor, *Orders, Algorithms and Applications*, number 831, pages 1–12, Lyon, France, July 1994. Springer Verlag.
8. M. Habib and L. Nourine. Tree structure for distributive lattices and its applications. *TCS*, 165(2):391–405, october 1996.
9. C. Jard, G.-V. Jourdan, and J.-X. Rampon. Computing on-line the lattice of maximal antichains of posets. Thechnical report, IRISA, Rennes, France, February 1994. To appear in Order.
10. H.M. MacNeille. Partially ordered sets. *Trans. Amer. Soc.*, 42:416–460, 1937.
11. G. Markowsky. The representation of posets and lattices by sets. *Algebra Universalis*, 11:173–192, 1980.

12. M. Morvan and L. Nourine. Simplicial elimination schemes, extremal lattices and maximal antichain lattices. *Order 13*, No 2:159–173, 1996.
13. R. Wille. Restructuring lattice theory: An approach based on hierarchies of contexts. *in Ordered sets,I. Rival, Eds. NATO ASI No 83, Reidel, Dordecht, Holland*, pages 445–470, 1982.