# A VDM-style specification of the lattice building algorithm

Guy Tremblay

April 2001

## 1 The data types

This first section describes the basic data types used in the algorithm, including the notions of context, concept, etc. An important point to stress is that, in order to better model the algorithm, the ordering associated with a concept lattice is not represented explicitly. Instead, it can be *deduced* from the set of immediate predecessors associated with an concept.

### 1.1 Object and attribute

Objects and attributes are primitive notions which need not be defined further.

$$\mathit{Object} = \mathit{token}$$

$$\mathit{Attribute} = \mathit{token}$$

### 1.2 Context and concept

A (formal) context is defined by a set of objects, a set of attributes, and some incidence matrix relating objects and attributes, that is, indicating which attributes are possessed by an object, and vice-versa.

$$\begin{array}{lrl} \mathit{Context} :: & \mathit{objs} & : \mathit{Object}\text{-}\mathbf{set} \\ & \mathit{attrs} & : \mathit{Attribute}\text{-}\mathbf{set} \\ & \mathit{incidence} & : (\mathit{Object} \times \mathit{Attribute})\text{-}\mathbf{set} \end{array}$$

**where**

$$\begin{array}{l} \mathit{inv\text{-}Context}(\mathit{mk\text{-}Context}(\mathit{objs}, \mathit{attrs}, \mathit{incidence})) \quad \triangleq \\ \quad \mathit{incidence} \subseteq \mathit{objs} \times \mathit{attrs} \end{array}$$

The notion of concept is a *generic* one. More precisely, a concept is always relative to an appropriate context (generally denoted by $K$ in what follows). The *preds* attribute represents the set of immediate predecessors to a concept relative to a given context. Of course, all elements of this set must indeed be predecessors, which is specified by a part of the invariant. The last attribute, *linear-order-info*, is described in section 4.1.

$$\begin{array}{lrl} \mathit{Concept}[\mathit{Context}] :: & \mathit{extent} & : \mathit{Object}\text{-}\mathbf{set} \\ & \mathit{intent} & : \mathit{Attribute}\text{-}\mathbf{set} \\ & \mathit{preds} & : \mathit{Concept}[\mathit{Context}]\text{-}\mathbf{set} \\ & \mathit{linear\text{-}order\text{-}info} & : [\mathit{Linear\text{-}Order\text{-}Info}] \end{array}$$

**where**

$$\begin{array}{l} \mathit{inv\text{-}Concept}[K](c) \quad \triangleq \\ \quad \mathit{is\text{-}concept\text{-}for}((c.\mathit{objs}, c.\mathit{attrs}), K) \wedge \forall p \in c.\mathit{preds} \cdot \mathit{precedes}[K](p, c) \wedge \mathit{linear\text{-}order\text{-}info\text{-}valid}(c) \end{array}$$

### 1.3 Lattice of "concepts" on a context

In the following definition of a *Concept-Lattice*, it must be noted that the *ordering* is a *conceptual* attribute, that is, it will not be defined explicitly, as it can be computed from the set of predecessors associated with each concept.

$$\mathit{Concept\text{-}Lattice}[\mathit{Context}] = \mathit{Concept}[\mathit{Context}]\textbf{-set}$$

$$\mathit{concepts} : \mathit{Concept\text{-}Lattice}[K] \to \mathit{Concept}[\mathit{Context}]\textbf{-set}$$

$$\mathit{concepts}(L) \quad \triangleq \quad L$$

$$\mathit{inv\text{-}Concept\text{-}Lattice}[K](L) \quad \triangleq \quad \mathit{concepts}(L) = \{c \in \mathit{Concept}[K] \mid \mathit{is\text{-}concept\text{-}for}((c.\mathit{extent}, c.\mathit{intent}), K)\}$$

Note: if $c \in \mathit{Concept}[K]$, then this implies that $c$ is indeed an appropriate formal concept for the indicated context. However, for the *concepts* part of the lattice, we need to have all such $\mathit{Concept}[K]$, thus the indicated invariant.

$$\mathit{ordering} : \mathit{Concept\text{-}Lattice}[K] \to (\mathit{Concept}[\mathit{Context}] \times \mathit{Concept}[\mathit{Context}])\textbf{-set}$$

$$\mathit{ordering}(L) \quad \triangleq \quad \{(c_1, c_2) \in \mathit{Concept}[K] \times \mathit{Concept}[K] \mid c_1 \in \mathit{concepts}(L) \wedge c_2 \in \mathit{concepts}(L) \wedge \mathit{precedes}[K](c_1, c_2)\}$$

$$\mathit{ordering} : \mathit{Concept\text{-}Lattice}[K] \to (\mathit{Concept}[\mathit{Context}] \times \mathit{Concept}[\mathit{Context}])\textbf{-set}$$

$$\mathit{ordering}(L) \quad \triangleq \quad \{(c_1, c_2) \in \mathit{Concept}[K] \times \mathit{Concept}[K] \mid c_1 \in \mathit{concepts}(L) \wedge c_2 \in \mathit{concepts}(L) \wedge \mathit{is\text{-}predecessor}(c_1, c_2)\}$$

Note: the *ordering* attribute is defined in two different ways that, for a completed concept lattice, should be equivalent: the first definition is in terms of the general *precedes* ordering relationship whereas the second one is expressed in terms of the immediate predecessors associated with each of the concepts.

## 2 Various auxiliary concepts for characterizing concepts and contexts

The following auxiliary functions correspond to the usual quote operators ($'$) (also called $f$ and $g$). Instead, more explicit and meaningful names have been used, to facilitate the understanding of the algorithm.

### 2.1 Auxiliary functions to define formal concepts

$$\mathit{common\text{-}attributes} : \mathit{Object}\textbf{-set} \times \mathit{Context} \to \mathit{Attribute}\textbf{-set}$$

$$\mathit{common\text{-}attributes}(X, K) \quad \triangleq \quad \{a \in K.\mathit{attrs} \mid \mathit{is\text{-}common\text{-}to\text{-}all}(a, X, K)\}$$

$$\mathit{is\text{-}common\text{-}to\text{-}all} : \mathit{Attribute} \times \mathit{Object}\textbf{-set} \times \mathit{Context} \to \mathbf{B}$$

$$\mathit{is\text{-}common\text{-}to\text{-}all}(a, X, K) \quad \triangleq \quad \forall o \in X \cdot (o, a) \in K.\mathit{incidence}$$

$common\text{-}objects : Attribute\textbf{-set} \times Context \rightarrow Object\textbf{-set}$

$common\text{-}objects(Y, K) \quad \triangleq$
$\quad \{o \in K.objs \mid possesses\text{-}all\text{-}attributes(o, Y, K)\}$

$possesses\text{-}all\text{-}attributes : Object \times Attribute\textbf{-set} \times Context \rightarrow \mathbf{B}$

$possesses\text{-}all\text{-}attributes(o, Y, K) \quad \triangleq$
$\quad \forall a \in Y \cdot (o, a) \in K.incidence$

$is\text{-}concept\text{-}for : (Object\textbf{-set} \times Attribute\textbf{-set}) \times Context \rightarrow \mathbf{B}$

$is\text{-}concept\text{-}for((X, Y), K) \quad \triangleq$
$\quad X \subseteq K.objs \land Y \subseteq K.attrs \land common\text{-}attributes(X, K) = Y \land common\text{-}objects(Y, K) = X$

### 2.2 Auxiliary functions associated with the ordering between concepts

As mentioned earlier, there are two ways in which the ordering is defined:

1. Formally and *conceptually*, in terms of relationships between set of attributes vs. set of objects.
2. Practically, in terms of the immediate predecessor relationship that exists between concepts.

$precedes[K] : Concept[K] \times Concept[K] \rightarrow \mathbf{B}$

$precedes[K](c_1, c_2) \quad \triangleq$
$\quad c_1.extent \subseteq c_2.extent \land c_2.intent \subseteq c_1.intent$

$is\text{-}predecessor : Concept[K] \times Concept[K] \rightarrow \mathbf{B}$

$is\text{-}predecessor(c_1, c_2) \quad \triangleq$
$\quad c_1 = c_2 \;\lor\; c_1 \in c_2.preds \;\lor\; \exists c \in c_2.preds \cdot is\text{-}predecessor(c_1, c)$

## 3 The functions/procedures for building the lattice

### 3.1 The top-level functions described in the paper

$build\text{-}lattice : Context \rightarrow Concept\text{-}Lattice[Context]$

$build\text{-}lattice(K) \quad \triangleq$
$\quad \textbf{if card } K.attrs = 1$
$\quad \textbf{then } build\text{-}lattice\text{-}ground(K)$
$\quad \textbf{else let } (A_1, A_2) = split(K.attrs) \textbf{ in}$
$\qquad \textbf{let } K_1 = create\text{-}sub\text{-}context(A_1, K) \textbf{ in}$
$\qquad \textbf{let } K_2 = create\text{-}sub\text{-}context(A_2, K) \textbf{ in}$
$\qquad assembly(build\text{-}lattice(K_1), build\text{-}lattice(K_2))$

$build\text{-}lattice\text{-}ground\ (K{:}\ Context)\ result{:}\ Concept\text{-}Lattice[K]$
$\textbf{pre } \exists a \in Attribute \cdot K.attrs = \{a\}$
$\textbf{post let } a \textbf{ be such that } K.attrs = \{a\} \textbf{ in}$
$\quad \textbf{let } O{+} = common\text{-}objects(\{a\}, K) \textbf{ in}$
$\quad \textbf{let } c = mk\text{-}Concept[K](O{+}, \{a\}, \{\,\}) \textbf{ in}$
$\quad \textbf{if } O{+} = K.objs$
$\qquad \textbf{then } result = \{c\}$
$\qquad \textbf{else } result = \{c\} \cup \{mk\text{-}Concept[K](K.objs, \{\,\}, \{c\})\}$

$split\ (attrs\colon Attribute\text{-}\mathbf{set})\ A_1, A_2\colon Attribute\text{-}\mathbf{set}$
**pre card** $attrs > 1$
**post** $A_1 \neq \{\,\} \wedge A_2 \neq \{\,\} \wedge A_1 \cup A_2 = attrs$

$create\text{-}sub\text{-}context : Attribute\text{-}\mathbf{set} \times Context \to Context$

$create\text{-}sub\text{-}context(attrs, K) \quad \triangleq$
$\quad mk\text{-}Context(K.objs, attrs, K.incidence \cap (K.objs \times attrs))$

$assembly\ (L_1\colon Concept\text{-}Lattice[K_1], L_2\colon Concept\text{-}lattice[K_2])\ concepts\colon Concept\text{-}lattice[K]$
**ext wr** $L_1, L_2$
**code** $concepts \leftarrow \{\,\}$
$\quad define\text{-}linear\text{-}order(L_1.concepts)$
$\quad define\text{-}linear\text{-}order(L_2.concepts)$
$\quad$ **foreach** $c_1 \in linear\text{-}enumeration(L_1.concepts)$ **do**
$\quad\quad$ **foreach** $c_2 \in linear\text{-}enumeration(L_2.concepts)$ **do**
$\quad\quad\quad E \leftarrow c_1.extent \cap c_2.extent$
$\quad\quad\quad psi\text{-}images \leftarrow \{embed[rank(c^-), rank(c_-)] \mid (c^-, c_-) \in product\text{-}lower\text{-}covers(c_1, c_2)\}$
$\quad\quad\quad c \leftarrow find\text{-}psi(E, psi\text{-}images)$
$\quad\quad\quad$ **if** $c =$ **nil then**
$\quad\quad\quad\quad c \leftarrow mk\text{-}Concept[K](E, c_1.intent \cup c_2.intent, \{\,\})$
$\quad\quad\quad\quad concepts \leftarrow concepts \cup \{c\}$
$\quad\quad\quad\quad update\text{-}order(c, psi\text{-}images)$
$\quad\quad\quad embed[rank(c_1), rank(c_2)] \leftarrow c$

In procedure *assembly*, the appropriate *linear-order-info* is first defined, using *define-linear-order*. This has the effect of setting the appropriate attributes (*linear-order-info*) of each concept relative to the other concepts of the indicated set, which include both the *rank* attribute and the *next* field. All this information is valid since all concepts are made part of the relative order (see section 4.1 for further details about setting this information).

Also, still in procedure *assembly*, the *embed* matrix does not need to be initialized: this is because of the way the linear order is defined (in *define-linear-order*) in conjunction with the way the elements are then used (in *linear-enumeration*), together with the fact that only elements that come before in that order are accessed when creating the *psi-images* (because of *product-lower-covers*). For instance, the first time around the loop, $c_1$ and $c_2$ are bottom elements, so they have no predecessors, so that the set returned by *product-lower-covers* is empty, so no access is made to *embed*.

## 3.2 The functions and procedures used in *assembly*

$define\text{-}linear\text{-}order\ (concepts\colon Concept[K]\text{-}\mathbf{set})$
**ext wr** $concepts$
**post** $linear\text{-}order\text{-}info\text{-}defined(concepts) \wedge\ linear\text{-}order\text{-}info\text{-}valid(concepts)$

$linear\text{-}enumeration\ (concepts\colon Concept[K]\text{-}\mathbf{set})\ result\colon Concept[K]^*$
**post elems** $result = concepts\ \wedge$
$\quad$ **len** $result =$ **card** $concepts\ \wedge$
$\quad \forall i \in \mathbf{N} \cdot 1 \leq i < \mathbf{len}\ result \ \Rightarrow\ is\text{-}predecessor(result[i], result[i+1])$

Note: The implementation of the above two operations are defined in section 5 (*linear-enumeration′* and *define-linear-order′*). Here, only the abstract specifications are given.

$product\text{-}lower\text{-}covers : Concept[K] \times Concept[K] \rightarrow (Concept[K] \times Concept[K])\textbf{-set}$

$product\text{-}lower\text{-}covers(c_1, c_2) \quad \triangleq$
$\quad \{(c_1', c_2) \mid c_1' \in c_1.preds\} \cup \{(c_1, c_2') \mid c_2' \in c_2.preds\}$

$find\text{-}psi : Object\textbf{-set} \times Concept[K]\textbf{-set} \rightarrow [Concept[K]]$

$find\text{-}psi(E, candidates) \quad \triangleq$
$\quad \textbf{if } \exists c \in candidates \cdot \textbf{card } c.extent = \textbf{card } E$
$\quad \textbf{then } \iota\, c \in candidates \cdot \textbf{card } c.extent = \textbf{card } E$
$\quad \textbf{else nil}$

*update-order* ($c$: $Concept[K]$, *candidates*: $Concept[K]$**-set**)
**ext wr** $c$, *candidates*
**code** $known\text{-}intent \leftarrow c.intent$
$\quad define\text{-}linear\text{-}order(candidates)$
$\quad \textbf{foreach } c' \in linear\text{-}enumeration(candidates) \textbf{ do}$
$\quad\quad \textbf{if } c'.intent \cap known\text{-}intent = c.intent \textbf{ then}$
$\quad\quad\quad c.preds \leftarrow c.preds \cup \{c'\}$
$\quad\quad\quad known\text{-}intent \leftarrow known\text{-}intent \cup c'.intent$

# 4 Linear ordering information

## 4.1 The data type and its associated observers

The *linear-Order-Info* attribute associated with a concept, when defined (it can be **nil**, as indicated by the [...] in the definition of section 1.2), specifies information that defines the position of the concept relative to a specific linear ordering that has been produced for a set of elements. This ordering information is defined during the lattice building process and is always relative to a set of concepts.

$$\begin{aligned} Linear\text{-}Order\text{-}Info :: \; & rank : \mathbf{N} \\ & next : [Concept[K]] \end{aligned}$$

$rank : Concept[K] \rightarrow \mathbf{N}$

$rank(c) \quad \triangleq$
$\quad c.linear\text{-}order\text{-}info.rank$

$next : Concept[K] \rightarrow Concept[K]$

$next(c) \quad \triangleq$
$\quad c.linear\text{-}order\text{-}info.next$

## 4.2 Auxiliary predicates that characterize the linear order (related with the topological sorting)

$linear\text{-}order\text{-}info\text{-}valid : Concept[K]\textbf{-set} \to \mathbf{B}$

$linear\text{-}order\text{-}info\text{-}valid(concepts) \quad \underline{\triangle}$
$\quad linear\text{-}order\text{-}info\text{-}defined(concepts)$
$\quad \Rightarrow$
$\quad (\forall c \in concepts \cdot linear\text{-}order\text{-}info\text{-}valid(c)) \wedge (\exists c \in concepts \cdot rank(c) = 0)$

$linear\text{-}order\text{-}info\text{-}valid : Concept[K] \to \mathbf{B}$

$linear\text{-}order\text{-}info\text{-}valid(c) \quad \underline{\triangle}$
$\quad linear\text{-}order\text{-}info\text{-}defined(c)$
$\quad \Rightarrow$
$\quad precedes(c, next(c)) \wedge rank(c) = rank(next(c)) - 1$

$linear\text{-}order\text{-}info\text{-}defined : Concept[K]\textbf{-set} \to \mathbf{B}$

$linear\text{-}order\text{-}info\text{-}defined(concepts) \quad \underline{\triangle}$
$\quad \forall c \in concepts \cdot linear\text{-}order\text{-}info\text{-}defined(c)$

$linear\text{-}order\text{-}info\text{-}defined : Concept[K] \to \mathbf{B}$

$linear\text{-}order\text{-}info\text{-}defined(c) \quad \underline{\triangle}$
$\quad c.linear\text{-}order\text{-}info \neq \mathbf{nil}$

# 5 Detailed implementation of some functions

$define\text{-}linear\text{-}order'$ $(concepts\colon Concept[K]\textbf{-set})$
**ext wr** $concepts$
**code let** $m = \mathbf{maximum}(c\colon Concept[K]$ **such that** $c \in concepts :: \mathbf{card}\ c.intent)$ **in**
  **foreach** $i \in \{0..m\}$ **do**
    $bunches(i) \leftarrow \{\,\}$
  **foreach** $c \in concepts$ **do**
    $bunches(\mathbf{card}\ c.intent) \leftarrow bunches(\mathbf{card}\ c.intent) \cup \{c\}$
  $rank \leftarrow 0$
  $pred \leftarrow \mathbf{nil}$
  **foreach** $i \in [0..m]$ **do**
    **foreach** $c \in bunches(i)$ **do**
      $rank(c) \leftarrow rank$
      $next(c) \leftarrow \mathbf{nil}$
      $rank \leftarrow rank + 1$
      **if** $pred \neq \mathbf{nil}$ **then**
        $next(pred) \leftarrow c$
      $pred \leftarrow c$

$linear\text{-}enumeration'$ $(concepts\colon Concept[K]\textbf{-set})$ $result\colon Concept[K]^*$
**pre** $linear\text{-}order\text{-}info\text{-}defined(concepts)$
**code** $first \leftarrow \iota\, c \in concepts \cdot rank(c) = 0$
  $result \leftarrow [first]$
  $nxt \leftarrow next(first)$
  **while** $nxt \neq \mathbf{nil}$ **do**
    $result \leftarrow result \frown [nxt]$
    $nxt \leftarrow next(nxt)$