A Galois Connection in Composite Objects

Abdelsalam Shanneb and John Potter

Programming Languages and Compilers Group
School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052, Australia
{shanneba,potter}@cse.unsw.edu.au

Technical Report

UNSW-CSE-TR-604

Abstract

With the advent of multiprocessors on the desktop, software applications are increasingly likely to adopt multithreaded architectures. To cope with the complexity of concurrent systems, programmers build systems from thread-safe components. This produces excessive and redundant locking, restricting the potential for concurrency within the system.

Rather than deploying individual thread-safe components, we advocate deferring the deployment of locks until the code dependencies are known. This avoids redundant locking, and allows the granularity of concurrency to be chosen in a flexible way. In earlier work we identified a formal relationship, known as a Galois connection, between the potential for concurrency in a composite system and the locking requirements for its components.

This report highlights the role of fixpoints for lock selection. The subsequent report (UNSW-CSE-TR-605) will investigate strategies for selecting locks in a composite system.

1 Introduction

Concurrent and multi-threaded programming is gaining popularity these days as the complexity of computer applications grows and desktop hardware (especially desktop multiprocessors) become more powerful. Writing concurrent programs is a challenging task. Concurrent access of multiple threads to shared data should be regulated and controlled to preserve data invariants and ensure consistent behaviour. This is what we mean by *thread-safety*. In order to provide thread-safety for software components, the simplest approach is to force mutually exclusive access to the components' interface. For example, in Java, we can declare all the methods of a class as *synchronized*, serializing concurrent calls to each instance of that class so that each object acts as a monitor.

To increase the potential concurrency in a system while maintaining thread safety, we can adopt two complementary approaches. First, we can move monitor boundaries from high-level components down to subcomponents, so that rather than single-threading an entire subsystem, only the shared objects within that subsystem are single-threaded. Second, we can adopt a finer granularity of exclusion control, such as read-write locks, rather than simply single-threading entire components. We can of course adopt both of these approaches simultaneously, and provide finer grain locking internally rather than at the external interface.

In this report, we contribute an advance in our research for reasoning about concurrency and exclusion in component-based object-oriented systems. In the 2004 CSJP Workshop [PSY04] and also in [PSY05], we demonstrated the effectiveness of a general-purpose exclusion lock that can provide any required exclusion. For locking of a single object, the experimental results confirmed the effectiveness of fine-grain (method-level) exclusion control. In our work we use a simple notation for expressing concurrency control in objects. The *algebra of exclusion* [NHP00] is an algebraic notation for expressing exclusion requirements and also the potential concurrency for objects.

In [SP05, SPN05] we characterised the relationship between exclusion requirements and potential concurrency of components in a composite system. Based on prior knowledge of the dependencies amongst a composite's components, and exclusion requirements for primitive components, we have established a more structured approach for propagating exclusion requirements amongst components than was originally proposed in [NHP00]. This model relies on a Galois connection between the outward mapping of exclusion requirements, and the inward mapping of potential concurrency, to reduce the locks considered per component to a minimal subset of those possible. We present a more detailed analysis of this Galois connection in this report. Given the choice of a particular distribution of locks throughout the components of the system, we can calculate whether or not each component is indeed thread-safe, where locks are redundant and where high level or coarse grain locks cause potential for concurrency to be lost.

The remainder of this report is organised as follows. Section 2 gives some background on concurrency and objects, and on exclusion requirements and algebra of exclusion. Section 3 presents a brief overview in partially ordered sets and lattices. In section 4 the notion of *Galois Connection* is defined with several applications such as Formal Concept Analysis. We also present the fixpoint properties. Section 5 is the core of this report where we present a proof of the occurrence of a *Galois Connection* in our model. First, we show the occurrence of a *Galois Connection* between the exclusion requirements and concurrency potential; then we present some examples to illustrate the significance of the fixpoint concept in choosing an appropriate lock. Section 6 provides some concluding remarks.

2 Background

Concurrency and synchronisation have always been associated with the object paradigm since its birth. Early languages and systems [BDMN79, Hoa74, Han73] adopted the object as the unit of synchronisation with the concept of a monitor. We do not attempt a full survey here—see for instance Briot et al [BGL98] or Philippsen [Phi00] for comprehensive surveys of systems and approaches that integrate concurrency and object-oriented languages.

Greenhouse and Scherlis [GS02] present a model for expressing design intent that may help programmers to assure consistency between design and code. Their client policy notation for describing safe/unsafe method interactions is analogous to our method-level exclusion specification for components of a composite. Whereas their concern is to relate design intent to code, our focus is on calculating what locks are sufficient to satisfy given exclusion requirements in a given concurrent environment.

Two recent articles present the spectrum of modern, language-based approaches to synchronisation. Caromel et al [CMT04] describe a monitor-based extension to Java that is similar to various other aspect-oriented synchronisation schemes [BA05, HNP97, Hol99, LK98, Lop05]. Such schemes provide language extensions so that synchronisation or scheduling code can be executed whenever a method enters or leaves an object. Programming languages such as Polyphonic C# [BCF04] and JoinJava [vIK02] are at the other end of the spectrum — incorporating constructs from the Join Calculus [FG02] directly into programming languages. In these languages, synchronisation policies are expressed using chords — combinations of synchronous and asynchronous methods whose execution implicitly establishes rendezvous between multiple threads. Both aspect-oriented and join-calculus languages can be used to implement a wide variety of concurrency management techniques, from basic exclusion, to state-dependent and transactional semantics. This control is provided, however, by writing code, rather than a declarative specification, so there is no notion of a separation of synchronisation policy and mechanism, and synchronisation policies can only be changed — say to distribute locks over composite objects — by changing code.

2.1 Exclusion Requirement

In this report we adopt a general model for synchronisation control for programming languages and component models. It is geared towards composite object models with distinct interfaces on internal components, and is therefore not directly applicable for concurrency control in database systems.

In our model we presume some knowledge of the internal implementation of the component: we need to know the conflicts between the different methods of the interface, this is known as exclusion requirements of a component. The exclusion requirement for a component is specified as the set of method pairs that may conflict. Typically the exclusion requirement depends on the internal implementation of that component. The components exclusion requirements must be met to guarantee safe concurrent access to its interfaces.

As described in [SP05], we model individual components as in Figure 1. In this model internal exclusion requirements R_{int} come from within the component and concurrency potential P_{ext} is determined externally. The lock L is provided by the component. The missing requirement is $R_{ext} = R_{int} - L$ which must be managed externally. Also, the remaining concurrency potential is $P_{int} = P_{ext} - L$.

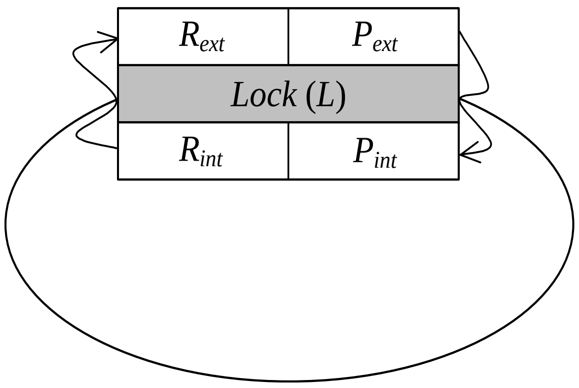

Figure 1: Exclusion Requirement vs Concurrency Potential

We can characterise several properties pertaining to safety, minimum locking, and lock redundancy for a single component.

Safety Condition: For a component to be safe, none of the method pairs in R_I should be concurrently activated; but only those pairs in P_E have this potential. So, provided the lock L blocks all pairs in both R_I and P_E, the component is safe. The safety condition is therefore:

$$R_I \cap P_E \subseteq L$$

This is equivalent to the condition that, at each layer (external and internal), there is nothing in common between exclusion requirements and concurrency potential; in other words

$$R_E \cap P_E = \{\} \quad \text{and} \quad R_I \cap P_I = \{\}$$

Minimum Safe Lock: For given R_I and R_E, the above safety condition clearly identifies the minimum safe lock as:

$$L_{min} \quad \widehat{=} \quad R_I \cap P_E$$

Redundant Locking: If a lock for a component is not the minimum required for safety, then any non-minimal pairs that are blocked are redundant, either because their exclusion is not required by R_I, or they will not occur concurrently, by P_E. For a given local lock L, the redundancy is therefore:

$$L_{red} \quad \widehat{=} \quad L - L_{min}$$

2.2 Algebra of Exclusion

In our earlier work [SP05, SPN05] we have described a simple language for compactly writing exclusion requirements in a compact textual way, which is useful for describing examples.

The basic syntax is given by:

$$e \quad ::= \quad e\,e \mid e|e \mid e \times e \mid \bar{e}$$

In effect this is a language for describing undirected graphs. Each expression denotes a set of elements (the vertices) and a symmetric relation on those elements (the undirected edges). For each of these expressions, the set of elements is the union of the elements of its sub-expressions. The relation described by: the sum e_1e_2 (equivalently $e_1|e_2$) is the union of the relations for the sub-expressions; the product $e_1 \times e_2$ is the union of the relation for e_1e_2 and the symmetric cartesian product of the elements of e_1 and e_2; the completion $\bar{e}$ is the cartesian product of the set of elements of e with itself. The second form of sum operator has lowest precedence; the first form (concatenation) has highest. The reason for incorporating two forms of summation is that it allows us to express multiple read-write sets neatly without any parentheses. For example, the expression $r_1r_2 \times \overline{w_1w_2} \mid r_3r_4 \times \overline{w_3w_4}$ denotes a pair of read-write locks, with two readers and two writers in each set. Without the concatenation operator, this would be written as $(r_1|r_2) \times \overline{w_1|w_2} \mid (r_3|r_4) \times \overline{w_3|w_4}$ which is much less clear.

The main focus of this report is to demonstrate the existence of a *Galois Connection* in our model. Specifically, we show the occurrence of a *Galois Connection* in the relationship between exclusion requirements and concurrency potential. We also show how the fixpoint lattice, associated with the *Galois Connection* property, can be used to partition the sets of exclusion requirements and concurrency potential into equivalence classes to be used in reducing the set of controls considered.

3 Order and Lattices

In this section we start some background for our theoretical work. We present material on order theory and lattices, and show some examples of partitioning exclusion requirements using lattice diagrams. More detailed background can be found in [DP02].

3.1 Ordered Sets

An ordered set is a set equipped with a special type of binary relation. Recall that abstractly a binary relation on a set P is just a subset $R \subseteq P \times P = \{(p, q) : p, q \in P\}$. $(p, q) \in R$ simply means that "p is related to q under R". A binary relation R thus contains all the pairs of points that are related to each other under R.

The basic concept in order theory is that of a partial order; it formalises the notion of a hierarchy and is ubiquitous in mathematics and computer science. An ordered set (or partially ordered set or poset) is an ordered pair $(P,\leq)$ of a set P and a binary relation $\leq$ contained in $P \times P$, called the order (or partial order) on P, such that

1. The relation $\leq$ is *reflexive*. That is, each element is related to itself; $\forall p \in P : p \leq p$
2. The relation $\leq$ is *antisymmetric*. That is, if p is related to q and q is related to p, then p must equal q; $\forall p, q \in P : (p \leq q \wedge q \leq p) \Rightarrow (p = q)$.
3. The relation $\leq$ is *transitive*. That is, if p is related to q and q is related to r, then p is related r; $\forall p, q, r \in P : (p \leq q \wedge q \leq r) \Rightarrow q \leq r$.

The elements of P are called the *points* of the ordered set. Order relations introduce a hierarchy on the underlying set. The statement $p \leq q$ is read p is less than or equal to q or q is greater than or equal to p. The antisymmetry of the order relation ensures that there are no two-way ties ($p \leq q$ and $q \leq p$ for distinct p and q) in the hierarchy. The transitivity (in conjunction with the antisymmetry) ensures that no cyclic ties ($p_1 \leq p_2 \leq ... \leq p_n \leq p_1$ for distinct p_1,p_2,...,p_n) exist.

3.2 Bounds

Let $A \subseteq P$ be a subset of the poset P. An element p is an *upper bound* for A if $a \leq p$ for every a in A. An element p in P is the *least upper bound* of A (l.u.b of A), or *supermum* of A (sup A) if p is an upper bound of A, and $a \leq b$ for every a in A implies $p \leq b$ (i.e., p is the smallest among the upper bounds of A). Similarly we can define what it means for p to be a *lower bound* of A, and for p to be the *greatest lower bound* of A (g.l.b of A), also called the *infimum* of (inf A).

3.3 Lattices

If the *least upper bound* of A exists in a poset, we denote it by $\bigvee A$, and if the *greatest lower bound* of A exists, we denote it by $\bigwedge A$. If $A = x_1, x_2, ..., x_n$ is a finite subset, then we can also write $\bigvee A = x_1 \vee x_2 \vee ... \vee x_n$, and $\bigwedge A = x_1 \wedge x_2 \wedge ... \wedge x_n$.

A poset L is a **lattice** if $x \vee y$ and $x \wedge y$ exist for all elements $x, y \in L$. In a lattice $\bigvee$ and $\bigwedge$ can be regarded as binary operations, called **join** and **meet**, $\bigvee$: $L \times L \rightarrow L$ and $\bigwedge$: $L \times L \rightarrow L$. Based on this definition, the natural numbers $\mathbb{N}$, the integers $\mathbb{Z}$, the rational numbers $\mathbb{Q}$ and the real numbers $\mathbb{R}$ are all lattices with their usual orders are ordered sets. In these $x \vee y = max\{x, y\}$ and $x \wedge y = min\{x, y\}$.

Any set of sets is ordered by set inclusion $\subseteq$. Similarly, geometric figures (like circle in a plane) are ordered by inclusion. The simplest example of a set system ordered by inclusion is the power set $\mathbb{P}(X)$ of a set X. Figure 2 depicts a sublattice of the power set (all possible subsets) of the three element set $\{a, b, c\}$.

Notice that the set at the top of the figure, U, consists of all of the elements of the set.

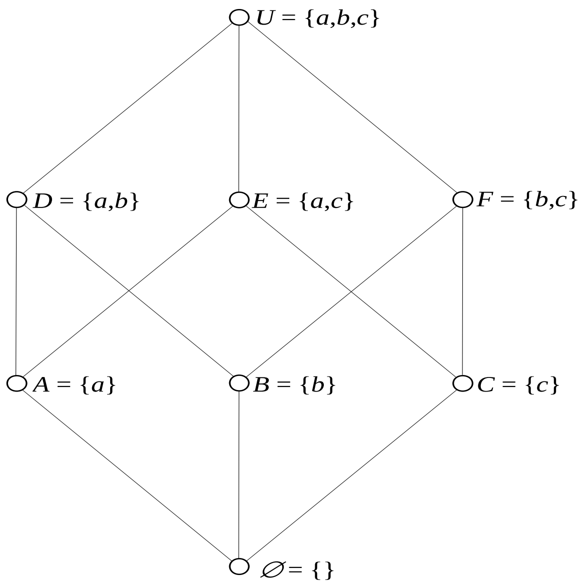

Figure 2: A Sublattice of the Set {a,b,c}

Sets D, E, and F are each in the subset relation to U, (for example, every element of D is an element of U); and so on. This subset relation is a basis for partially ordering the sets. We have placed the sets D, E and F below U to represent the fact that they are ordered with respect to U. Note that they are not ordered with respect to each other. It is for this reason that we refer to this as a partial order. Notice further that the set A is a subset of D and of E; B a subset of D and F and so on. Finally, at the bottom of the figure is the empty set. It is ordered in the figure with respect to sets A, B and C.

Another example more related to our work is depicted in Figure 3. This is a lattice of exclusion requirements on the set of methods $\{1, 2\}$. This complete lattice depicts the powerset of all possible symmetric pairs of the set. Adding one more element to the previous set increases the number of symmetric pairs from 3 to 6, and the size of the lattice to $2^6 = 64$ points (nodes) with 7 order levels as shown in Figure 4. In general, with n methods, there are $\frac{n(n+1)}{2}$ symmetric pairs, and $2^{\frac{n(n+1)}{2}}$ lattice elements with $\frac{n(n+1)}{2} + 1$ levels.

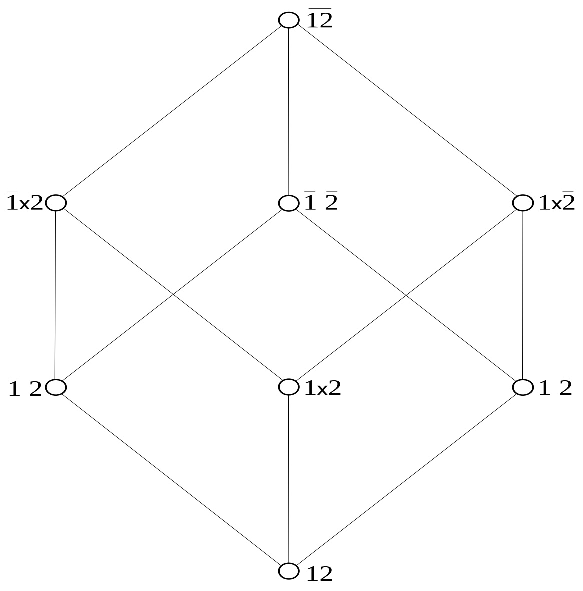

Figure 3: Lattice of Exclusion Requirements on {1,2}

4 Galois Connection

In this section we introduce the notion of *Galois Connection* and show some related examples. We also talk about the fixpoint notion and its properties.

4.1 What is a Galois Connection?

A *Galois Connection* can be used as an effective research tool, it provides an interesting way of comparing two pre-ordered sets. Consider the two partially ordered sets P and Q, also consider the two functions $\rhd : P \to Q$ and $\lhd : Q \to P$. $\rhd$ and $\lhd$ form a *Galois Connection*, written $GC(\rhd, \lhd, P, Q)$, for all $p \in P$, $q \in Q$, if

$$p \ \leq_P \ q^{\lhd} \quad \text{iff} \quad p^{\rhd} \ \leq_Q \ q$$

4.2 Examples: Formal Concept Analysis

Formal Concept Analysis [DP02] is increasingly used for data mining, where the sets are objects and attributes. *Formal Concept Analysis* is based on the philosophical understand-

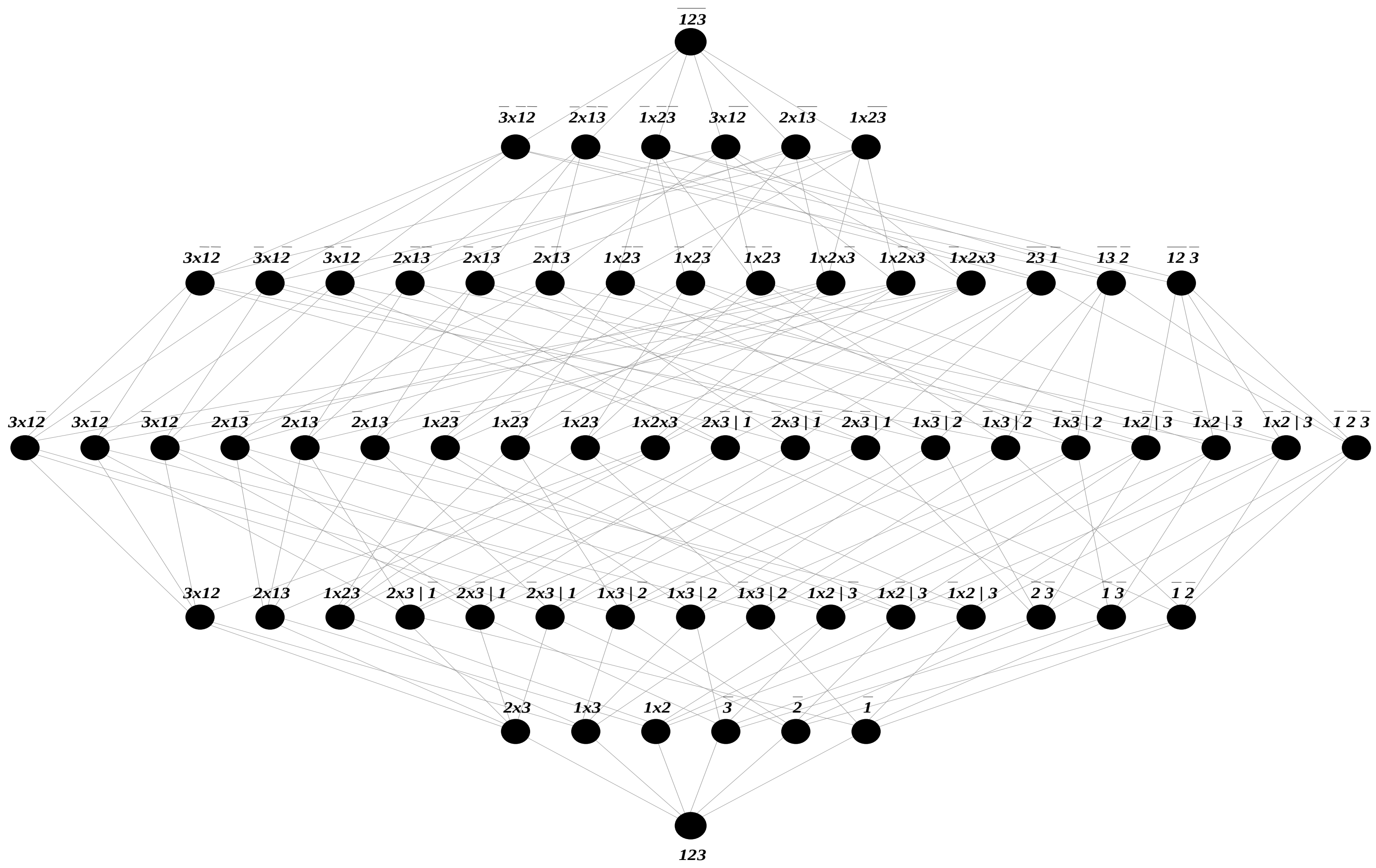

Figure 4: Lattice of Exclusion Requirements on {1,2,3}

ing of a concept as a unit of thought described by its extension and intension. These two entities exhibit a *Galois Connection.*

The extension consists of all *objects* belonging to the concept, and the intension contains all *attributes* shared by all the objects of the extension. For formalising this understanding a set G of objects and a set M of attributes has to be specified. They are connected by a binary relation I between G and M. The triple (G, M, I) is called a formal context. For each formal context its formal concepts can be introduced as pairs (A, B) where A is a subset of G, B is a subset of M and A is the set of all objects which have all attributes of B and B is the set of all attributes shared by all objects of A. These pairs constitute a *Galois Connection* between G and M. Consider:

$$\begin{aligned} A^{\triangleright} &= \{\, b \mid \forall\, a \in\ A. \quad a\ I\ b\} \\ B^{\triangleleft} &= \{\, a \mid \forall\, b \in\ B. \quad a\ I\ b\} \end{aligned}$$

Then A *Galois Connection* exists between $(G, \subseteq)$ and $(M, \supseteq)$ if:

$$A \subseteq B^{\triangleleft} \quad \textsf{iff} \quad A^{\triangleright} \supseteq B$$

4.3 Examples: Galois Connection in Usage Relation

Consider two components (sets of methods) A and B, with a known dependency relation $\textsf{uses} : A \leftrightarrow B$. Define the pair of mappings $(\textsf{uses}^{\triangleright}, \textsf{uses}^{\triangleleft})$ as:

$$\begin{aligned} \textsf{uses}^{\triangleright} &: \mathbf{P}A \to \mathbf{P}B \\ \textsf{uses}^{\triangleright}(X) &= \{\, y \in B \mid \exists x \in X \,.\, x \ \textsf{uses}\ y \,\}^{c} \\ \textsf{uses}^{\triangleleft} &: \mathbf{P}B \to \mathbf{P}A \\ \textsf{uses}^{\triangleleft}(Y) &= \{\, x \in A \mid \exists y \in Y \,.\, x \ \textsf{uses}\ y \,\}^{c} \end{aligned}$$

The following property, which is straightforward to prove, establishes that the pair of mappings, $(\mathsf{uses}^{\rhd}, \mathsf{uses}^{\lhd})$ forms a *Galois Connection* between the orders $(\mathbf{P}A, \subseteq)$ and $(\mathbf{P}B, \supseteq)$. The corollary identifies the fixpoints of the connection which is the key for us.

$$X \ \subseteq\ \mathsf{uses}^{\lhd}\,(Y) \quad \mathsf{iff} \quad \mathsf{uses}^{\rhd}\,(X) \ \supseteq\ Y$$

Indeed, this example of a *Galois Connection* is just the same as that of formal concept analysis working with the complement of the usage relation.

4.4 Fixpoints of a Galois Connection

What makes a *Galois Connection* between two posets P , Q interesting is that it establishes a partition of convex subsets of P, and another of Q. Further more, there is a 1-1 connection between these two partitions. The partitions are determined by the inverse image of the composite maps $\rhd\lhd$ on P, and $\lhd\rhd$ on Q. These maps have the following properties:

$$\begin{array}{rll} \mathit{closure}: & 1) & p^{\rhd\lhd\rhd} = p^{\rhd} \\ & 2) & q^{\lhd\rhd\lhd} = q^{\lhd} \\ \mathit{convexity}(\mathit{max}/\mathit{min}): & & \text{if} \quad p_1 \ \leq\ p_2 \ \leq\ p_1^{\lhd\rhd} \\ & & \text{then} \quad p_2^{\lhd\rhd} = p_1^{\lhd\rhd} \end{array}$$

We call the sets

$$\begin{array}{rl} & \{p^{\rhd\lhd} \mid p \ \in\ P\} \\ \text{and} & \{q^{\lhd\rhd} \mid q \ \in\ Q\} \end{array}$$

the fixpoints of the *Galois Connection.* The fixpoints form a lattice. In the case of formal concept analysis, the usual conceptual hierarchy defined by the containment relation between the extents is an order relation on the set of all formal concepts of the given formal context. This ordered set is always a complete lattice, called the *concept lattice* of the given formal context. Each complete lattice is a concept lattice. Each concept lattice represents its formal context without any loss of information.

4.5 Examples: Fixpoint for Usage Relation

We illustrate the fixpoint lattice for the uses relations with some simple examples.

Example 1.

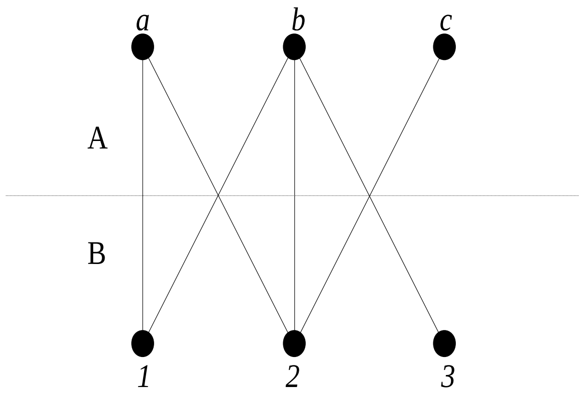

Figure 5: Example 1 Usage Graph

Figure 5 shows the graph of a uses relation, where $A = \{a, b, c\}$ and $B = \{1, 2, 3\}$. Figure 6 shows two sublattices that depict the order of the elements in component A and B.

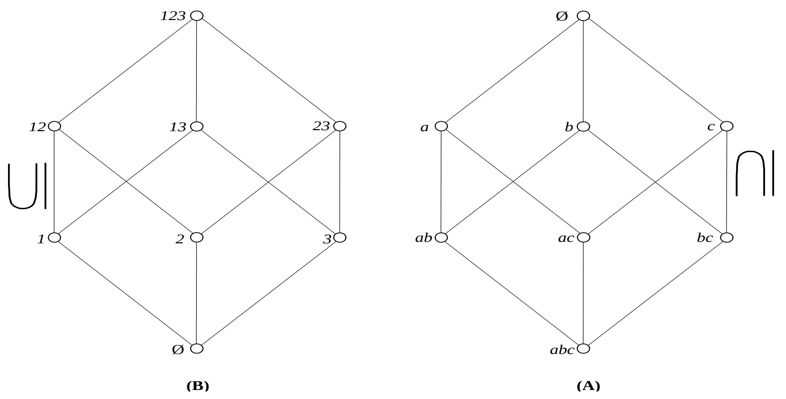

Figure 6: Example 1 Sublattices

A	$uses^{\triangleright}$	$uses^{\triangleright\triangleleft}$
ϕ	$\{1,2,3\}$	ϕ
$\{a\}$	$\{3\}$	$\{a,c\}$
$\{b\}$	ϕ	$\{a,b,c\}$
$\{c\}$	$\{1,3\}$	$\{c\}$
$\{a,b\}$	ϕ	$\{a,b,c\}$
$\{a,c\}$	$\{3\}$	$\{a,c\}$
$\{b,c\}$	ϕ	$\{a,b,c\}$
$\{a,b,c\}$	ϕ	$\{a,b,c\}$

Table 1: Component A *uses* Mappings

Table 1 and 2 show result of applying *uses* function on subsets of the two components A and B respectively. Notice that the last column contains just the fixpoints of the mappings. Indeed the entries in the middle column of one table are the fixpoints in the other table. e.g. $uses^{\triangleleft}\{1\} = uses^{\triangleleft}\{1,3\}$ and $uses^{\triangleleft}\{\phi\} = uses^{\triangleleft}\{2\} = uses^{\triangleleft}\{1,2\} = uses^{\triangleleft}\{2,3\} = uses^{\triangleleft}\{1,2,3\}$.

B	$uses^{\triangleleft}$	$uses^{\triangleleft\triangleright}$
ϕ	$\{a,b,c\}$	ϕ
$\{1\}$	$\{c\}$	$\{1,3\}$
$\{2\}$	ϕ	$\{1,2,3\}$
$\{3\}$	$\{a,c\}$	$\{3\}$
$\{1,2\}$	ϕ	$\{1,2,3\}$
$\{1,3\}$	$\{c\}$	$\{1,3\}$
$\{2,3\}$	ϕ	$\{1,2,3\}$
$\{1,2,3\}$	ϕ	$\{1,2,3\}$

Table 2: Component B *uses* Mappings

Figure 7 shows how the $\triangleright\triangleleft$ mapping induces a partition on A whose components are convex and with maximal elements that are fixpoints of the mapping (the black dots in the figure). Similarly for $\triangleleft\triangleright$ and B. Then in Figure 8 we see that there is an order isomorphism between the fixpoints of A and B, and that they form a lattice (in this case, a total order).

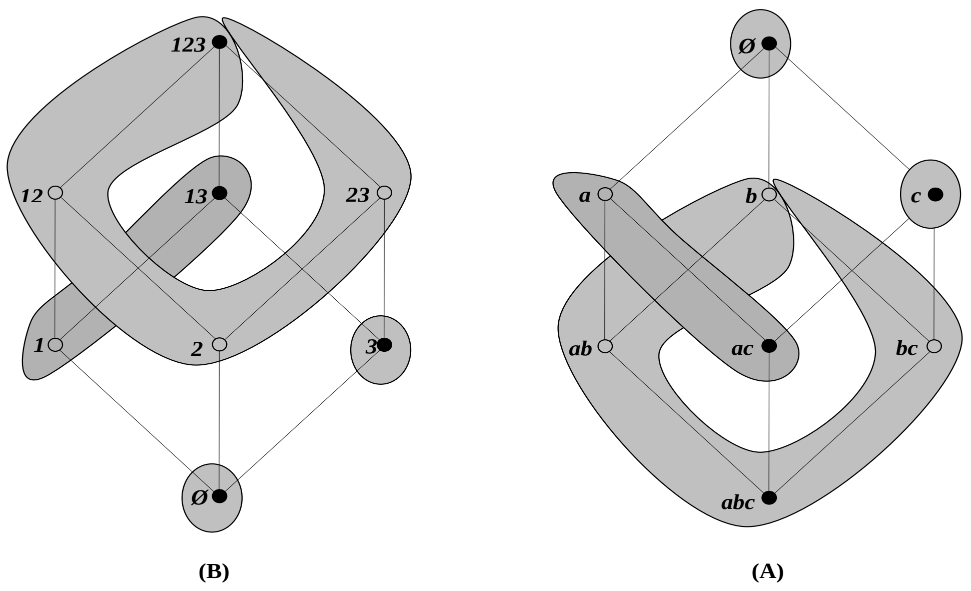

Figure 7: Example 1 Highlighted Sublattices

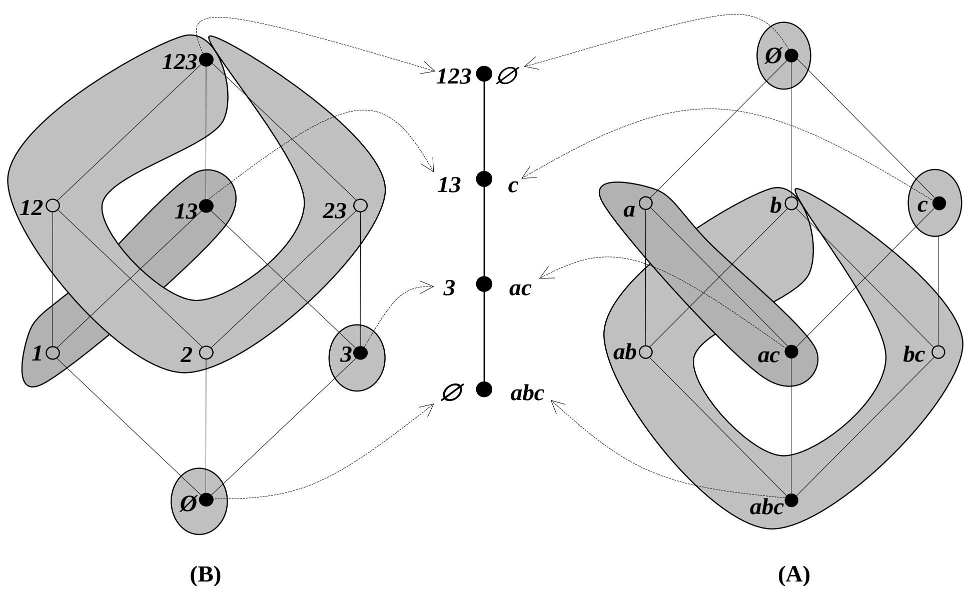

Figure 8: Example 1 Joint Sublattice

Example 2.

In this example we use the usage relation graph shown in Figure 9. Again Table 3 and 4 show the result of applying the *uses* function on subsets of the two components A and B respectively. The results of the $\triangleright\triangleleft$ mapping function on component A and the $\triangleleft\triangleright$ mapping on B are shown in Figure 10. Finally, Figure 11 shows the joint fixpoint lattice for both mappings.

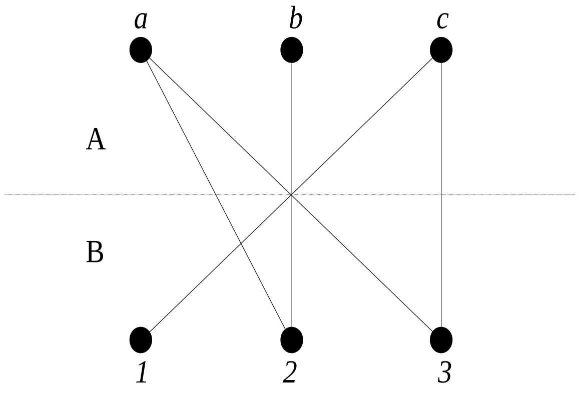

Figure 9: Example 2 Usage Graph

A	$uses^{\triangleright}$	$uses^{\triangleright\triangleleft}$
ϕ	$\{1,2,3\}$	ϕ
$\{a\}$	$\{1\}$	$\{a,b\}$
$\{b\}$	$\{1,3\}$	$\{b\}$
$\{c\}$	$\{2\}$	$\{c\}$
$\{a,b\}$	$\{1\}$	$\{a,b\}$
$\{a,c\}$	ϕ	$\{a,b,c\}$
$\{b,c\}$	ϕ	$\{a,b,c\}$
$\{a,b,c\}$	ϕ	$\{a,b,c\}$

Table 3: Component A *uses* Mappings

B	$uses^{\lhd}$	$uses^{\lhd\rhd}$
ϕ	$\{a, b, c\}$	ϕ
$\{1\}$	$\{a, b\}$	$\{1\}$
$\{2\}$	$\{c\}$	$\{2\}$
$\{3\}$	$\{b\}$	$\{1, 3\}$
$\{1, 2\}$	ϕ	$\{1, 2, 3\}$
$\{1, 3\}$	$\{b\}$	$\{1, 3\}$
$\{2, 3\}$	ϕ	$\{1, 2, 3\}$
$\{1, 2, 3\}$	ϕ	$\{1, 2, 3\}$

Table 4: Component B *uses* Mappings

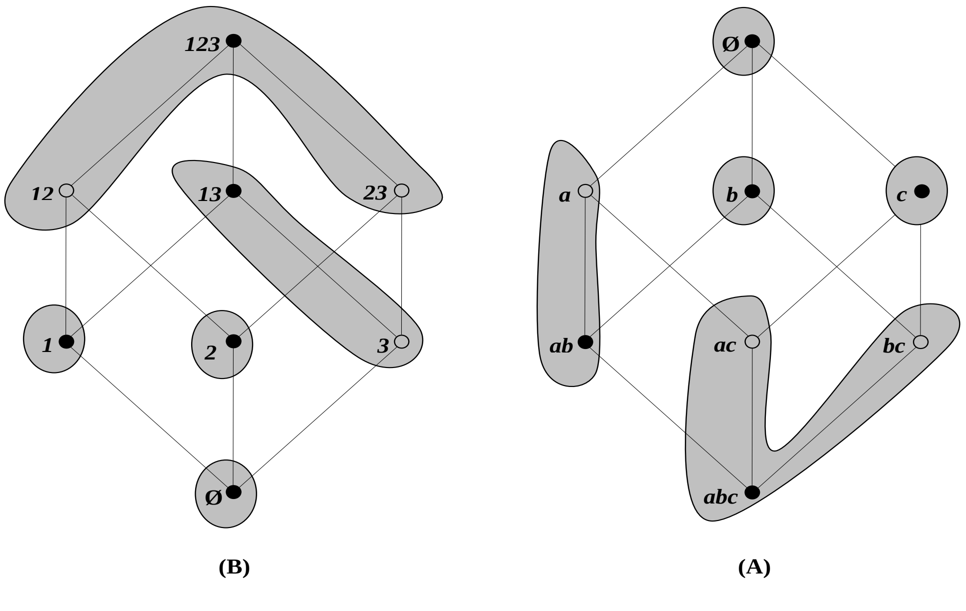

Figure 10: Example 2 Highlighted Sublattices

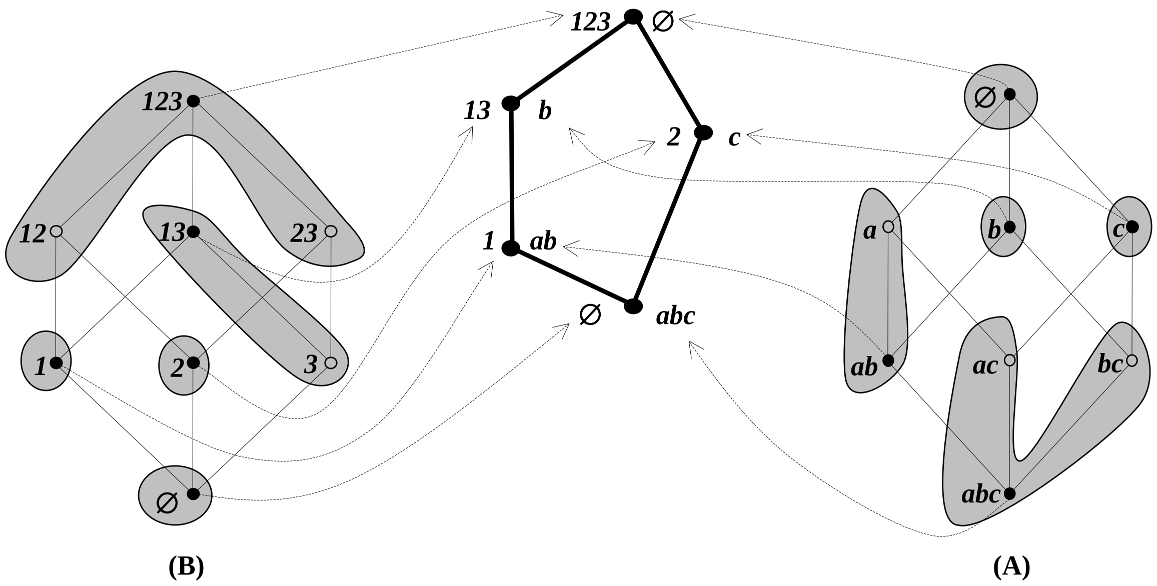

Figure 11: Example 2 Joint Sublattice

5 The Galois Connection in Our Model

In this section we show how a *Galois Connection* occurs in our model, that is, we demonstrate the occurrence of the *Galois Connection* between the exclusion requirements and concurrency potential.

5.1 Definition

In our model, we defined basic mapping of internal exclusion requirements of an internal component to its composite interface according to the composite usage pattern. This mapping is simply achieved by substituting each method name in each inner component with the name of the composite interface that uses that method.

External requirement on inner component C $[uses(m)/m]$

$\Rightarrow$ Internal requirement on outer composite

for each m in C.

Consider the example in Figure 12 which depicts a simple composite object that contains two internal components. Using the usage pattern shown we apply our simple mapping function to get the internal requirement: $\overline{I_1} \times\ I_2$ on the composite.

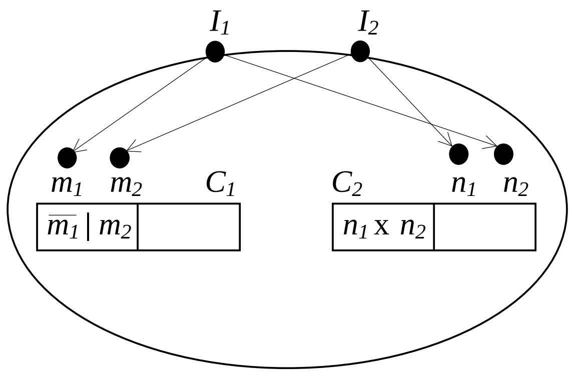

Figure 12: Component Dependency

In order to discuss fixpoint properties of these mappings we find it convenient to reformulate them using relational composition rather than substitution of usage sets. Consider the (directed) usage relation u defined on the underlying set of names. We also consider exclusion requirements R (and concurrency potential P) as symmetric relations on their respective sets. Given such a relation R, we define its outward mapping with respect to u as:

$$R^{\leftarrow} \quad \widehat{=} \quad u \cdot R \cdot u^{-1}$$

and the inward mapping likewise:

$$P^{\rightarrow} \quad \widehat{=} \quad u^{-1} \cdot R \cdot u$$

Here the dot operator denotes forward composition of relations, and the u^{-1} denotes relational inverse. $R^{\leftarrow}$ will relate those pairs of methods that call (according to u) some pair in R. Similarly $P^{\rightarrow}$ will relate those pairs of methods that are called by some pair in P.

An inner exclusion requirement $R = R_{E\ inner}$ on the components of a composite object already takes locks of the internal components into account. It can be propagated to an internal exclusion requirement $R_{I\ outer}$ for the object, via the inner-outer usage relation. So $R_{I\ outer}\ = R^{\leftarrow}$ The complement of this is the object's internal allowed concurrency

$A_{I\ outer}$. We use $^{\lhd}$ to denote this mapping, combining inner-outer usage, and complement. So $A_{I\ outer} = R^c_{I\ outer} = R^{\lhd}$.

In summary we map inner exclusion requirements R to allowable outer concurrency $R^{\lhd}$ using:

$$\begin{aligned} R^{\lhd} &\mathrel{\widehat{=}} (R^{\leftarrow})^c \\ &= (u.R.u^{-1})^c \end{aligned}$$

Furthermore we can propagate P inwards to calculate the concurrency potential for the internal components, the complement of which is the collection of *excluded activation* pairs, $P^c_{E\ inner} = P^{\rhd}$, where

$$\begin{aligned} P^{\rhd} &\mathrel{\widehat{=}} (P^{\rightarrow})^c \\ &= (u^{-1}.P.u)^c \end{aligned}$$

see Figure 13.

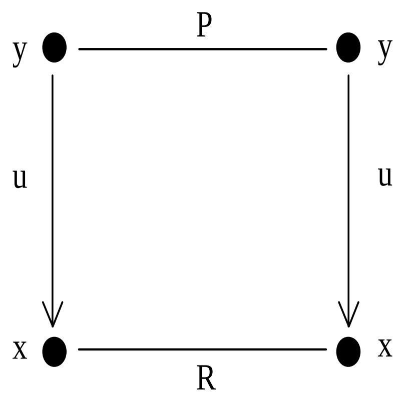

Figure 13: Outwards and Inwards Mapping

For safety at the outer level, the concurrency potential $P = P_{I\ outer}$, must not exceed the maximum allowable, so we require $P \subseteq R^{\lhd}$. For safety at the inner level, the excluded concurrent activations for the components must contain all the required exclusion: $R \subseteq P^{\rhd}$.

In the next section we prove the inner-outer correspondence:

Inner Safety holds iff *Outer Safety holds*

which is formalised as:

$$R \subseteq P^{\triangleright} \quad \mathsf{iff} \quad P \subseteq R^{\triangleleft}$$

This is precisely the statement that the pair of mappings $(^{\triangleright}, ^{\triangleleft})$ is a *Galois Connection* [DP02]. It follows that $^{\triangleleft\triangleright}$ and $^{\triangleright\triangleleft}$ are closure operators for inner exclusion requirements and outer concurrency potential respectively.

Furthermore, as we have seen in Section 3, there is an order isomorphism between the fixpoints of these closure operators, $\{R^{\triangleleft\triangleright}\}$ and $\{P^{\triangleright\triangleleft}\}$, where R varies over all possible inner exclusion requirements and P over all outer concurrency potentials. Because $(^{\triangleright}, ^{\triangleleft})$ is a *Galois Connection* [DP02], given an inner exclusion requirement R, its corresponding fixpoint $R^{\triangleleft\triangleright}$ is the maximum exclusion expression that maps to the same outer allowed concurrency as does R, that is $R^{\triangleleft\triangleright\triangleleft} = R^{\triangleleft}$. By finding these fixpoints, we partition the inner exclusion expressions into equivalence classes, each with a unique maximal representative. Similarly we can partition the concurrency potentials of the outer layer. The partitions of exclusion and concurrency potential expressions are order isomorphic. In fact the *Galois Connection* between exclusion requirements and concurrency potential is really just a special case of the one sketched in Section 3.3 and 3.5.

5.2 The Theorem

Given sets A,B and relations

$$\begin{array}{rcl} u & : & A \leftrightarrow B \\ P & : & A \leftrightarrow A \\ R & : & B \leftrightarrow B \end{array}$$

define

$$P^{\rhd} = (u^{-1}.P.u)^c$$

$$R^{\lhd} = (u.R.u^{-1})^c$$

Then

$$P \subseteq R^{\lhd} \quad \textsf{iff} \quad R \subseteq P^{\rhd}$$

Proof: $\exists \quad y_1\, y_2 \quad \in \quad P - R^{\lhd}$

$\textsf{iff} \quad \exists \quad y_1 y_2 \quad \in \quad P \cap \ (u.R.u^{-1})$

$\textsf{iff} \quad \exists \quad y_1 y_2 \quad \cdot \quad y_1 P y_2 \quad \wedge \quad y_1(u.R.u^{-1})y_2$

$\textsf{iff} \quad \exists \quad y_1 y_2\, x_1 x_2 \quad \cdot \quad y_1 P y_2 \quad \wedge \quad y_1 u x_1 \ \wedge \ x_1 R x_2 \quad \wedge \quad x_2 u^{-1} y_2$

$\textsf{iff} \quad \exists \quad x_1 x_2\, y_1 y_2 \quad \cdot \quad x_1 R x_2 \quad \wedge \quad x_1 u^{-1} y_1 \quad \wedge \quad y_1 P y_2 \quad \wedge \quad y_2 u x_2$

$\textsf{iff} \quad \exists \quad x_1 x_2 \quad \cdot \quad x_1 R x_2 \quad \wedge \quad x_1(u^{-1}.P.u)x_2$

$\textsf{iff} \quad \exists \quad x_1 x_2 \quad \in \quad R \cap (u^{-1}.P.u)$

$\textsf{iff} \quad \exists \quad x_1 x_2 \quad \in \quad R - P^{\rhd}$

Thus

$$P - R^{\lhd} = \phi \quad \textsf{iff} \quad R - P^{\rhd} = \phi$$

as required to prove.

5.3 The Role of Fixpoints for Lock Selection

The importance of the *Galois Connection* and its associated fixpoint lattice is that it allows us to precisely characterise locks for a component which provide safety without redundancy. We find that two locks on an inner component may have the same effect at the outer level; because of the fixpoint property, there will be a unique minimal lock with the same effect. For lock selection purposes it is sufficient to restrict attention to these.

For example, for the usage relation of Example 2 (Figure 9) in Section 3.5, we obtain the fixpoint lattice depicted in Figure 14. Let's assume that component B has exclusion requirement $R_I = \overline{13}\times 2$. And suppose a lock is arbitrarily chosen as $L_B = \bar{1}\times 3|2$. Figure 15 shows the lock configuration after selecting the given lock. The remaining requirement is $R_E = R_I - L = 1\bar{3}\times 2$. This then induces a maximal allowable concurrency potential for component B: $P_I = R_E^{\triangleleft} = a\bar{b}c$. Note that this is a fixpoint. The corresponding concurrency potential for B is $P_I{}^{\rightarrow} = 1\bar{2}3$.

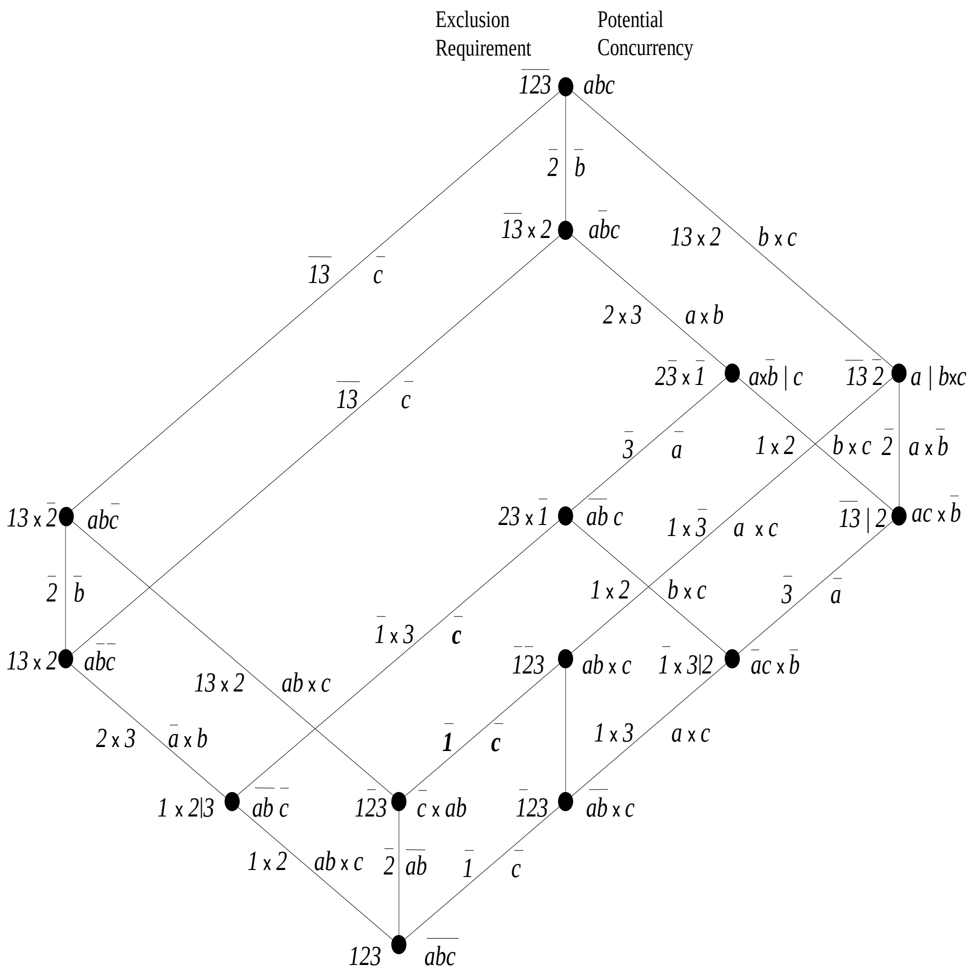

Figure 14: Full Lattice - Example 1

The redundancy for the chosen lock is:

$$
\begin{aligned}
&= \bar{1}\times 3|2 - (\overline{13}\times 2 \cap 1\bar{2}3) \\
&= \bar{1}\times 3|2
\end{aligned}
$$

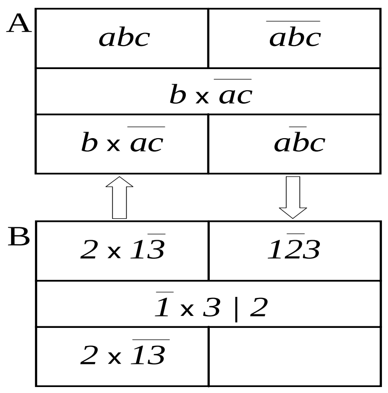

Figure 15: Outward and Inward Mapping

This implies that all of the chosen lock is redundant - there will be no concurrent behaviour on B for it to block. The unique minimal lock with equivalent behaviour is $L = \phi$ (i.e. no lock).

If instead as in Figure 16 we had chosen $L = \overline{13}|2$ (a mutex on 1 and 3), we find $R_E = 13 \times 2$ and $P_I = a\bar{b}\bar{c}$, with corresponding concurrency potential for $B = \bar{2}|\overline{13}$.

In this case, the lock redundancy is:

$$
\begin{aligned}
&= \overline{13}|2 - (\overline{13} \times 2 \cap \bar{2}|\overline{13}) \\
&= \overline{13}|2 - \overline{13}|2 \\
&= \phi
\end{aligned}
$$

No part of the lock is redundant.

This choice of lock corresponds to moving from one fixpoint ($a\bar{b}c$) to another ($a\bar{b}\bar{c}$) in the lattice. The lattice diagram of Figure 14 has labelled the corresponding edges with the difference between the two fixpoints, namely $\bar{c}$ corresponding to $\overline{13}$. All non-redundant choices for locks can be read off the lattice directly. They are: $\phi, \overline{13}, 2 \times 3, 2 \times 13, 2 \times \bar{3}, 1\bar{3} \times 2, 1 \times 2 \times \bar{3}, \bar{3} \times \bar{1}2, \overline{13} \times 2$.

With a different usage relation (Example 1, Figure 5.4) we obtain a different lattice as

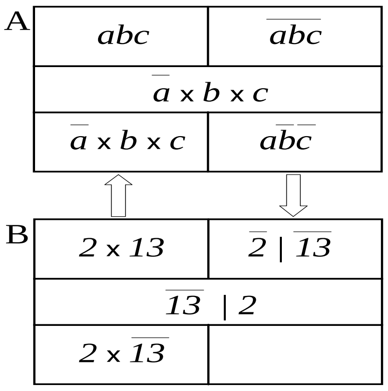

Figure 16: Outward and Inward Mapping

in Figure 17. Assume that the given exclusion requirement for the inner component B is $R_I = \bar{1}2 \times \bar{3}$ which is also a fixpoint.

To calculate all non-redundant locks we simply need to calculate the difference between the given exclusion requirement and the fixpoint value for each fixpoint below the initial one. Figure 17 has labelled all locks that correspond to each fixpoint, so lock number 5 is calculated as: $\bar{1}2 \times \bar{3} - 12\bar{3} = \bar{1}2 \times 3$.

Table 5 shows all non-redundant lock combinations for both components A and B. Observe the trade-off between outer locking (on A) and inner locking (on B).

Lock Number	Lock B	Lock A
1	ϕ	$\bar{a}c \times \bar{b}$
2	2×3	$\overline{ab}$
3	$\bar{1}$	$ac \times \bar{b}$
4	$\bar{1}\|2 \times 3$	$a \times \bar{b}$
5	$\bar{1}2 \times 3$	$\bar{b}$
6	$\bar{1}2 \times \bar{3}$	ϕ

Table 5: Calculation of Minimum Locks

6 Conclusion

In this report, we have formally demonstrated the occurrence of a *Galois Connection* between the exclusion requirements and concurrency potential. We also demonstrated how

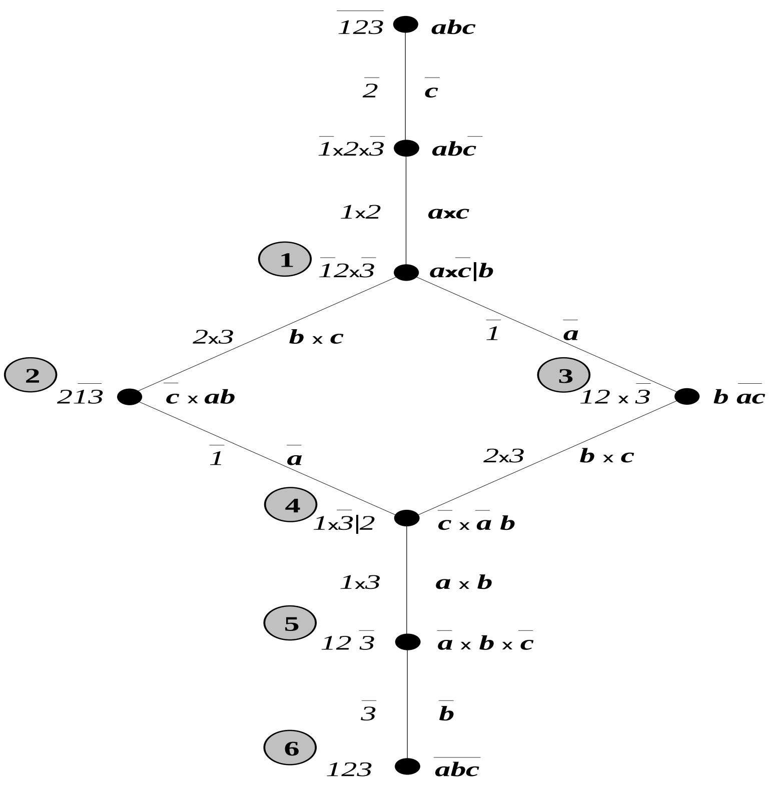

Figure 17: Full Lattice - Example 2

the fixpoint lattice associated with the *Galois Connection* allows us to characterise the non-redundant locking combinations between two layers of components in a composite object. This implies that, for lock selection, we only need to consider a subset of the exponential number of lock combinations. In our next technical report (UNSW-CSE-TR-605), we will exploit this restriction to efficiently guide users through the process of lock selection.

References

[BA05] Lodewijk Bergmans and Mehmet Akşit. *Aspect-Oriented Software Development*, pages 63–95. Addison-Wesley, Boston, 2005.

[BCF04] Nick Benton, Luca Cardelli, and Cédric Fournet. Modern concurrency abstractions for C#. *ACM Trans. Program. Lang. Syst.*, 26(5):769–804, 2004.

[BDMN79] G. M. Birtwistle, O. J. Dahl, B. Myhrhaug, and K. Nygaard. *Simula Begin.* Studentlitteratur, Box 1717, S-221 01 Lund, Sweden, 1979.

[BGL98] Jean-Pierre Briot, Rachid Guerraoui, and Klaus-Peter Lohr. Concurrency and Distribution in Object-Oriented Programming. *ACM Computing Surveys*, 30(3):291–329, 1998.

[CMT04] Denis Caromel, Luis Mateu, and Éric Tanter. Sequential object monitors. In Martin Odersky, editor, *Object-Oriented Programming ECOOP2004, 18th European Conference, Oslo, Norway, June 14-18, 2004, Proceedings*, Lecture Notes in Computer Science, pages 316–340. Springer, 2004.

[DP02] B. A. Davey and H. A. Priestley. *Introduction to Lattice and Order.* Cambridge University Press, 2nd edition, 2002.

[FG02] C. Fournet and G. Gonthier. The join calculus: a language for distributed mobile programming. *LNCS*, 2395:268–385, 2002.

[GS02] Aaron Greenhouse and William L. Scherlis. Assuring and evolving concurrent programs: annotations and policy. In *Proceedings of the 22rd International Conference on Software Engineering, ICSE 2002, 19-25 May 2002, Orlando, Florida,USA*, pages 453–463, 2002.

[Han73] Per Brinch Hansen. *Operating System Principles.* Prentice Hall PTR Upper Saddle River, NJ, USA, 1973.

[HNP97] David Holmes, James Noble, and John Potter. Aspects of Synchronization. In *TOOLS '97: Proceedings of the Technology of Object-Oriented Languages and Systems – Tools–25*, pages 2–14, Washington, DC, USA, 1997. IEEE Computer Society.

[Hoa74] C. A. R. Hoare. Monitors: An Operating System Structuring Concept. *Commun. ACM*, 17–10:549–557, 1974.

[Hol99] David Holmes. *Synchronisation Rings - Composable Synchronisation for Object-Oriented Systems*. PhD thesis, Macquarie University, 1999.

[LK98] Cristina Videira Lopes and Gregor Kiczales. Recent Developments in Aspect. In *ECOOP Workshops*, pages 398–401, 1998.

[Lop05] Cristina Videira Lopes. *Aspect-Oriented Software Development*, chapter AOP: A Historical Perspective (What's in a Name?), pages 97–122. Addison-Wesley, Boston, 2005.

[NHP00] James Noble, David Holmes, and John Potter. Exclusion for Composite Objects. In *Proceedings of the* 15th *ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 13–28. ACM Press, 2000.

[Phi00] Michael Philippsen. A Survey of Concurrent Object-Oriented Languages. *Concurrency – Practice and Experience*, 12(10):917–980, 2000.

[PSY04] John Potter, Abdelsalam Shanneb, and Eric Yu. Exclusion Control for Java and C# : Experimenting with Granularity of Locks. In *PODC Workshop on Concurrency and Synchronization in Java Programs*. Memorial University of Newfoundland, Canada – (2004-01), Jul. 2004.

[PSY05] John Potter, Abdelsalam Shanneb, and Eric Yu. Demonstrating the Effectiveness of Exclusion Control for Components. In *ASWEC '05: Proceedings of the* 16th *Australian conference on Software Engineering*, pages 344–353, Washington, DC, USA, 2005. IEEE Computer Society.

[SP05] Abdelsalam Shanneb and John Potter. Flexible Exclusion Control for Composite Objects. In *CRPIT '38: Proceedings of the* 28th *Australasian conference on Computer Science*, pages 277–286, Darlinghurst, Australia, 2005. Australian Computer Society, Inc.

[SPN05] Abdelsalam Shanneb, John Potter, and James Noble. Exclusion Requirements and Potential Concurrency for Composite Objects. *(Elsevier) Science of Computer Programming*, 58(3):344–365, 2005.

[vIK02] G. Stewart von Itzstein and D. Kearney. Applications of join java. In *Proceedings of the Seventh Asia Pacific Computer Systems Architecture Conference ACSAC'2002*, pages 1–20. Australian Computer Society, 2002.