JOURNAL OF OBJECT TECHNOLOGY
Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering. ©JOT, 2007

Vol. 6, No. 9, Special Issue: TOOLS EUROPE 2007, October 2007

Incremental Lock Selection for Composite Objects

John Potter and **Abdelsalam Shanneb**
Programming Languages and Compilers Group
School of Computer Science and Engineering
University of New South Wales, Sydney, Australia

With the trend towards multi-core processors, support for multi-threaded programming is increasingly important. We are interested in providing development and deployment options to allow programmers to select minimal locks, achieving maximal concurrency, at different levels of granularity within a composite system. We explore local properties of the fixpoint lattice of a Galois connection between exclusion requirements and concurrency potential of a composite object. This allows us to develop incremental algorithms for lock selection. Implemented within integrated development environments, such algorithms will allow programmers to interactively select minimal locks with safety.

1 INTRODUCTION

One of the main challenges in concurrent programming is preserving data invariants by controlling concurrent access of multiple threads to shared data. In systems with mutable state, programmers are often overwhelmed by the code dependencies, and admit defeat by imposing single-threading throughout their code, applying simple mutex controls at all external access points. In languages like Java and C# where all objects are heap-allocated, this can result in much redundant locking and loss of potential concurrency. We can increase the potential concurrency in a system while maintaining thread safety by adopting two complementary approaches. First, we can move monitor boundaries from high-level objects down to sub-objects, 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 just mutexes.

This paper presents a novel approach for lock selection in a composite object system, given exclusion requirements at the innermost levels, and known potential for concurrent activity in the operating environment. One of the novel contributions of this paper is the identification of strategies that allow programmers to systematically select locks, either adding or subtracting locks as they traverse the composite object structure in either a top-down or bottom-up manner. At each stage of the traversal, the lock selection is kept safe but with no redundancy—in other words, with as much locking as necessary, but no more. At each stage the current lock selection can be incrementally modified. The choices for this modification are determined from a lattice structure (the fixpoint lattice of a Galois connection) that identifies certain critical combinations of exclusion requirements and potential concurrency. A second novelty of this paper, is an algorithm for incremental calculation of relevant fixpoints, computing them on demand. There is no need to compute the whole lattice, which may be exponentially large. The examples of this paper were checked using an implementation of this algorithm.

Our underlying model comes from Noble et al [1], which introduced the *Exclusion Algebra* as a syntactic device for propagating locking requirements through the call dependencies of a composite object system. This work assumes that a system is composed of objects, that method calls can be factored into a tree-structure based on the object composition, and that exclusion requirements can be calculated through the layers of a system, starting with specified innermost exclusions. We recognized the inherent duality of this model in [2, 3] where we introduced the complementary notions of *exclusion requirements* and *concurrency potential.* The duality essentially arises from different views of what locking achieves: from the internal point of view, locking protects inconsistent states from occurring; from the external point of view, locking blocks calls, and restricts the amount of concurrency that may be achieved. For our purposes, we consider a single lock to be a device that controls method calls, and blocks pairs of calls from executing concurrently. This includes mutexes, read-write locks and their generalization, read-write sets. Elsewhere [4] we have reported on performance experiments for a general purpose exclusion lock. This lock implements a simple table-based scheme for an object wrapper so that the lock can be configured to block any specified set of conflicting pairs of method calls.

In Section 2 we review our model for a composite object system with internal exclusion requirements and external concurrency potential. We characterize safety for given distributions of locks, and identify minimal combinations of locks for achieving safety. This relies on the fixpoint lattice of a Galois connection between exclusion requirements and concurrency potential, that we identified in earlier work. We summarize the key ideas in Section 3 and provide some examples. In [5] we demonstrated how lock selection can be made easy with appropriate tool support. This leads into the new contributions of this paper. In Section 4 of this paper we introduce specific strategies for lock selection, and in Section 5 we formulate local properties about the fixpoint lattice which enable us to present our algorithm for computing lock choices at each step of a selection process. We close with a discussion of practical concerns, future work and a brief summary.

2 A MODEL FOR HIERARCHICAL EXCLUSION CONTROL

Exclusion Requirements and Concurrency Potential

We present a simple concurrency control model for objects. We assume all object access is via a defined interface comprising a set of methods M. Furthermore we assume access is controlled by method level locks associated with an interface. A particular lock L can be specified in terms of the set of pairs of methods whose calls are excluded from being concurrently executed on the object. So a lock can be simply described by:

$$L \subseteq M \times M.$$

Thus, for us, a lock is simply a set of pairs of methods of an object. We use subset ordering to compare locks: one lock is smaller than another if it is a subset of it.

Each object has an *exclusion requirement* R which is determined by internal sharing of methods; shared data can be treated as a special case with separate reader and writer methods.

$$R \subseteq M \times M.$$

When a lock L is selected for an object with internal exclusion requirement R_I, some part of the requirement may still be missing. We call this the *external exclusion requirement* R_E:

$$R_E \quad \widehat{=} \quad R_I - L.$$

Any missing requirement R_E can only be met by restricting the context of external method calls for the object. To allow us to capture this notion we introduce the concept of *concurrency potential* P:

$$P \subseteq M \times M.$$

Because a lock will block concurrent execution of any of its pairs of methods (that is, it excludes these pairs), any concurrency potential available internally (P_I) can be determined by subtracting the lock's pairs of methods from the external concurrency potential (P_E) according to:

$$P_I = P_E - L.$$

The main ideas of this model are sketched in Figure 1. The object bubble represents the internal part of the system which can only be accessed via the interface, where a local lock L may be selected. The exclusion requirements of the internal part appear at the object interface as an internal requirement R_I. Following the squiggly arrow, this requirement may be partially satisfied by L. The remainder

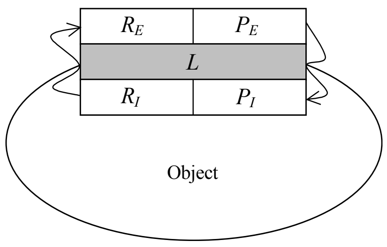

Figure 1: Required Exclusion & Concurrency Potential

then appears as the external exclusion requirement R_E. Similarly the external environment has the potential to make concurrent calls on the system. The concurrency potential, P_E, specifies which pairs of methods can be called concurrently. Following the squiggly line, this external concurrency potential gets reduced by the lock L to provide an internal concurrency potential P_I.

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

Safety Condition: For an object to be safe, none of the method pairs in R_I should be executed concurrently ; but only those pairs in P_E have this potential. So, providing the lock L blocks all pairs that are in both R_I and P_E, the object is safe. The safety condition is therefore:

$$\textsc{Safe} \quad \widehat{=} \quad R_I \cap P_E \subseteq L$$

This is equivalent to having nothing in common between exclusion requirements and concurrency potential at either layer (external or internal); in other words

$$\textsc{Safe} \quad \text{iff} \quad R_E \cap P_E = \{\} \quad \text{iff} \quad R_I \cap P_I = \{\}$$

Minimum Safe Lock: For given R_I and P_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 an object 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:

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

Algebra of Exclusion

Earlier work [1, 2, 3] described a simple language for compactly writing exclusion requirements. This is useful for our examples. The basic syntax is given by:

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

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 symmetric relation is defined as by cases: 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 with 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; method name n denotes a singleton set. The second form of sum operator has lowest precedence; the first form (concatenation) has highest. 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 reader and two writer methods in each set.

The Composite Model

In this section we extend the model for individual objects to a composite model in which objects may be composed of other objects. We assume that all component objects of a composite are independent of one another, so that the composite is tree-structured. Figure 2 suggests how the internal exclusion requirements R_I of a composite are determined from the external exclusion requirements R_{Ei} of the composite's internal objects. The upward arrows indicate where outward propagation of exclusion requirements occurs. Conversely the internal concurrency potential P_I determines the external concurrency potential of the inner objects, as indicated by the downward arrows. Observe the duality: exclusion requirements depend on internal objects, whereas concurrency potential depends on the external environment. Furthermore these dependencies can be composed through the layers of the composite system.

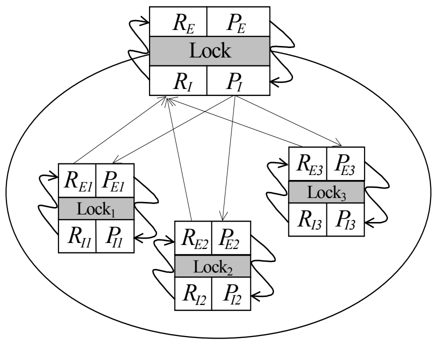

Figure 2: Hierarchical Composite

Outward Propagation of Exclusion Requirements

We now show how external exclusion requirements for inner objects propagate to internal exclusion requirements for their container. If an inner object requires exclusion between methods, n_1 and n_2 say, and does not provide this exclusion with its own local lock, then $n_1 \times n_2$ is part of the inner object's external exclusion requirements. This requirement should be met by the callers of n_1 and n_2; therefore, the exclusion $n_1 \times n_2$ propagates to an internal required exclusion between all callers of n_1 and n_2 for the container.

Consider a composite object with methods I_1, I_2, and two internal objects C_1 and C_2 with methods m_1, m_2 and n_1, n_2. Assume the usage dependency is: $I_1 \mapsto m_1 \quad I_1 \mapsto n_2 \quad I_2 \mapsto m_2 \quad I_2 \mapsto n_1$ and that each object has internal exclusion requirements $C_1 : \overline{m_1}|m_2$ and $C_2 : n_1 \times n_2$. We calculate the internal requirement on the composite object from the external requirements on inner objects by substituting the users of each internal method in the exclusion expressions.

For $C_1 : \overline{m_1}|m_2[I_1/m_1, I_2/m_2]$ yields $\overline{I_1}|I_2$

For $C_2 : n_1 \times n_2[I_2/n_1, I_1/n_2]$ yields $I_2 \times I_1$

We then combine the result as: $(\overline{I_1}|I_2)|(I_2 \times I_1)$ yields $\overline{I_1} \times I_2$. So, the external exclusion requirements of the internal objects collectively form a composition of requirements representing the internal exclusion requirements of the composite: $R_I = \overline{I_1} \times \ I_2$. If the local lock provides the needed internal required exclusion (R_I) (i.e $L = \overline{I_1} \times \ I_2$) the external exclusion requirement ($R_E = R_I - L$) is void (written as $I_1|I_2$ in Figure 3).

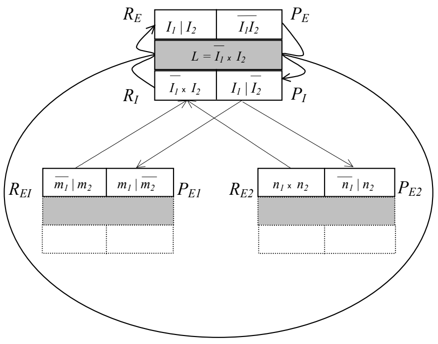

Figure 3: Outward Propagation of Exclusion Requirements and Inward Propagation of Concurrency Potential

Inward Propagation of Concurrency Potential

Inward propagation of external concurrency potential is dual to the outward propagation of the internal exclusion requirements. In our example we consider the composite object to be operating in a fully concurrent environment; Figure 3 shows an external concurrency potential $\overline{I_1 I_2}$ assuming that any pair of methods may be concurrently activated.

The composite object is provided with a lock for the top object; that lock may reduce some or all of the incoming concurrency potential. Before propagating the concurrency potential inward, we have to consider the provided lock. $P_I = P_E - L$ simply calculates what is left of the concurrency potential after blocking the activities specified by the lock. So the internal concurrency potential to be propagated

[illegible]

into the internal objects is $\overline{I_1 I_2} - (\overline{I_1} \times I_2) = I_1 | \overline{I_2}$ (Figure 3). The process of inward propagation of external concurrency potential is simply the dual of the outward propagation of internal exclusion requirements into external exclusion requirements. For our example, we find, as shown in Figure 3:

For $C_1 : I_1 | \overline{I_2}$ yields $m_1 | \overline{m_2}$

For $C_2 : I_1 | \overline{I_2}$ yields $n_2 | \overline{n_1}$

Locking and Safety

We have seen that an individual object is safe just when $R_E \cap P_E = \{\}$, or equivalently, $R_I \cap P_I = \{\}$. For a composite system, we require this to hold at the outermost level. Indeed it then follows that all objects will be safe. For the example of Figure 3 we found $R_E = I_1 I_2$. So the maximum allowable concurrency is $P_E = \overline{I_1 I_2}$, that is, there is no limit on the concurrency — we have fully met the exclusion requirements at the outer level.

3 FORMALISATION OF THE MODEL

This section explains how knowledge of the call dependencies between objects of a system can be exploited to reduce the number of conflict pairs that need to be considered for locking purposes. Our model relies on knowing *exclusion requirements* for internal objects, the *potential concurrency* for calls on an external interface, and the *usage dependency* of the external interface on the internal objects. A significant aspect of our model is the recognition of a Galois connection between external potential concurrency and internal exclusion requirements. The key idea is that we need only consider the fixpoints of this connection for a given usage relation. We briefly introduce some notation to allow us to illustrate this idea; appropriate Order Theory background can be found in [6].

Galois Connection for Usage Relation

As a special case, we consider a composite object with one component object. The relation $uses : A \leftrightarrow B$ describes the call dependency of the outer object A on the inner object B. The outward propagation of an exclusion requirement R for B can be expressed as $uses.R.uses^{-1}$ (the '.' operator denotes forward relational composition). Similarly the inward propagation of concurrency potential P for A is $uses^{-1}.P.uses$. But these propagated values can be restricted by given values: for object A we assume an external concurrency potential is given, P_{E_A} say, and for B we assume an internal exclusion requirement, R_{I_B} say. We now define a pair of mappings $(\triangleright, \triangleleft)$ as:

$$\triangleright \quad : \quad \mathbf{P}(A \times A) \rightarrow \mathbf{P}(B \times B)$$

$$\begin{array}{rcll} \lhd & : & \mathbf{P}(B \times B) \to \mathbf{P}(A \times A) & \\ P^{\rhd} & = & R_{I_B} - P^{\to}, & \text{where } P^{\to} = uses^{-1}.P.uses \\ R^{\lhd} & = & P_{E_A} - R^{\leftarrow}, & \text{where } R^{\leftarrow} = uses.R.uses^{-1} \end{array}$$

Here, when P is the internal concurrency potential for the outer object A, $P^{\to}$ is the external concurrency potential for the inner object B. Consequently $P^{\rhd} = R_{I_B} - P^{\to}$ is the maximal external exclusion requirement for which B remains safe. In other words, given P, we can determine the minimal lock required on B, namely, $L_{B_{min}} = R_{I_B} - P^{\rhd} = R_{I_B} \cap P^{\to}$. Dually, when R is the external exclusion requirement for the inner object B, $R^{\leftarrow}$ is the internal exclusion requirement for A, and $R^{\lhd}$ is the maximal internal concurrency potential for which A remains safe. In other words, given R, we can determine the minimal lock required on A, namely, $L_{A_{min}} = P_{I_A} - R^{\lhd} = P_{I_A} \cap R^{\leftarrow}$.

The fact that inner objects being safe just when outer ones are, leads to the recognition of the following.

Property 1 (Galois Connection)

$(\rhd, \lhd)$ is a Galois connection on $(\mathbf{P}(P_{I_A}), \mathbf{P}(R_{I_B}))$: given $P \subseteq P_{I_A}$ and $R \subseteq R_{I_B}$,

$$P^{\rhd} \supseteq R \quad \text{iff} \quad P \subseteq R^{\lhd}.$$

The Galois connection is between subsets of the external concurrency potential P_{I_A} and subsets of the internal exclusion requirement R_{I_B}. The proof of Property 1 is straightforward. Property 2 summarises standard results for Galois connections [6].

Property 2 (Fixpoints)

1. The composed operators $\rhd\lhd$: $\mathbf{P}(A \times A) \to \mathbf{P}(A \times A)$ and $\lhd\rhd$: $\mathbf{P}(B \times B) \to \mathbf{P}(B \times B)$ are closure operators.

2. The set of fixpoints (the range) of $\rhd\lhd$ is order isomorphic to that of $\lhd\rhd$.

3. The fixpoints form a complete sublattice of $\mathbf{P}(A \times A)$ (resp. $\mathbf{P}(B \times B)$).

The key point for us is that the *uses* relation induces a one-one mapping between particular subsets, the fixpoints of outer concurrency potential and inner exclusion requirements. The significance for lock selection is that we only need to consider a subset of possible locks: essentially those that take us to fixpoints.

Example 1. Consider the usage relation between objects $A = \{a, b, c\}$ and $B = \{1, 2, 3\}$ as in Table 1 . Let us assume that $P_{E_A} = \overline{abc}$ (full concurrency potential) and $R_{I_B} = \overline{123}$. We obtain the full fixpoint lattice depicted in Figure 4. The edges on the lattice are labelled with the difference between the fixpoint sets they connect. In this example, there are 6 different pairs of methods in both A and B. So the total number of concurrency potentials on A, paired with all the exclusion requirements on B, is $2^6 \times 2^6 = 4096$. From Figure 4 we see that we only need to consider 14 of these combinations.

$uses$		$uses^{-1}$	
A	B	B	A
a	$2, 3$	1	c
b	2	2	a, b
c	$1, 3$	3	a, c

Table 1: Usage Relation for Examples 1 and 2

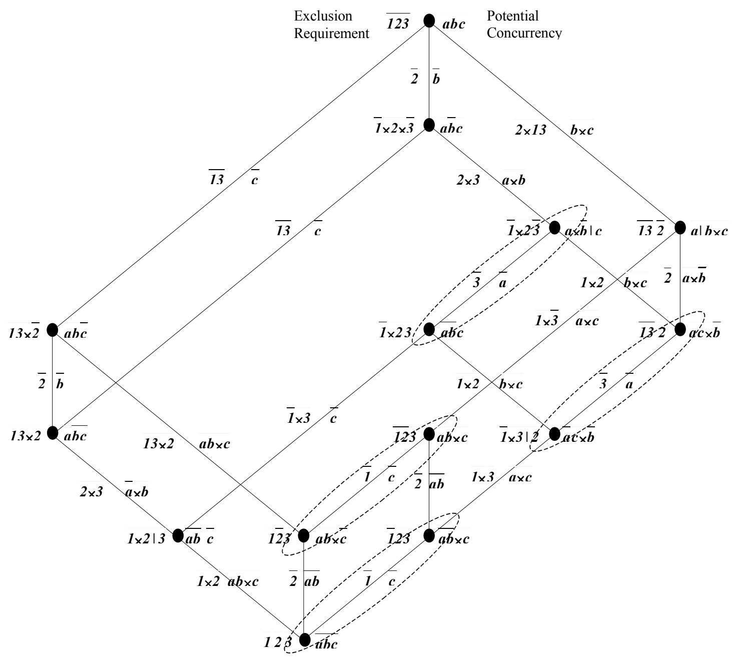

Figure 4: Lattice for Example 1 with $R_{I_B} = \overline{123}$

Example 2. Next, with the same usage relation, let us assume that object B has exclusion requirement $R_I = 1 \times \bar{2} \times 3$ and concurrency potential for A remains the same, $P_{E_A} = \overline{abc}$. Then, the lattice of fixpoints shown in Figure 5 depicts a restricted form after collapsing the circled edges in the lattice of Figure 4. These edges are labelled $\bar{1}$ or $\bar{3}$, which are the exclusion pairs now missing from R_I. The number of points to consider is now only 10.

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 an object which provide safety without redundancy. We find that two locks on an inner object 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.

Suppose in Example 2 a lock is arbitrarily chosen as $L_B = 1 \times \bar{2}|3$. Figure 6(a) shows the lock configuration after selecting the given lock. The remaining require-

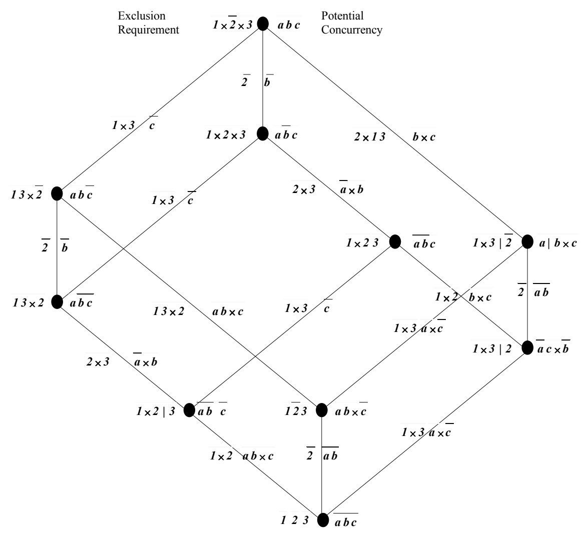

Figure 5: Lattice for Example 2 with $R_{I_B} = 1 \times \bar{2} \times 3$

ment is $R_E = R_I - L = 12 \times 3$. This then induces a maximal allowable concurrency potential for object B: $P_I = R_E^{\triangleleft} = a\bar{b}c$. Note that this is a fixpoint. This also implies we must provide a lock on A equal to $b \times \overline{ac}$. The corresponding concurrency potential for B is $P_I^{\rightarrow} = 1\bar{2}3$.

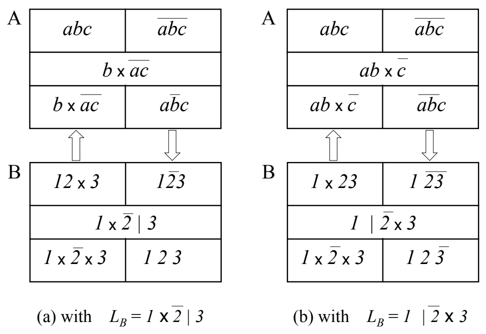

Figure 6: Different Lock Selections in Example 2

The redundancy for the chosen lock is: $1 \times \bar{2}|3 - (1 \times \bar{2} \times 3 \cap 1\bar{2}3) = 1 \times 2|3$. This implies that the exclusion between methods 1 and 2 is redundant – there will be no concurrent behaviour on B for it to block. The unique minimal lock with equivalent behaviour is $L = 1\bar{2}3$, a mutex on method 2. This corresponds to selecting the fixpoint pair $1 \times 2 \times 3/a\bar{b}c$ in Figure 5.

If instead as in Figure 6(b) we had chosen $L = 1|\bar{2} \times 3$ (a read-write lock on 3 and 2), we find $R_E = 1 \times 23$ and $P_I = \overline{abc}$, with corresponding concurrency potential for B as $P_E = 1\overline{23}$. In this case, the lock redundancy is $1|\bar{2} \times 3 - (1 \times \bar{2} \times 3 \cap 1\overline{23}) = \{\}$. No part of the lock is redundant. Note that this non-redundant lock corresponding to the fixpoint $1 \times 23/\overline{ab}c$ can be determined by accumulating the edge labels on the path from the top of the lattice to the chosen fixpoint.

Example 3. Consider the uses relation depicted in Table 2. Assuming $P_{E_A} = \overline{abc}$ we obtain the lattices of Figure 7. Lattice (a) is for $R_{I_C} = \overline{123}$. Lattice (b) is for $R_{I_C} = \bar{1}2 \times \bar{3}$.

uses		$uses^{-1}$	
A	C	C	A
a	$1, 2$	1	a, b
b	$1, 2, 3$	2	a, b, c
c	2	3	b

Table 2: Usage Relation for Example 3

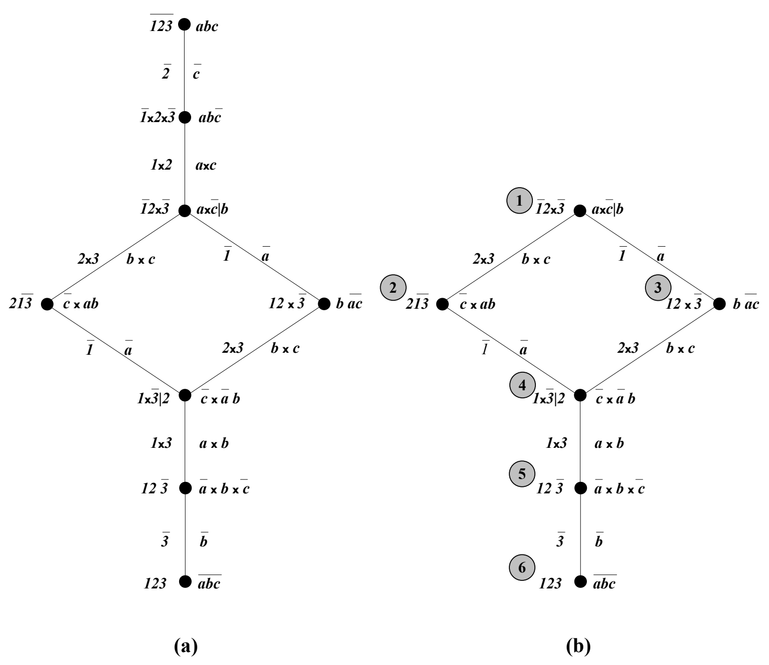

Figure 7: Fixpoint Lattices for Example 3

We can enumerate all the safe combinations of non-redundant locks for objects A and C, simply by adding or removing method pairs corresponding to edge labels on the lattice. For lattice (b) these lock combinations are shown in Table 3, where each row is numbered according to the fixpoint labels of Figure 7(b). The first row corresponds to the top fixpoint, with no locking on the inner object C, and full locking on the outer object A. The lock on A is the difference between the external concurrency potential $a\bar{b}c$ and the fixpoint $a \times \bar{c}|b$; this can also be determined by accumulating all the lattice edge labels from the bottom to the top. Moving to the second row corresponds to moving to the fixpoint one edge down to the left; in turn, this corresponds to adding a lock on A and removing part of the lock on C—they are complementary, as indicated by the edge label. This complementary addition and removal of locks works for all edge traversals, and is the key to our strategies for lock selection.

Lock Number	Lock C	Lock A
1	$-$	$\bar{a}c \times \bar{b}$
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	$\bar{1}2 \times \bar{3}$	$--$

Table 3: Locks for Example 3

4 LOCK SELECTION STRATEGIES

So far, we have seen that with knowledge of usage dependency, internal exclusion requirements and external concurrency potential, we can theoretically identify all safe, non-redundant lock combinations. There is a trade-off between internal locks, which will be more fine-grained, and which may need to be repeatedly acquired, and external locks which may be held for a longer duration, and therefore reduce overall concurrency.

We cannot expect a programmer to calculate these combinations. Instead, we propose using a tool to systematically offer choices between internal and external locks. The tool would offer choices that preserve lock safety and minimality, by simply offering the choices that correspond to all the edges incident on a given fixpoint of the lattice. The programmer need not be aware of the underlying lattice, but rather just steps through a lock selection wizard, at each stage choosing to trade-off between inner and outer level locks.

Two Level Lock Selection

When the system comprises just two layers, choosing an optimal (minimal, safe) lock distribution just requires the selection of a fixpoint in the lattice for the usage relation between the two layers, as we have just illustrated.

The *additive-at-top* approach starts with fine-grain locking and incrementally coarsens the locking. This corresponds to stepping up the fixpoint lattice, starting with no top-level lock, and all bottom-level locks satisfying the exclusion requirements there. At each step, a restricted set of choices is offered, corresponding to the covering relation for the current fixpoint. As a selection is made, we step up the lattice, adding locks to the top-level and removing locks at the bottom, corresponding to the labels on the selected lattice edge. At every step we maintain an optimal lock distribution; so the process may be stopped whenever the desired degree of lock coarsening is attained. The *additive-at-bottom* is the dual approach, starting with full coarse-grain locking at the top, and incrementally selecting locks to add at the bottom level.

Example 4. Consider the add-to-top approach with Example 2. We illustrate the lock selection process with the available choices at each step shown in Table 4. The actual selections, shown in bold give rise to the path in the lattice as in Figure 8.

Step	Lock A	Lock B	Choices	Selection
0	–	$1 \times \bar{2} \times 3$	$ab \times c$ $\overline{ab}$ $a \times \bar{c}$	 $\mathbf{a \times \bar{c}}$
1	$a \times \bar{c} \vert b$	$13 \times \bar{2}$	$b \times c$ $\overline{ab}$	 $\overline{\mathbf{ab}}$
2	$\bar{a} \times \bar{b}\bar{c}$	13×2	$b \times c$	$\mathbf{b \times c}$
3	$\overline{abc}$	–	–	–

Table 4: Lock Choices for Example 4

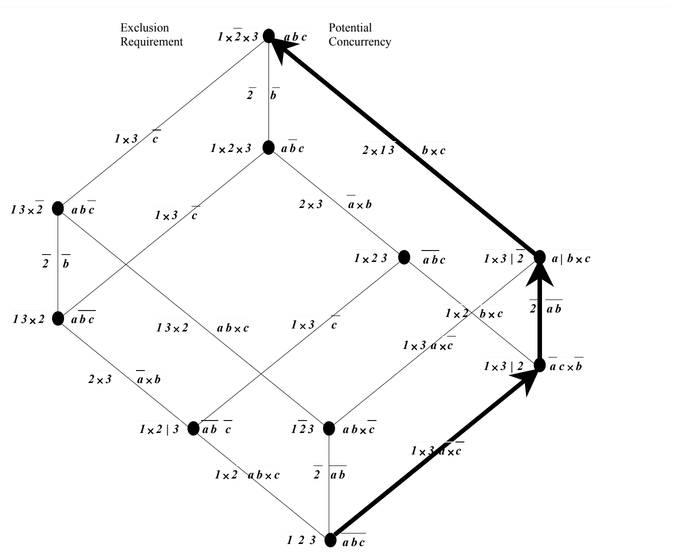

Figure 8: Selection Path in Example 4

When a composite object has more than one sub-object, we can decompose the *uses* relation and exclusion requirements correspondingly. This assists with reducing the complexity of the calculation of the fixpoints, but we must take all children into account simultaneously – lock choices for one child can affect those for the other. So we need to work with the fixpoint lattice between layers, and not between pairwise combinations of parent-child objects.

Example 5. Consider the combination of Example 2 and 3, where object A has two sub-objects B and C with given usage dependencies and exclusion requirements. The fixpoint lattice for the $A - B$ pair has 10 points, and 6 points for the $A - C$ pair. It turns out that the fixpoint lattice for the two layer system, $A - BC$ has 20 points. However the maximum path length in the lattice remains at 6, so we can handle the lock selection for the layer in the same number of steps as for each object individually.

With an add-at-top strategy, the path shown in Example 4 is also a path in the lattice of Example 5. However the choice $\overline{ab}$ at Example 4 Step 0 is now refined (because of the choice for Example 3) to just $\bar{b}$. With that choice, another possible path is $\bar{b}, a\times b, \bar{a}, b\times c, a\times c, \bar{c}$. The first four steps correspond to a path (Figure 7)(b).

Multi-level Lock Selection

When there are more than two layers of sub-objects in a composite object, there are more approaches for lock selection. We now describe the simplest of these, by adopting our two level strategies appropriately. All of these strategies guarantee: no objects will be re-visited once their locks are selected; no redundant locking will be produced; and any optimal lock distribution is achievable.

First we consider *subtractive* strategies: we work, level by level, removing locks from the current level and compensating by adding locks at the next level to re-establish safety. Thus a subtractive approach incrementally selects locks to migrate from level to level. This can be done either *top-down* or *bottom-up*. For example, the *subtractive top-down* approach starts with coarse grain locks at the top-level with no internal locks. We then offer lock selections, moving locks to the next layer down (just as for *add-at-bottom* strategy for the two-layer system). Then we move top-down, repeating this strategy at the second and third levels. And so on. Similarly, the *subtractive bottom-up* process starts with fine grain locks at the bottom level, and repeatedly applies the two-level *add-at-top* process to successive pairs of levels.

Next we consider *additive* strategies: again we work, level by level, selecting locks to add at each level, and compensating by removing locks at the extreme level (bottom when top-down, top when bottom-up). The *additive bottom-up* strategy, we start with coarse grain locks at the top-level. Then we apply the two-level add-to-bottom strategy between the top and bottom level; we repeat the add-to-bottom approach between the top and second bottom level; and so on. The idea is that we progressively add fine grain locks bottom-up, but always achieve safety and non-redundancy by removing duplicates locks at the top-most level. The *additive top-down* approach works dually. Whereas the subtractive approach migrates selected locks level by level, the additive approach is appropriate when a combination of outermost and innermost locks is wanted.

5 INCREMENTAL ALGORITHM FOR LOCK CHOICES

Even for relatively small object interfaces the size of the fixpoint lattice may be unmanageably large. For example, with n methods there are $n(n+1)/2$ pairs of methods, and up to $2^{n(n+1)/2}$ fixpoints in the Galois connection between the object and its internals. We therefore consider localised algorithms for navigating the fixpoint lattice. Given a fixpoint R/P, with $R = P^{\rhd}$ and $P = R^{\lhd}$, our task is to find the set of *covering increments* $\triangle/\nabla$ such that $(R \cup \triangle)/(P - \nabla)$ covers R/P in the fixpoint lattice.

First, some notation. For non-empty $X \subseteq R_I - R$, and $r_1, r_2 \in R_I - R$, define:

$$\begin{aligned} \nabla_R\, X &\mathrel{\widehat{=}} P \cap X^{\leftarrow} \\ r_1 \equiv_R r_2 &\mathrel{\widehat{=}} \nabla_R\, r_1 = \nabla_R\, r_2 \end{aligned}$$

The corresponding equivalence class is denoted by $[r]_R$.

Theorem For non-empty $\triangle \subseteq R_I - R$,
$\triangle/\nabla$ is a covering increment

$$\begin{aligned} \text{iff}\quad a)\quad & \nabla_R\, \triangle = \nabla \\ b)\quad & \forall r \in \triangle\ .\ [r]_R = \triangle \end{aligned}$$

See [7] and [8] for the proofs. As a consequence of this theorem we can restrict our search for covering increments to those generated by singletons, as in the following.

Algorithm:

1. For each $r \in R_I - R$, calculate $\nabla_R\, r[as\ R \cap r^{\leftarrow}]$.
2. Partition $R_I - R$ according to $\equiv_R$.
3. Output equivalence classes $[r]_R$ with minimal $\nabla_R\, [r]_R$.

An efficient implementation of this algorithm can exploit the partitioning of equivalence classes to eagerly eliminate non-minimal candidates. Another optimisation is to cache the values of $\nabla_R\, r$ and use these to calculate the values for the next fixpoint up the lattice:

$$\nabla_{R\cup\triangle}\, r = \nabla_R\, r - \nabla$$

Example 6: Consider Example 4 after one step where $R = 1 \times 3|2, P = \bar{b} \times \bar{a}c$. We find in Step 1 that each $\nabla_R\, r$ is distinct as shown in Table 5. So, the set of exclusion pairs is partitioned into singletons in Step 2. Then in Step 3, we find that $[2 \times 3]$ is non-minimal, because $b \times c \subseteq b \times \bar{a}c$. At the next fixpoint $1 \times 23/\overline{ab}c$ we find the partitions $[\bar{2}]$ and $[2 \times 3]$, but now $[\bar{2}]$ is non-minimal, because $b \times \bar{a} \subseteq \overline{ab}$.

$r \in R_I - R$	$r^{\leftarrow}$	Step 1: $\nabla_R\ r$	Step 2
1×2	$ab \times c$	$b \times c$	$[1 \times 2]$
$\bar{2}$	$\overline{ab}$	$\overline{ab}$	$[\bar{2}]$
2×3	$\bar{a} \times b \times c$	$b \times \bar{a}c$	$[2 \times 3]$

Table 5: Example 6

6 CONCLUSION

The object paradigm, since its emergence from the icy Oslo fjord, has been coupled with concurrency Objects are natural candidates for providing synchronisation mechanisms for concurrent activities. So many concurrent programming languages and models have emerged in the past three decades. We do not attempt a full survey here—see for instance Briot et al [9] or Philippsen [10] for comprehensive surveys of systems and approaches that integrate concurrency and object-oriented languages. Little of this work has addressed issues of lock distribution and how to select amongst alternative safe locking strategies.

There have been recent advances in using static analysis to instrument code with locks [11]. This work requires the programmer to annotate code blocks as being atomic and is only concerned with simple mutexes. Their lock allocation guarantees the required atomicity without risk of deadlock. Although they allow nesting of locks, there is less emphasis in this work on establishing hierarchical structure, though clearly the imposition of lock ordering implies that such structure is implicit. In our scheme, the hierarchical structure is explicit. The best of our knowledge, noone has considered the problem of lock distribution for a hierarchical model with arbitrary pair-wise exclusions, as we do.

Further work needs to be done to make our approach practical. To date, we have implemented and demonstrated the viability of a general purpose exclusion lock which can be configured to exactly meet a specific locking requirement [4], [12]. We have also prototyped the algorithm outlined in Section 5 in Haskell. What remains to be done is to provide a programming language based solution to the automatic derivation of the tree-structured *uses* relation. In this regard, we intend to pursue both development-time models, relying on object ownership types, and deployment-time techniques, relying on static analysis techniques.

This paper has considered the problem of lock selection for composite object systems, with known internal exclusion requirements and external potential for concurrent activity. We have identified strategies for selecting optimal lock distributions, exploiting fixpoint lattice of a Galois connection between the internal exclusion requirements and external concurrency environment. To reduce the difficulty of the lock selection process for programmers, we have presented a novel algorithm for local computation of the covering relation for the lattice. A wizard can use this algorithm to intelligently guide a programmer through the lock selection process.

REFERENCES

[1] Noble, J., Holmes, D., Potter, J.: Exclusion for composite objects. In: Proceedings of the 15th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), New York, NY, USA, ACM Press (2000) 13–28

[2] Shanneb, A., Potter, J.: Flexible exclusion control for composite objects. In: ACSC '05: Proceedings of the 28th Australasian Conference on Computer Science, Darlinghurst, Australia, Australian Computer Society, Inc. (2005) 277–286

[3] Shanneb, A., Potter, J., Noble, J.: Exclusion requirements and potential concurrency for composite objects. Sci. Comput. Program. **58** (2005) 344–365

[4] Potter, J., Shanneb, A., Yu, E.: 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)

[5] Shanneb, A., Potter, J.: Lock selection made easy. In: ASWEC '06: Proceedings of the 17th Australian Conference on Software Engineering., Sydney, Australia, IEEE Computer Society (2006) 341–350

[6] Davey, B.A., Priestley, H.A.: Introduction to Lattice and Order. 2 edn. Cambridge University Press (2002)

[7] Shanneb, A., Potter, J.: A Galois connection in composite objects. Technical Report -UNSW-CSE-TR-604, Programming Languages and Compilers Group, School of Computer Science and Engineering, University of New South Wales, Sydney, Australia (2006)

[8] Shanneb, A., Potter, J.: Lock selection in practice. Technical Report -UNSW-CSE-TR-605, Programming Languages and Compilers Group, School of Computer Science and Engineering, University of New South Wales, Sydney, Australia (2006)

[9] Briot, J.P., Guerraoui, R., Lohr, K.P.: Concurrency and distribution in object-oriented programming. ACM Computing Surveys **30** (1998) 291–329

[10] Philippsen, M.: A survey of concurrent object-oriented languages. Concurrency – Practice and Experience **12** (2000) 917–980

[11] Emmi, M., Fischer, J.S., Jhala, R., Majumdar, R.: Lock allocation. In: POPL '07: Proceedings of the 34th Symposium on Principles of Programming Languages, New York, NY, USA, ACM Press (2007) 291–296

[12] Potter, J., Shanneb, A., Yu, E.: Demonstrating the effectiveness of exclusion control for components. In: ASWEC '05: Proceedings of the 16th Australian Conference on Software Engineering, Brisbane, Australia, IEEE Computer Society (2005) 344–353

ABOUT THE AUTHORS

John Potter is currently with the Programming and Compilers Research Group at University of New South Wales, Sydney. He can be reached via potter@cse.unsw.edu.au. His research interests include ownership type systems, programmatic access control, and concurrency for objects.
Abdelsalam Shanneb recently completed his PhD studies at UNSW. His research is focused on concurrency control for object systems.