

SOFTWARE VERIFICATION RESEARCH CENTRE
DEPARTMENT OF COMPUTER SCIENCE
THE UNIVERSITY OF QUEENSLAND

Queensland 4072
Australia

TECHNICAL REPORT

No. 94-28

**A Methodology for Behavioural
Retrieval from Class Libraries**

Steven Atkinson
Roger Duke

September 1994

Phone: +61 7 3365 1003

Fax: +61 7 3365 1533

Note: Most SVRC technical reports are available via anonymous ftp, from `ftp.cs.uq.edu.au` in the directory `/pub/SVRC/techreports`.

A Methodology for Behavioural Retrieval from Class Libraries

Steven Atkinson *Roger Duke*

Software Verification Research Centre
Department of Computer Science
University of Queensland

Abstract

The central problem associated with libraries of reusable software components such as classes is that of retrieval: how to find in the library those classes that can be used in the construction of a particular application. At the heart of class retrieval is the issue of behaviour: to decide if a library class is appropriate for a particular application it is necessary to know how objects of that class behave, i.e. how they react when sent messages by their environment. In this paper a methodology for searching libraries to find classes whose objects behave in some specified manner is defined. In general, it will not be possible to find a class whose objects behave exactly as required. In this case the methodology finds classes whose objects best approximate the specified behaviour in a precisely defined sense. The aim is to retrieve classes that can be easily modified to define objects that behave exactly as required.

Keywords: class libraries, reuse, behavioural compatibility.

1 Introduction

It has been an article of faith for some time that comprehensive libraries of reusable software components will transform and mature the practice of software engineering [4, 15]. Furthermore, object orientation has been seen as the technology most likely to achieve this through the construction of class libraries[5, 6]. In practice, however, the problems associated with software reuse are formidable despite considerable recent effort[1, 12].

The central problem associated with libraries of reusable software components is that of retrieval. A library is of little use unless it can be easily searched and suitable components retrieved. Until recently, most effort in solving the retrieval problem has gone into software classification, with emphasis upon domain-specific aspects such as the signature of operations[16], and the determination of static metrics that can be used when searching and retrieving[8, 11].

However, it is the dynamic behaviour of objects that is of the most concern when building applications[3, 9]. At heart, the problem of retrieval is to find those classes in the library whose objects behave in some specified way when sent messages by their environment. It is this behavioural view of retrieval that we discuss. In Section 2 a class is modelled in terms of the way objects of that class respond to sequences of messages.

In general, it will not be possible to find a class in the library whose objects behave exactly as required. In this case the retrieval problem becomes that of finding the class (or classes) which are best approximations. In Section 3 a partial order is constructed to compare the behaviour of different classes. This partial order forms a lattice which is used in Section 4 to precisely define a notion of best approximation in finding the most suitable class (or classes, as the process does not always determine a uniquely best class). The classes so retrieved are those which are likely to be most easily modified to define objects that behave exactly as required. The methodology is illustrated in Section 5 with a detailed example, while issues to do with its practical application are discussed in Section 6.

The principle of behavioural retrieval from software libraries has been suggested by Podgurski and Pierce[9]; the idea of using a lattice structure as the basis of behavioural retrieval has been suggested by Mili[7]. In our paper we combine this work and extend it in three directions. Firstly, an abstract model of classes and class behaviour is defined, laying the foundation for a general theory of behavioural retrieval, independent of any particular language formalism. In particular, a notion of approximate behaviour is made precise. Secondly, a notion of failure is included in the behavioural model, and the idea that failures can be removed is incorporated into the underlying partial order between behaviours. Thirdly, a prescription for

behavioural retrieval involving the *join* and *meet* lattice operations is given.

Throughout this paper, the Z notation is used to specify concepts. In the main, Z is much like standard mathematical notation; readers unfamiliar with the notation may wish to consult[10, 14].

2 An Abstract Class Model

In this section an abstract model of class behaviour is formulated in terms of the response of objects of a class to sequences of messages.

Let *Name* denote the set of all possible names for operations (methods), and let *Val* denote the set of all possible input and output values associated with operations. The set *Val* may in practice include structured input and output values, but treatment of such details is beyond the scope of this paper. An object can be sent a message, which is a request to perform a named operation with some particular input; hence define

$$Msg == Name \times Val.$$

When an object is sent a message, it either succeeds in executing the named operation with the given input, and as a consequence produces some output, i.e. some value in *Val*, or the message is rejected (i.e. blocked). In this model it is supposed that the successful execution of an operation always results in some output in *Val*, even if only a message to the effect that the operation has succeeded.

In practice, there are various reasons why a message may be rejected. The message name may not be the name of an operation of the object, or the supplied input may not satisfy the operation's preconditions, or even that performing the operation would lead to non-termination. The precise reasons will not be of concern at this level of abstraction.

In general, an object can be sent not just one message, but a program consisting of a finite sequence of messages; hence define

$$Prog == seq\ Msg.$$

When sent a program an object will respond by executing each message in the sequence and producing a corresponding output sequence (unless it rejects a message, in which case the execution ceases at that stage). The overall response to the program is therefore some sequence of outputs in *Val*; hence define

$$Output == seq\ Val.$$

A class can be envisaged as a collection of objects, where each object has a common set of operations and responds in a similar way to a given program. This view is captured abstractly by the schema

| <i>Class</i> |
|--|
| $operations : \mathbb{P} Name$ $msgs : \mathbb{P} Msg$ $response : Prog \leftrightarrow Output$ |
| $\forall (name, val) : msgs \bullet name \in operations$ $dom response = \{pg : Prog \mid ran pg \subseteq msgs\}$ $\forall (pg, out) : response \bullet \#out \leq \#pg$ $\forall pg : dom response; n : \mathbb{N} \bullet$ $\quad \{t : Output \mid ((1..n) \triangleleft pg) \underline{response} t\}$ $\quad = \{t : Output \mid pg \underline{response} t \bullet (1..n) \triangleleft t\}$ |

In this schema, *operations* denotes the set of names of operations that any object in the class can undergo. The set *msgs* denotes the set of messages that are understood by the objects (note that an understood message may still be rejected). The first conjunct in the predicate of the schema ensures that the name part of any message in *msgs* is the name of an operation of the objects (i.e. a name in the set *operations*).

The attribute *response* records the way objects in the class respond to programs. Because of non-determinism, there may be several different possible responses to a given program; hence *response* is modelled as a relation rather than as a function.

If $pg \underline{response} out$ for program *pg* and output *out*, each message in the sequence *pg* must be in the set *msgs*, and the number of output values (i.e. $\#out$, the length of the sequence *out*) cannot exceed the number of messages in *pg* (i.e. $\#pg$, the length of the sequence *pg*). In fact, $\#out = \#pg$ except in the case when one of the messages is rejected, in which case the program is blocked at that point and no further output is produced. As a special case, notice that the empty program consisting of no messages conforms to the protocol of *msgs*; in this case the response will be the empty output, i.e. the empty sequence of values.

The final conjunct of the predicate of the schema *Class* captures the idea that if a program is extended (or shortened) by appending (or removing) messages, the expected output responses will each be an extension (or shortening) of the original output responses. This predicate ensures that the set of all possible outputs in response to the first *n* messages of a program *pg* (i.e. in response to the program $(1..n) \triangleleft pg$) is the set obtained by restricting the outputs of *pg* to their first *n* values.

To complete our model of classes, it only remains to specify the effect of executing a program on objects of a class. The relation *response* in the schema *Class* defines this effect for programs conforming to the set *msgs*, but in general an object could be sent messages not in the set *msgs*, e.g. a message requesting the execution of an operation not known to the objects. We model this case by supposing such messages are rejected. Hence we define

$$\begin{array}{|l}
\hline
execute : Prog \rightarrow Class \rightarrow \mathbb{P} Output \\
\hline
\forall prog : Prog; cl : Class; out : Output \bullet \\
\quad out \in execute(prog)(cl) \Leftrightarrow \\
\quad \exists pg : \text{dom } cl.response \bullet \\
\quad \quad pg \subseteq prog \\
\quad \quad pg \subset prog \Rightarrow prog(\#pg + 1) \notin cl.msgs \\
\quad \quad pg \underline{cl.response} out
\end{array}$$

The predicate of the function *execute* ensures that the result of executing a program *prog* on a class *cl* is the set of outputs given by the relation *response* of *cl* to the program *pg* defined as the largest possible prefix of *prog* all of whose messages are in the set *cl.msgs*. (If the first message in *prog* is not in *cl.msgs*, the program *pg* will be the empty sequence.)

3 Comparing Class Behaviours

Behavioural retrieval of classes from a class library critically depends upon having a sound basis for behavioural comparison; otherwise library retrieval operations would not select the most appropriate classes. In this section, class behaviour is defined and used in the construction of a partial order over class behaviours which assists behavioural comparison. This partial order is a lattice, and the definitions of its join and meet operations are given here and used to define a best-approximation class retrieval operation in Section 4.

3.1 Class Behaviour

Throughout the paper, class behaviour is understood to be with respect to some given program. In effect, each program determines a context in which the behaviour of a class is exhibited. The behaviour of class *c* for a given program *p* is derived from *execute(p)(c)* by removing outputs which are extensions of other outputs. Thus, we define behaviour as the set of guaranteed responses to a program. Behaviour sets in general satisfy:

$$Behaviour == \{O : \mathbb{P} Output \mid \forall o : O \bullet \nexists x : O \bullet x \subset o\}$$

i.e. a behaviour *b* : *Behaviour* is a set of sequences where each sequence has no prefix in the set. In order to obtain a behaviour from a set of sequences, we apply the following filter function:

$$\begin{array}{|l}
\hline
\downarrow : \mathbb{P} Output \rightarrow Behaviour \\
\hline
\forall O : \mathbb{P} Output \bullet \\
\quad \downarrow O = \{o : O \mid \nexists x : O \bullet x \subset o\}
\end{array}$$

This function extracts the minimal sequences from a set of outputs. It is important to note that this filter function is not one-to-one; there are classes whose outputs in response to a program are different, but whose filtered behaviour is the same. The classes whose behaviour is identical with respect to a given program are said to be *behaviourally equivalent*. It will be seen in Section 4.2 that if a class is returned as a result of a retrieval, all other behaviourally equivalent classes are also returned.

3.2 The Behavioural Projection of a Class Library

During behavioural retrieval in Section 4, the behaviour of each class in the library is compared with the desired behaviour (input by the library user). The collection containing the behaviour of each class in the library under a program p is called the *behavioural projection* of the library under p , and is defined by

$$\left| \begin{array}{l} \pi_{execute(p)} : \mathbb{P} \textit{Class} \rightarrow \mathbb{P} \textit{Behaviour} \\ \hline \forall L : \mathbb{P} \textit{Class} \bullet \\ \pi_{execute(p)}(L) = \{c : L \bullet \downarrow(execute(p)(c))\} \end{array} \right.$$

The calculation of the behavioural projection of a class library involves executing the program p on each class in the library and filtering each class' responses to get a set of behaviours, one behaviour for each class in the library. In practice, the process of executing a program p on a class in the library to obtain all possible responses can be done by sampling actual executions of the program on objects of the class, or by automated and human reasoning based on the class text. The calculation of the behavioural projection is very computationally expensive, but such efficiency concerns are not within the scope of the paper.

3.3 The Partial Order over Behaviours

To facilitate the comparison of class behaviours, a partial order over behaviours is defined.

$$\left| \begin{array}{l} _ \sqsubseteq _ : \textit{Behaviour} \leftrightarrow \textit{Behaviour} \\ \hline \forall B_1, B_2 : \textit{Behaviour} \bullet \\ B_1 \sqsubseteq B_2 \Leftrightarrow \\ \forall b : B_2 \bullet \exists b' : B_1 \bullet b' \subseteq b \end{array} \right.$$

That is, behaviours B_1 and B_2 are related by \sqsubseteq if for all sequences in B_2 , either the sequence or a prefix exists in the set B_1 . There are two reasons why this particular partial order over behaviours has been chosen. Firstly, this definition allows for the removal of non-determinism as one travels higher up the partial order, since it is

possible that some sequences in B_1 are not in B_2 . Secondly, the definition allows for removal of rejections as one travels up the partial order by allowing sequences in B_2 to be extensions of sequences in B_1 . The proof that \sqsubseteq defines a partial order is straightforward.

Theorem 1

The relation \sqsubseteq is a partial order.

Proof

This proof is given in Appendix A.

3.4 The Lattice of Behaviours

The partial order \sqsubseteq on behaviours defines a lattice whose join and meet operations are defined in this section. Given the behaviours of two classes, each of these operations form a new behaviour that is related to the two given behaviours by the partial order \sqsubseteq . These new behaviours do not necessarily correspond to any class in the class library; nonetheless, these behaviours are useful in the retrieval process. Their precise use will be given in Section 4.

3.4.1 The Meet of Behaviours

We define the meet of two class behaviours by

$$\left| \begin{array}{l} _ \sqcap _ : Behaviour \times Behaviour \rightarrow Behaviour \\ \hline \forall B_1, B_2 : Behaviour \bullet \\ B_1 \sqcap B_2 = \downarrow(B_1 \cup B_2) \end{array} \right.$$

That is, the meet of two behaviours B_1 and B_2 is the union of their behaviours with extensions filtered out. It is straightforward to show that the meet operator above does define the greatest lower bound with respect to the \sqsubseteq partial order.

Theorem 2

The meet operator \sqcap defines the greatest lower bound operation on the lattice defined by \sqsubseteq .

Proof

This proof is given in Appendix B.

3.4.2 The Join of Behaviours

The join of class behaviours is not always defined because two classes may respond to a program p with completely unrelated output. Two outputs are said to be *comparable* if one is a prefix or extension of the other, i.e.

$$\frac{}{\text{comparable} : \text{Output} \leftrightarrow \text{Output}} \quad \left| \begin{array}{l} \forall o_1, o_2 : \text{Output} \bullet \\ o_1 \text{ comparable } o_2 \Leftrightarrow o_1 \sqsubseteq o_2 \vee o_2 \sqsubseteq o_1 \end{array} \right.$$

The function *comparables* extracts the comparable sequences from a pair of behaviours:

$$\frac{}{\text{comparables} : \text{Behaviour} \times \text{Behaviour} \rightarrow \mathbb{P} \text{Output}} \quad \left| \begin{array}{l} \forall B_1, B_2 : \text{Behaviour} \bullet \\ \text{comparables}(B_1, B_2) = \\ \{b : B_1 \mid \exists b' : B_2 \bullet b \text{ comparable } b'\} \cup \\ \{b : B_2 \mid \exists b' : B_1 \bullet b \text{ comparable } b'\} \end{array} \right.$$

Notice that for behaviours which contain no comparable sequences, the *comparables* function returns the empty set. The *comparables* function can be viewed as defining a weak intersection on behaviours, where the test for intersection has been relaxed from strict equality of sequences to equality up to comparability.

The join of two behaviours will exist (and the behaviours are said to be *join compatible*) if and only if the set of comparable sequences is not empty. Then the join of two behaviours is given by

$$\frac{}{_ \sqcup _ : \text{Behaviour} \times \text{Behaviour} \rightarrow \text{Behaviour}} \quad \left| \begin{array}{l} \lambda B_1 : \text{Behaviour} \bullet \\ \lambda B_2 : \text{Behaviour} \mid \text{comparables}(B_1, B_2) \neq \emptyset \bullet \\ \uparrow \text{comparables}(B_1, B_2) \end{array} \right.$$

where the \uparrow filter function is defined as:

$$\frac{}{\uparrow : \mathbb{P} \text{Output} \rightarrow \text{Behaviour}} \quad \left| \begin{array}{l} \forall O : \mathbb{P} \text{Output} \bullet \\ \uparrow O = \{o : O \mid \nexists e : O \bullet o \subset e\} \end{array} \right.$$

This filter function extracts the maximal sequences from a set of outputs. The join of two behaviours B_1 and B_2 is the set of comparable sequences of B_1 and B_2 (which must be non-empty for the join to be defined) with prefixes filtered out. The proof that \sqcup defines the least upper bound with respect to the \sqsubseteq partial order is given below.

Theorem 3

The join operator \sqcup defines the least upper bound operation on the lattice defined by \sqsubseteq .

Proof

This proof is given in Appendix C.

4 Behavioural Retrieval

In this section a methodology for searching class libraries to find classes whose objects behave in some specified manner is given. If a class exists whose behaviour exactly matches the desired behaviour, the methodology yields this class, otherwise it yields those classes which are likely to be easily modified to behave as desired. The methodology is described in three phases.

4.1 The First Phase: Behavioural Projection

The user of a class library \mathcal{L} needs to construct a program p and specify the behaviour being sought. Hence we take as inputs to the retrieval process the following:

- a class library $\mathcal{L} : \mathbb{P} \textit{Class}$;
- a testing program $p : \textit{Prog}$;
- a desired behaviour $b : \textit{Behaviour}$.

The first step is to construct the *behavioural projection* $\mathcal{B} = \pi_{\textit{execute}(p)}(\mathcal{L})$ of \mathcal{L} under the program p . The collection \mathcal{B} is embedded in the lattice structure determined by the partial order \sqsubseteq between behaviours, although \mathcal{B} itself is not necessarily closed under the join and meet operations.

4.2 The Second Phase: Finding Close Behaviours

Having obtained the collection \mathcal{B} , one may now use the partial order \sqsubseteq to determine which behaviours in \mathcal{B} are closest (in the sense of easily modifiable) to the desired behaviour b . The behaviours being sought are those behaviours in \mathcal{B} that maximize the number of sequences in common with b , while minimizing the number of sequences not in common with b . We consider two strategies.

Strategy 1

- (a) Locate those behaviours in \mathcal{B} which maximize the number of sequences in common with b ;
- (b) Of those, find the behaviours that minimize the number of sequences not in common with b .

Strategy 2

- (a) Locate the behaviours in \mathcal{B} which minimize the number of sequences that are not in b ;
- (b) Of those, find the behaviours that maximize the number of sequences in common with b .

As these strategies will not in general select identical sets of behaviours in \mathcal{B} , both are employed in order to maximize the recall (defined to be ratio of the number of relevant retrievals to the number of relevant solutions[13]) of the retrieval process. Each of these strategies is now formally described using the lattice of behaviours defined in Section 3.

4.2.1 Strategy 1 – Maximize, then Minimize

A measure of the number of output sequences a behaviour $b' \in \mathcal{B}$ has in common with b can be determined by computing the join $b' \sqcup b$. If one performs this computation for all join compatible behaviours $b' \in \mathcal{B}$ one obtains a collection of behaviours which we call the *join projection* of \mathcal{B} under the behaviour b . Hence, define

$$\left| \begin{array}{l} \pi_{\sqcup b} : \mathbb{P} \text{ Behaviour} \rightarrow \mathbb{P} \text{ Behaviour} \\ \hline \forall B : \mathbb{P} \text{ Behaviour} \bullet \pi_{\sqcup b}(B) = \{b' : B \mid \text{comparables}(b', b) \neq \emptyset \bullet b' \sqcup b\} \end{array} \right.$$

This function is not one-to-one: behaviours are effectively collapsed into equivalence classes whose “distance” from behaviour b is the same.

The minimal elements under \sqsubseteq in the join projection $\pi_{\sqcup b}(\mathcal{B})$ are images under $\pi_{\sqcup b}$ of those behaviours $b' \in \mathcal{B}$ that contain the most number of sequences in common with b . The set of such behaviours $b' \in \mathcal{B}$ satisfy the condition of Strategy 1(a). In order to satisfy Strategy 1(b), take the maximal behaviours under \sqsubseteq of all such behaviours b' . In effect, moving up the partial order to find the maximals of the set maximizing the common responses minimizes the number of responses that are not in b .

To be precise, define the function \min_{\sqsubseteq} (and \max_{\sqsubseteq}) to extract the minimal (or maximal) behaviours under \sqsubseteq from a set of behaviours as:

$$\begin{array}{|l}
\hline
min_{\sqsubseteq}, max_{\sqsubseteq} : \mathbb{P} Behaviour \rightarrow \mathbb{P} Behaviour \\
\hline
\forall B : \mathbb{P} Behaviour \bullet \\
min_{\sqsubseteq}(B) = \{b : B \mid \nexists b' : B \bullet b' \sqsubseteq b\} \\
max_{\sqsubseteq}(B) = \{b : B \mid \nexists b' : B \bullet b \sqsubseteq b'\} \\
\hline
\end{array}$$

The behaviours in \mathcal{B} satisfying the condition of Strategy 1 are those obtained by the following steps:

Step 1. Calculate $\mathcal{B}_{min \pi} = min_{\sqsubseteq}(\pi_{\sqcup b}(\mathcal{B}))$

Step 2. Calculate $\mathcal{B}_{strategy1} = max_{\sqsubseteq}\{b' : \mathcal{B} \mid comparables(b, b') \neq \emptyset \wedge b' \sqcup b \in \mathcal{B}_{min \pi}\}$

A summary of Strategy 1 is given by the definition of the following function which selects the behaviours in $\mathcal{B}_{strategy1}$ from \mathcal{B} according to the above algorithm.

$$\begin{array}{|l}
\hline
strategy1 : \mathbb{P} Behaviour \rightarrow \mathbb{P} Behaviour \\
\hline
\forall B : \mathbb{P} Behaviour \bullet \\
strategy1(B) = \\
max_{\sqsubseteq}\{b' : B \mid comparables(b, b') \neq \emptyset \wedge b \sqcup b' \in min_{\sqsubseteq}(\pi_{\sqcup b}(\mathcal{B}))\} \\
\hline
\end{array}$$

That is, Strategy 1 applied to \mathcal{B} yields the behaviours $strategy1(\mathcal{B})$.

4.2.2 Strategy 2 – Minimize, then Maximize

As the formal description of Strategy 2 is similar to that presented above for Strategy 1, only the salient points are mentioned. A measure of the number of output sequences a behaviour $b' \in \mathcal{B}$ does not have in common with b can be determined by computing the meet $b' \sqcap b$. The *meet projection* of \mathcal{B} under behaviour b is defined as:

$$\begin{array}{|l}
\hline
\pi_{\sqcap b} : \mathbb{P} Behaviour \rightarrow \mathbb{P} Behaviour \\
\hline
\forall B : \mathbb{P} Behaviour \bullet \pi_{\sqcap b}(B) = \{b' : B \bullet b' \sqcap b\} \\
\hline
\end{array}$$

The maximal elements under \sqsubseteq in the meet projection $\pi_{\sqcap b}(\mathcal{B})$ are images under $\pi_{\sqcap b}$ of those behaviours $b' \in \mathcal{B}$ that contain the fewest number of sequences not in common with b . The set of such behaviours $b' \in \mathcal{B}$ satisfy the condition of Strategy 2(a). In order to satisfy Strategy 2(b), take the minimal behaviours under \sqsubseteq of all such behaviours b' . In effect, moving down the partial order to find the minimals of the set minimizing the responses not in common maximizes the number of responses that may be in b .

Thus the behaviours in \mathcal{B} satisfying Strategy 2 are those obtained by performing the following steps:

Step 1. Calculate $\mathcal{B}_{max \pi} = max_{\sqsubseteq}(\pi_{\sqcap b}(\mathcal{B}))$

Step 2. Calculate $\mathcal{B}_{strategy2} = min_{\sqsubseteq}\{b' : \mathcal{B} \mid b' \sqcap b \in \mathcal{B}_{max \pi}\}$

A summary of Strategy 2 is given by the definition of the following function which selects the behaviours in $\mathcal{B}_{strategy2}$ from \mathcal{B} according to the above algorithm.

$$\left| \begin{array}{l} strategy2 : \mathbb{P} Behaviour \rightarrow \mathbb{P} Behaviour \\ \hline \forall B : \mathbb{P} Behaviour \bullet \\ \quad strategy2(B) = \\ \quad \quad min_{\sqsubseteq}\{b' : B \mid b \sqcap b' \in max_{\sqsubseteq}(\pi_{\sqcap b}(\mathcal{B}))\} \end{array} \right.$$

That is, Strategy 2 applied to \mathcal{B} yields the behaviours $strategy2(\mathcal{B})$.

4.3 The Third Phase: Retrieving the Classes

Together, Strategies 1 and 2 yield the behaviours in \mathcal{B} which are most easily modifiable to obtain the desired behaviour b . It now remains to retrieve those classes in \mathcal{L} which when sent the program p yield these behaviours. The following function retrieves those classes

$$\left| \begin{array}{l} behavioural_retrieval : \mathbb{P} Class \rightarrow \mathbb{P} Class \\ \hline \forall L : Class \bullet \\ \quad behavioural_retrieval(L) = \\ \quad \quad \{c : L \mid \downarrow(execute(p)(c)) \in \\ \quad \quad \quad (strategy1(\pi_{execute(p)}(L)) \cup strategy2(\pi_{execute(p)}(L)))\} \end{array} \right.$$

That is, the behavioural retrieval operation takes a set of classes (a class library), and returns a subset of those classes which are most easily modifiable to obtain a desired class, as determined by Strategies 1 and 2, the testing program p , and the desired behaviour b .

It can be verified that if there are classes in the library \mathcal{L} whose behaviour with respect to the program p is equal to the desired behaviour b , then these classes (and no others) will be retrieved by the methodology.

Theorem 4

The methodology is sound, i.e. if the set C of those classes in \mathcal{L} having a behaviour with respect to a program p identical to the desired behaviour b is non-empty, then

$$behavioural_retrieval(\mathcal{L}) = C$$

Proof

This proof is given in Appendix D.

5 An Example

This section provides an example to illustrate the method of behavioural retrieval established in Section 4. Suppose the library \mathcal{L} consists of 4 vending machine classes named A , B , C and D . Each vending machine has just one operation, $GetTreat$, which dispenses a single treat (either a *choc* or a *candy*) each time the environment inserts a coin. So for this section we take $Name = \{GetTreat\}$ and $Val = \{coin, choc, candy\}$. The machines behave in the following way:

- A first dispenses a chocolate, then a candy, and alternates between dispensing a chocolate then a candy as the operation $GetTreat$ is repeatedly performed;
- B dispenses a chocolate whenever $GetTreat$ is performed;
- C non-deterministically dispenses a chocolate or a candy whenever $GetTreat$ is performed, except that once a chocolate has been dispensed the operation $GetTreat$ is rejected and no further treat is dispensed;
- D non-deterministically dispenses a chocolate or a candy when $GetTreat$ is first performed, but then $GetTreat$ is always rejected and no further treat is dispensed.

Suppose we want a vending machine that first non-deterministically dispenses a chocolate or a candy, and subsequently alternates between dispensing a chocolate or candy as the operation $GetTreat$ (with *coin* as input) is repeatedly performed. Hence the required outcome of performing the program

$$p = \langle (GetTreat, coin), (GetTreat, coin) \rangle$$

is the behaviour

$$b = \{ \langle choc, candy \rangle, \langle candy, choc \rangle \}.$$

Suppose we search the library for the behaviour b relative to the program p . In this case the behavioural projection $\pi_{execute(p)}(\mathcal{L})$ is the set $\mathcal{B} = \{a', b', c', d'\}$ of behaviours where

$$\begin{aligned} a' &= execute(p)(A) = \{ \langle choc, candy \rangle \} \\ b' &= execute(p)(B) = \{ \langle choc, choc \rangle \} \\ c' &= execute(p)(C) = \{ \langle choc \rangle, \langle candy, choc \rangle, \langle candy, candy \rangle \} \\ d' &= execute(p)(D) = \{ \langle choc \rangle, \langle candy \rangle \} \end{aligned}$$

By Strategy 1, the join projection $\pi_{\sqcup b}(\mathcal{B})$ is the set $J = \{Y, Z\}$ where

$$\begin{aligned} Y &= a' \sqcup b = \{ \langle choc, candy \rangle \} \\ Z &= c' \sqcup b = d' \sqcup b = \{ \langle choc, candy \rangle, \langle candy, choc \rangle \}. \end{aligned}$$

(The behaviour b' is not join compatible with b .)

In this case $Z \sqsubseteq Y$, so $\mathcal{B}_{\min \pi} = \min_{\sqsubseteq}(J) = \{Z\}$. Then

$$\mathcal{B}_{\text{strategy1}} = \max_{\sqsubseteq}(\{c', d'\}) = \{c'\} \quad (\text{as } d' \sqsubseteq c').$$

By Strategy 2, the meet projection $\pi_{\cap b}(\mathcal{B})$ is the set $K = \{O, P, Q, R\}$ where

$$\begin{aligned} O &= a' \sqcap b = \{\langle \text{choc}, \text{candy} \rangle, \langle \text{candy}, \text{choc} \rangle\} \\ P &= b' \sqcap b = \{\langle \text{choc}, \text{candy} \rangle, \langle \text{candy}, \text{choc} \rangle, \langle \text{choc}, \text{choc} \rangle\} \\ Q &= c' \sqcap b = \{\langle \text{choc} \rangle, \langle \text{candy}, \text{choc} \rangle, \langle \text{candy}, \text{candy} \rangle\} \\ R &= d' \sqcap b = \{\langle \text{choc} \rangle, \langle \text{candy} \rangle\}. \end{aligned}$$

In this case $R \sqsubseteq Q \sqsubseteq P \sqsubseteq O$, so $\mathcal{B}_{\max \pi} = \max_{\sqsubseteq}(K) = \{O\}$. Then

$$\mathcal{B}_{\text{strategy2}} = \min_{\sqsubseteq}(\{a'\}) = \{a'\}.$$

The classes retrieved by behavioural retrieval are those which yield behaviours in $\mathcal{B}_{\text{strategy1}} \cup \mathcal{B}_{\text{strategy2}}$ which in this case is equal to $\{a', c'\}$; hence the set of classes retrieved is $\{A, C\}$. This is as we would expect, as $c' \sqsubseteq b \sqsubseteq a'$. Hence to obtain the required vending machine class, either take class A and modify it so that the behaviour $\langle \text{candy}, \text{choc} \rangle$ is added to the outcome when executing p , or take class C and modify it so that when executing p the blocked behaviour $\langle \text{choc} \rangle$ is extended to $\langle \text{choc}, \text{candy} \rangle$ and the behaviour $\langle \text{candy}, \text{candy} \rangle$ is no longer available.

As another example, suppose the library \mathcal{L} and program p (and hence the behavioural projection \mathcal{B}) are as above, but suppose we are now seeking a vending machine that non-deterministically dispenses a chocolate or a candy whenever GetTreat is performed. In this case the required behaviour with respect to the program p is

$$b = \{\langle \text{choc}, \text{candy} \rangle, \langle \text{choc}, \text{choc} \rangle, \langle \text{candy}, \text{candy} \rangle, \langle \text{candy}, \text{choc} \rangle\}.$$

By Strategy 1, the join projection $\pi_{\sqcup b}(\mathcal{B})$ is the set $L = \{V, W, X\}$ where

$$\begin{aligned} V &= a' \sqcup b = \{\langle \text{choc}, \text{candy} \rangle\} \\ W &= b' \sqcup b = \{\langle \text{choc}, \text{choc} \rangle\} \\ X &= c' \sqcup b = d' \sqcup b = \{\langle \text{choc}, \text{candy} \rangle, \langle \text{choc}, \text{choc} \rangle, \langle \text{candy}, \text{candy} \rangle, \langle \text{candy}, \text{choc} \rangle\}. \end{aligned}$$

In this case $X \sqsubseteq W$ and $X \sqsubseteq V$, so $\mathcal{B}_{\min \pi} = \min_{\sqsubseteq}(L) = \{X\}$. Then

$$\mathcal{B}_{\text{strategy1}} = \max_{\sqsubseteq}(\{c', d'\}) = \{c'\} \quad (\text{as } d' \sqsubseteq c' \text{ as before}).$$

By Strategy 2, the meet projection $\pi_{\cap b}(\mathcal{B})$ is the set $M = \{S, T, U\}$ where

$$\begin{aligned} S &= a' \sqcap b = b' \sqcap b = \{\langle \text{choc}, \text{candy} \rangle, \langle \text{choc}, \text{choc} \rangle, \langle \text{candy}, \text{candy} \rangle, \langle \text{candy}, \text{choc} \rangle\} \\ T &= c' \sqcap b = \{\langle \text{choc} \rangle, \langle \text{candy}, \text{choc} \rangle, \langle \text{candy}, \text{candy} \rangle\} \\ U &= d' \sqcap b = \{\langle \text{choc} \rangle, \langle \text{candy} \rangle\}. \end{aligned}$$

In this case $U \sqsubseteq T \sqsubseteq S$, so $\mathcal{B}_{max \pi} = max_{\sqsubseteq}(M) = \{S\}$. Then

$$\mathcal{B}_{strategy2} = min_{\sqsubseteq}(\{a', b'\}) = \{a', b'\}$$

(as the behaviours a' and b' are not related by \sqsubseteq).

The classes retrieved by behavioural retrieval are those which yield behaviours in $\mathcal{B}_{strategy1} \cup \mathcal{B}_{strategy2}$ which in this case is equal to $\{a', b', c'\}$; hence the set of classes retrieved is $\{A, B, C\}$. Notice that behaviour $a' \sqcap b'$ would be a better approximation to b than either a' or b' . Hence if there existed class operations in the library to create a class $A \sqcap B$, say, whose behaviour is the meet of the behaviours of A and B , retrieval could be sharpened, i.e. enhancing a library with operations for class composition can improve retrieval.

6 Future Work

It should be emphasised that this paper concentrates on the specification of a theoretical framework for behavioural retrieval, rather than giving optimized algorithms for a practical implementation. There are number of practical considerations when considering the implementation of the methodology. Among these are the questions of how to determine correspondences between names of operations in the program p and names of operations of library classes, how much of the burden of calculation of desired outputs in behaviour b should rest with the library user, and how to provide efficient methods of taking the behavioural join and meet projections. The existence of our theoretical framework provides a basis in which these practical considerations may be discussed.

There are a number of ways in which the methodology presented in Section 4 can be extended and applied. A simple extension allows the process to cater for a set of programs \mathcal{P} as input, rather than a single program p . Additionally, a scheme to return those classes which perform sub-tasks of the desired behaviour is needed, since more classes are relevant to solving the retrieval query at hand than those that solve it directly.

The relationship between instances of the abstract class model described in Section 2 and classes specified in object-oriented specification languages such as Object-Z[2] needs to be investigated, particularly to determine a method of deriving behaviour from formal specifications. The aim is to enable the methodology to be applied to a library of classes described by formal specifications.

The partial order \sqsubseteq was chosen because of its close relationship with the notion of refinement. The methodology developed using \sqsubseteq describes a notion of behavioural

affinity which is in fact quite distinct from the usual substitutability notion of refinement. There may exist other filters and partial orders over sets of outputs that describe separate yet equally valuable notions of affinity.

There is also a need for development of class composition operators which are capable of performing semi-automatic adaption of the library classes returned by this (or any other) methodology. The aim of this would be to apply class modification operators to the classes presented as close solutions.

The eventual goal of this research is to formally specify a class-based software reuse repository by integrating a number of retrieval methods with other repository operations.

7 Conclusion

By introducing an abstract model of classes we have been able to specify a methodology for searching a library of classes looking for specific behaviour. This methodology is sound in that if a class with the required behaviour exists it will be found; if no such class exists it will find the best approximation relative to a lattice structure induced by a partial order between behaviours.

Our methodology lays the necessary theoretical foundations upon which, we believe, practical tools for behavioural retrieval from class libraries can be built.

References

- [1] T. J. Biggerstaff and C. Richter. Reusability framework, assessment, and directions. *IEEE Software*, 4(2):41–49, March 1987.
- [2] R. Duke, P. King, G. Rose, and G. Smith. The Object-Z specification language: Version 1. Technical Report 91-1, Software Verification Research Centre, Dept. of Computer Science, Univ. of Queensland, Australia, 1991.
- [3] R. J. Hall. Generalized behavior-based retrieval. In *Proceedings of the 15th International Conference on Software Engineering*, pages 371–380, May 1993.
- [4] M. D. McIlroy. Mass produced software components. *Proceedings of the 1969 NATO Conference on Software Engineering*, 1969.
- [5] B. Meyer. Reusability: The case for object-oriented design. *IEEE Software*, 4(2):50–64, March 1987.

- [6] B. Meyer. *Reusable software - The base object-oriented component libraries*. Prentice Hall, ISE, Santa Barbara and SOL, Paris, 1994.
- [7] A. Mili, R. Mili, and R. Mittermeir. Storing and retrieving software components: A refinement based system. In *Proceedings of the 16th International Conference on Software Engineering*, pages 91–100. IEEE Computer Society Press, May 1994.
- [8] Xavier Pintado. Selection and exploration in an object-oriented environment: The affinity browser. Object management, Centre Universitaire d’Informatique, University of Geneva, July 1990.
- [9] A. Podgurski and L. Pierce. Behaviour sampling: A technique for automated retrieval of reusable components. In *Proceedings of the 14th International Conference on Software Engineering*, pages 349–360, 1992.
- [10] B. Potter, J. Sinclair, and D. Till. *An Introduction to Formal Specification and Z*. International Series in Computer Science. Prentice-Hall, 1990.
- [11] R. Prieto-Diaz. Implementing faceted classification for software reuse. *Commun. ACM*, 34(5):88–97, May 1991.
- [12] R. Prieto-Diaz. Status report – software reusability. *IEEE Software*, 10(3):61–66, May 1993.
- [13] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.
- [14] J.M. Spivey. *The Z Notation: A Reference Manual*. International Series in Computer Science. Prentice-Hall, 1989.
- [15] P. Wegner. Capital-intensive software technology. *IEEE Software*, 1(3), July 1984.
- [16] A. Zaremski and J. M. Wing. Signature matching: A key to reuse. Technical Report CMU-CS-93-151, Carnegie Mellon University, May 1993.

Appendix A

Lemma 1

The relation \sqsubseteq is transitive.

Proof

Suppose B_1, B_2, B_3 : *Behaviour* are behaviours satisfying:

$$B_1 \sqsubseteq B_2 \tag{1}$$

$$B_2 \sqsubseteq B_3 \tag{2}$$

Take any b_3 : B_3 .

$$\begin{aligned} & (2) \\ \equiv & \{ \text{definition of } \sqsubseteq \} \\ & \exists b_2 : B_2 \bullet b_2 \subseteq b_3 \\ \Rightarrow & \{ \text{by (1) and the definition of } \sqsubseteq \} \\ & \exists b_1 : B_1; b_2 : B_2 \bullet b_1 \subseteq b_2 \wedge b_2 \subseteq b_3 \\ \Rightarrow & \{ \text{transitivity of } \subseteq, \text{ simplification} \} \\ & \exists b_1 : B_1 \bullet b_1 \subseteq b_3 \end{aligned}$$

Hence

$$\begin{aligned} & \forall b_3 : B_3 \bullet \exists b_1 : B_1 \bullet b_1 \subseteq b_3 \\ \equiv & \{ \text{by definition of } \sqsubseteq \} \\ & B_1 \sqsubseteq B_3 \\ & \square \end{aligned}$$

Lemma 2

The relation \sqsubseteq is antisymmetric.

Proof

Suppose B_1, B_2 : *Behaviour* are behaviours satisfying:

$$B_1 \sqsubseteq B_2 \tag{1}$$

$$B_2 \sqsubseteq B_1 \tag{2}$$

Note the following property of behaviours:

$$\begin{aligned} & B \in \text{Behaviour} \\ \Rightarrow & \{ \text{within a behaviour, no sequence has a prefix} \} \\ & \forall x, y : B \bullet x \subseteq y \Rightarrow x = y \end{aligned} \tag{3}$$

Take any b_2 : B_2 .

$$\begin{aligned}
& (1) \\
\equiv & \{\text{by definition of } \sqsubseteq\} \\
& \exists b_1 : B_1 \bullet b_1 \subseteq b_2 \\
\Rightarrow & \{\text{by (2) and the definition of } \sqsubseteq\} \\
& \exists b_1 : B_1; b'_2 : B_2 \bullet b'_2 \subseteq b_1 \wedge b_1 \subseteq b_2 \\
\Rightarrow & \{\text{transitivity of } \subseteq\} \\
& \exists b_1 : B_1; b'_2 : B_2 \bullet \\
& \quad b'_2 \subseteq b_2 \wedge b'_2 \subseteq b_1 \subseteq b_2 \\
\Rightarrow & \{\text{by (3)}\} \\
& \exists b_1 : B_1; b'_2 : B_2 \bullet \\
& \quad b'_2 = b_2 \wedge b'_2 \subseteq b_1 \subseteq b_2 \\
\Rightarrow & \{\text{property of } \subseteq, \text{ simplification}\} \\
& \exists b_1 : B_1 \bullet b_1 = b_2
\end{aligned}$$

Hence

$$\begin{aligned}
& \forall b_2 : B_2 \bullet \exists b_1 : B_1 \bullet b_1 = b_2 \\
\equiv & \{\text{by definition of } \subseteq\} \\
& B_2 \subseteq B_1
\end{aligned}$$

By a symmetrical argument, $B_1 \subseteq B_2$, therefore

$$\begin{aligned}
& B_1 = B_2 \\
& \quad \square
\end{aligned}$$

Theorem 1

The relation \sqsubseteq is a partial order.

Proof

As \sqsubseteq is clearly reflexive, the result follows from Lemmas 1 and 2. \square

Appendix B

Lemma 3

The meet operator \sqcap constructs a lower bound.

Suppose B_1 and B_2 are behaviours. We are required to show:

$$\begin{aligned}
& (B_1 \sqcap B_2) \sqsubseteq B_1 \\
& (B_1 \sqcap B_2) \sqsubseteq B_2
\end{aligned}$$

Proof

Take any $b : B_1$.

$$\begin{aligned}
& b \in B_1 \\
\Rightarrow & \{\text{property of } \cup\} \\
& b \in B_1 \cup B_2 \\
\Rightarrow & \{\text{property of } \downarrow\} \\
& \exists b' : \downarrow(B_1 \cup B_2) \bullet b' \subseteq b \\
\Rightarrow & \{\text{by definition of } \sqcap\} \\
& \exists b' : (B_1 \sqcap B_2) \bullet b' \subseteq b
\end{aligned}$$

Hence

$$\begin{aligned}
& \forall b : B_1 \bullet \exists b' : (B_1 \sqcap B_2) \bullet b' \subseteq b \\
\equiv & \{\text{by definition of } \sqsubseteq\} \\
& (B_1 \sqcap B_2) \sqsubseteq B_1
\end{aligned}$$

By a symmetrical argument, $(B_1 \sqcap B_2) \sqsubseteq B_2$, therefore

$$\begin{aligned}
& (B_1 \sqcap B_2) \sqsubseteq B_1 \\
& (B_1 \sqcap B_2) \sqsubseteq B_2 \\
& \quad \square
\end{aligned}$$

Lemma 4

The meet operator \sqcap constructs a behaviour above any lower bound.

We are required to show that if B_1, B_2, B : *Behaviour* are behaviours satisfying:

$$B \sqsubseteq B_1 \tag{1}$$

$$B \sqsubseteq B_2 \tag{2}$$

then

$$B \sqsubseteq (B_1 \sqcap B_2)$$

Proof

Take any $b : (B_1 \sqcap B_2)$.

$$\begin{aligned}
& b \in (B_1 \sqcap B_2) \\
\equiv & \{\text{by definition of } \sqcap\} \\
& b \in \downarrow(B_1 \cup B_2) \\
\Rightarrow & \{\text{by definition of } \downarrow\} \\
& b \in B_1 \cup B_2
\end{aligned}$$

Performing case analysis on the type of b .

Case I: Suppose $b \in B_1$

$$\begin{aligned} & \text{(1)} \\ \Rightarrow & \{ \text{by definition of } \sqsubseteq \} \\ & \exists b' : B \bullet b' \subseteq b \end{aligned} \tag{3}$$

Case II: Suppose $b \in B_2$

$$\begin{aligned} & \text{(2)} \\ \Rightarrow & \{ \text{by definition of } \sqsubseteq \} \\ & \exists b' : B \bullet b' \subseteq b \end{aligned} \tag{4}$$

Hence,

$$\begin{aligned} & \text{(3)} \wedge \text{(4)} \\ \Rightarrow & \{ \text{case analysis above} \} \\ & \forall b : (B_1 \sqcap B_2) \bullet \exists b' : B \bullet b' \subseteq b \\ \equiv & \{ \text{by definition of } \sqsubseteq \} \\ & B \sqsubseteq (B_1 \sqcap B_2) \\ & \square \end{aligned}$$

Theorem 2

The meet operator \sqcap defines the greatest lower bound operation on the lattice defined by \sqsubseteq .

Proof

The result follows directly from Lemmas 3 and 4. □

Appendix C

Lemma 5

The join operator \sqcup constructs an upper bound.

Suppose B_1 and B_2 are join-compatible behaviours. We are required to show:

$$\begin{aligned} B_1 & \sqsubseteq (B_1 \sqcup B_2) \\ B_2 & \sqsubseteq (B_1 \sqcup B_2) \end{aligned}$$

Proof

Take any $b : (B_1 \sqcup B_2)$.

$$\begin{aligned}
& b \in (B_1 \sqcup B_2) \\
\Rightarrow & \{\text{by definition of } \sqcup\} \\
& b \in \uparrow(\text{comparables}(B_1, B_2)) \\
\Rightarrow & \{\text{by definition of } \uparrow\} \\
& b \in \text{comparables}(B_1, B_2) \\
& \wedge \nexists e : \text{comparables}(B_1, B_2) \bullet b \subseteq e \\
\Rightarrow & \{\text{by definition of } \text{comparables}\} \\
& b \in (\{b_1 : B_1 \mid \exists x : B_2 \bullet b_1 \underline{\text{comparable}} x\} \\
& \quad \cup \{b_2 : B_2 \mid \exists x : B_1 \bullet b_2 \underline{\text{comparable}} x\}) \\
& \wedge \nexists e : \text{comparables}(B_1, B_2) \bullet b \subseteq e
\end{aligned} \tag{3}$$

Performing case analysis on b .

Case I: $b \in \{b_1 : B_1 \mid \exists x : B_2 \bullet b_1 \underline{\text{comparable}} x\}$

$$\begin{aligned}
& (3) \\
\Rightarrow & \{\text{simplification of expression}\} \\
& b \in B_1 \\
& \wedge \exists x : B_2 \bullet b \underline{\text{comparable}} x \\
& \wedge \nexists e : \text{comparables}(B_1, B_2) \bullet b \subseteq e \\
\Rightarrow & \{\text{choose } b' = b\} \\
& \exists b' : B_1 \bullet b' \subseteq b
\end{aligned} \tag{4}$$

Case II: $b \in \{b_2 : B_2 \mid \exists x : B_1 \bullet b_2 \underline{\text{comparable}} x\}$

$$\begin{aligned}
& (3) \\
\Rightarrow & \{\text{simplification of expression}\} \\
& b \in B_2 \\
& \wedge \exists x : B_1 \bullet b \underline{\text{comparable}} x \\
& \wedge \nexists e : \text{comparables}(B_1, B_2) \bullet b \subseteq e \\
\Rightarrow & \{\text{further simplification}\} \\
& b \in B_2 \\
& \wedge \exists x : B_1 \bullet x \subseteq b \\
\Rightarrow & \{\text{choose } b' = x\} \\
& \exists b' : B_1 \bullet b' \subseteq b
\end{aligned} \tag{5}$$

Hence

$$\begin{aligned}
& (4) \wedge (5) \\
\Rightarrow & \{\text{by assumption and case analysis above}\} \\
& \forall b : (B_1 \sqcup B_2) \bullet \exists b' : B_1 \bullet b' \subseteq b \\
\equiv & \{\text{by definition of } \sqsubseteq\} \\
& B_1 \sqsubseteq (B_1 \sqcup B_2)
\end{aligned}$$

By a symmetrical argument, $B_2 \sqsubseteq (B_1 \sqcup B_2)$, therefore

$$\begin{aligned}
& B_1 \sqsubseteq (B_1 \sqcup B_2) \\
& B_2 \sqsubseteq (B_1 \sqcup B_2)
\end{aligned}$$

□

Lemma 6

The join operator \sqcup constructs a behaviour below any upper bound.

We are required to show that if B_1, B_2, B : *Behaviour* are behaviours satisfying:

$$B_1 \sqsubseteq B \tag{1}$$

$$B_2 \sqsubseteq B \tag{2}$$

then

$$(B_1 \sqcup B_2) \sqsubseteq B$$

Proof

Note that

$$\forall x, y, z : \text{Output} \bullet (x \subseteq z \wedge y \subseteq z) \Rightarrow x \underline{\text{comparable}} y \tag{3}$$

Take any $b : B$.

$$\begin{aligned} & (1) \wedge (2) \\ \Rightarrow & \{\text{by definition of } \sqsubseteq\} \\ & \exists b_1 : B_1 \bullet b_1 \subseteq b \wedge \exists b_2 : B_2 \bullet b_2 \subseteq b \\ \Rightarrow & \{\text{behaviours contain no prefixes}\} \\ & \exists b_1 : B_1 \bullet (b_1 \subseteq b \wedge \nexists b'_1 : B_1 \bullet b_1 \subset b'_1) \\ & \wedge \exists b_2 : B_2 \bullet (b_2 \subseteq b \wedge \nexists b'_2 : B_2 \bullet b_2 \subset b'_2) \\ \Rightarrow & \{\text{by (3)}\} \\ & \exists b_1 : B_1; b_2 : B_2 \bullet \\ & \quad b_1 \underline{\text{comparable}} b_2 \\ & \quad \nexists b'_1 : B_1 \bullet b_1 \subset b'_1 \\ & \quad \nexists b'_2 : B_2 \bullet b_2 \subset b'_2 \\ & \quad b_1 \subseteq b \wedge b_2 \subseteq b \\ \Rightarrow & \{\text{if } B_1 \text{ and } B_2 \text{ contain no prefixes, neither does } B_1 \cup B_2\} \\ & \exists b_1 : B_1; b_2 : B_2 \bullet \\ & \quad b_1 \underline{\text{comparable}} b_2 \\ & \quad \nexists e : B_1 \cup B_2 \bullet b_1 \subset e \wedge b_2 \subset e \\ & \quad b_1 \subseteq b \wedge b_2 \subseteq b \\ \Rightarrow & \{\text{property of comparables}\} \\ & \exists b_1 : B_1; b_2 : B_2 \bullet \\ & \quad \{b_1, b_2\} \subseteq \text{comparables}(B_1, B_2) \\ & \quad \nexists e : B_1 \cup B_2 \bullet b_1 \subset e \wedge b_2 \subset e \\ & \quad b_1 \subseteq b \wedge b_2 \subseteq b \end{aligned} \tag{4}$$

Performing case analysis on the relationship between b_1 and b_2 .

Case I: $b_1 = b_2$

$$\begin{aligned}
& (4) \\
\Rightarrow & \{ \text{simplify to eliminate } b_1 \} \\
& \exists b_2 : B_2 \bullet \\
& \quad b_2 \in \text{comparables}(B_1, B_2) \\
& \quad \nexists e : B_1 \cup B_2 \bullet b_2 \subset e \\
& \quad b_2 \subseteq b \\
\Rightarrow & \{ \text{by definition of } \uparrow \} \\
& \exists b_2 : \uparrow(\text{comparables}(B_1, B_2)) \bullet b_2 \subseteq b \tag{5}
\end{aligned}$$

Case II: $b_1 \subset b_2 \vee b_2 \subset b_1$

Without loss of generality, suppose $b_1 \subset b_2$.

$$\begin{aligned}
& (4) \\
\Rightarrow & \{ \text{simplify} \} \\
& \exists b_1 : B_1; b_2 : B_2 \bullet \\
& \quad \{b_1, b_2\} \subseteq \text{comparables}(B_1, B_2) \\
& \quad \nexists e : B_1 \cup B_2 \bullet b_1 \subset e \wedge b_2 \subset e \\
& \quad b_1 \subset b_2 \subseteq b \\
\Rightarrow & \{ \text{simplify to eliminate } b_1 \} \\
& \exists b_2 : B_2 \bullet \\
& \quad b_2 \in \text{comparables}(B_1, B_2) \\
& \quad \nexists e : B_1 \cup B_2 \bullet b_2 \subset e \\
& \quad b_2 \subseteq b \\
\Rightarrow & \{ \text{by definition of } \uparrow \} \\
& \exists b_2 : \uparrow(\text{comparables}(B_1, B_2)) \bullet b_2 \subseteq b \tag{6}
\end{aligned}$$

Hence,

$$\begin{aligned}
& (5) \wedge (6) \\
\Rightarrow & \{ \text{by assumption and case analysis above} \} \\
& \forall b : B \bullet \exists b' : \uparrow(\text{comparables}(B_1, B_2)) \bullet b' \subseteq b \\
\Rightarrow & \{ \text{by definition of } \sqcup \} \\
& \forall b : B \bullet \exists b' : (B_1 \sqcup B_2) \bullet b' \subseteq b \\
\equiv & \{ \text{by definition of } \sqsubseteq \} \\
& (B_1 \sqcup B_2) \sqsubseteq B \\
& \quad \square
\end{aligned}$$

Theorem 3

The join operator \sqcup defines the least upper bound operation on the lattice defined by \sqsubseteq .

Proof

The result follows directly from Lemmas 5 and 6. □

Appendix D

Theorem 4

The methodology is sound, i.e. if the set C of those classes in \mathcal{L} having a behaviour with respect to a program p identical to the desired behaviour b is non-empty, then

$$\text{behavioural_retrieval}(\mathcal{L}) = C$$

The definitions of *behavioural_retrieval*, *strategy1* and *strategy2* are repeated below for the reader's reference.

| | |
|--|---|
| $\overline{\text{behavioural_retrieval} : \mathbb{P} \text{ Class} \rightarrow \mathbb{P} \text{ Class}}$ | $\forall L : \text{Class} \bullet$ $\text{behavioural_retrieval}(L) =$ $\{c : L \mid \downarrow(\text{execute}(p)(c)) \in$ $(\text{strategy1}(\pi_{\text{execute}(p)}(L)) \cup \text{strategy2}(\pi_{\text{execute}(p)}(L)))\}$ |
| $\overline{\text{strategy1} : \mathbb{P} \text{ Behaviour} \rightarrow \mathbb{P} \text{ Behaviour}}$ | $\forall B : \mathbb{P} \text{ Behaviour} \bullet$ $\text{strategy1}(B) =$ $\max_{\sqsubseteq} \{b' : B \mid \text{comparables}(b, b') \neq \emptyset \wedge$ $b \sqcup b' \in \min_{\sqsubseteq}(\pi_{\sqcup b}(\mathcal{B}))\}$ |
| $\overline{\text{strategy2} : \mathbb{P} \text{ Behaviour} \rightarrow \mathbb{P} \text{ Behaviour}}$ | $\forall B : \mathbb{P} \text{ Behaviour} \bullet$ $\text{strategy2}(B) =$ $\min_{\sqsubseteq} \{b' : B \mid b \sqcap b' \in \max_{\sqsubseteq}(\pi_{\sqcap b}(\mathcal{B}))\}$ |

Proof

Let \mathcal{B} denote the set of behaviours $\pi_{\text{execute}(p)}(\mathcal{L})$. Then for any $c : C$,

$$\begin{aligned} & c \in \mathcal{L} \wedge \downarrow(\text{execute}(p)(c)) = b \\ \Rightarrow & \{\text{by definition of } \pi_{\text{execute}(p)}\} \\ & b \in \mathcal{B} \end{aligned} \tag{1}$$

Because $c \in \{c' : \mathcal{L} \mid \downarrow \text{execute}(p)(c') \in \{b\}\}$, it is enough to show that

$$\{b\} = \text{strategy1}(\mathcal{B}) \tag{Proposition I}$$

and

$$\{b\} = \text{strategy2}(\mathcal{B}) \tag{Proposition II}$$

Lemma 7

Proposition (I) holds.

Proof

Note that if a behaviour in \mathcal{B} is not join-compatible with b , the set $\pi_{\sqcup b}(\mathcal{B})$ will exclude that behaviour. Hence in this proof we first show that b is join-compatible with itself, and then show that Proposition I holds for those behaviours which are join-compatible with b . Throughout the rest of this proof, let \mathcal{B}' be the set

$$\mathcal{B} - \{c : \mathcal{B} \mid comparables(b, c) = \emptyset\}.$$

$$\begin{aligned} & (1) \\ \Rightarrow & \{comparables(b, b) = b \neq \emptyset\} \\ & b \in \mathcal{B}' \end{aligned} \tag{2}$$

In order to establish Proposition I, it now suffices to show

$$\begin{aligned} & \{b\} = strategy1(\mathcal{B}') \\ & (2) \\ \Rightarrow & \{b \sqcup b = \uparrow(comparables(b, b)) = \uparrow(b) = b \text{ for any behaviour } b\} \\ & b \in \pi_{\sqcup b}(\mathcal{B}') \end{aligned} \tag{3}$$

Take any $a \in \mathcal{B}'$.

$$\begin{aligned} & a \in \mathcal{B}' \\ \Rightarrow & \{\sqcup \text{ is the least upper bound (Theorem 3)}\} \\ & b \sqsubseteq (a \sqcup b) \end{aligned}$$

Hence,

$$\forall a : \mathcal{B}' \bullet b \sqsubseteq (a \sqcup b) \tag{4}$$

$$\begin{aligned} & (3) \wedge (4) \\ \Rightarrow & \{\text{by definition of } min_{\sqsubseteq}\} \\ & min_{\sqsubseteq}(\pi_{\sqcup b}(\mathcal{B}')) = \{b\} \end{aligned} \tag{5}$$

$$\begin{aligned} & (3) \wedge (5) \\ \Rightarrow & \{\text{by definition of } min_{\sqsubseteq} \text{ and } \in\} \\ & b \in min_{\sqsubseteq}(\pi_{\sqcup b}(\mathcal{B}')) \end{aligned} \tag{6}$$

$$\begin{aligned} & (2) \wedge (3) \wedge (6) \\ \Rightarrow & \{b \sqcup b = b, \text{ definition of } \mathcal{B}'\} \\ & b \in \{b' : \mathcal{B}' \mid comparables(b, b') \neq \emptyset \\ & \quad \wedge b' \sqcup b \in min_{\sqsubseteq}(\pi_{\sqcup b}(\mathcal{B}'))\} \end{aligned} \tag{7}$$

Let \mathcal{B}_1 denote the set

$$\{b' : \mathcal{B}' \mid \text{comparables}(b, b') \neq \emptyset \\ \wedge b' \sqcup b \in \min_{\sqsubseteq}(\pi_{\sqcup b}(\mathcal{B}'))\}$$

Take any $a \in \mathcal{B}_1$.

$$\begin{aligned} & (5) \\ \Rightarrow & \{\text{property of } \in \text{ on a singleton set}\} \\ & b = (a \sqcup b) \\ \Rightarrow & \{\sqcup \text{ is the least upper bound (Theorem 3)}\} \\ & b = (a \sqcup b) \wedge a \sqsubseteq (a \sqcup b) \\ \equiv & \{\text{simplification}\} \\ & a \sqsubseteq b \end{aligned}$$

Hence

$$\forall a : \mathcal{B}_1 \bullet a \sqsubseteq b \tag{8}$$

$$\begin{aligned} & (7) \wedge (8) \\ \Rightarrow & \{\text{by definition of } \max_{\sqsubseteq}\} \\ & \{b\} = \max_{\sqsubseteq}(\mathcal{B}_1) \\ \equiv & \{\text{by definition of } \textit{strategy1}\} \\ & \textit{strategy1}(\mathcal{B}') = \{b\} \\ & \square \end{aligned}$$

Lemma 8

Proposition (II) holds.

Proof

$$\begin{aligned} & (1) \\ \Rightarrow & \{b \sqcap b = \downarrow(b \cup b) = \downarrow(b) = b \text{ for any behaviour } b\} \\ & b \in \pi_{\sqcap b}(\mathcal{B}) \end{aligned} \tag{2}$$

Take any $a \in \mathcal{B}$.

$$\begin{aligned} & a \in \mathcal{B} \\ \Rightarrow & \{\sqcap \text{ is the greatest lower bound (Theorem 2)}\} \\ & (a \sqcap b) \sqsubseteq b \end{aligned}$$

Hence

$$\forall a : \mathcal{B} \bullet (a \sqcap b) \sqsubseteq b \tag{3}$$

$$\begin{aligned} & (2) \wedge (3) \\ \Rightarrow & \{\text{by definition of } \max_{\sqsubseteq}\} \\ & \max_{\sqsubseteq}(\pi_{\sqcap b}(\mathcal{B})) = \{b\} \end{aligned} \tag{4}$$

$$\begin{aligned}
& (2) \wedge (4) \\
\Rightarrow & \{ \text{by definition of } \max_{\sqsubseteq} \text{ and } \in \} \\
& b \in \max_{\sqsubseteq}(\pi_{\sqcap b}(\mathcal{B}))
\end{aligned} \tag{5}$$

$$\begin{aligned}
& (2) \wedge (5) \\
\Rightarrow & \{ b \sqcap b = b \} \\
& b \in \{ b' : \mathcal{B} \mid b' \sqcap b \in \max_{\sqsubseteq}(\pi_{\sqcap b}(\mathcal{B})) \}
\end{aligned} \tag{6}$$

Let \mathcal{B}_2 denote the set

$$\{ b' : \mathcal{B} \mid b' \sqcap b \in \max_{\sqsubseteq}(\pi_{\sqcap b}(\mathcal{B})) \}$$

Take any $a \in \mathcal{B}_2$

$$\begin{aligned}
& (4) \\
\Rightarrow & \{ \text{property of } \in \text{ on a singleton set} \} \\
& (a \sqcap b) = b \\
\Rightarrow & \{ \sqcap \text{ is the greatest lower bound (Theorem 2)} \} \\
& (a \sqcap b) = b \wedge (a \sqcap b) \sqsubseteq a \\
\equiv & \{ \text{simplification} \} \\
& b \sqsubseteq a
\end{aligned}$$

Hence

$$\forall a : \mathcal{B}_2 \bullet b \sqsubseteq a \tag{7}$$

$$\begin{aligned}
& (6) \wedge (7) \\
\Rightarrow & \{ \text{by definition of } \min_{\sqsubseteq} \} \\
& \{ b \} = \min_{\sqsubseteq}(\mathcal{B}_2) \\
\equiv & \{ \text{by definition of } \textit{strategy2} \} \\
& \textit{strategy2}(\mathcal{B}) = \{ b \}
\end{aligned}$$

□

Theorem 4 follows directly from Lemmas 7 and 8.

□