A Fully Abstract Semantics of Classes for Object-Z

Graeme Smith
Software Verification Research Centre, University of Queensland, Australia

Keywords: Full abstraction; Object orientation; Object-Z; Semantics

Abstract. This paper presents a fully abstract semantics of classes for the object-oriented formal specification language Object-Z. Such a semantics includes no unnecessary syntactic details and, hence, describes a class in terms of the external behaviour of its objects only. The semantics, based on an extension of existing process models, defines a notion of behavioural equivalence which is stronger than that of CSP and weaker than that of CCS.

1. Introduction

Object orientation [Boo90, Mey88] is a modular design methodology based on the notion that a system is composed of a collection of interacting objects whose behaviours are specified by classes1. Although it originated as a programming paradigm [BDM73, GoR83], recently it has found increasing application in the field of formal methods. In particular, a number of existing specification languages have been extended to incorporate object-oriented concepts [DKR91b, SBC92, CRS90, SIG89, Nar87]. The enhanced structuring provided by such approaches not only improves the clarity of specifications but can also aid in the subsequent steps of verification and refinement. This, however, is only possible once a formal semantics of the incorporated object-oriented features has been developed.

In terms of formal verification, the full benefits of the object-oriented approach are only realised when the properties of composite objects, i.e. objects

---

1 This paper assumes familiarity with the basic concepts of object orientation. Readers not so familiar may wish to consult [Boo90] or [Mey88].

Correspondence and offprint requests to: Graeme Smith, Software Verification Research Centre, Department of Computer Science, University of Queensland 4072, Australia.
which are composed of other objects, can be derived from the properties of their components. This is only possible if the semantic denotation of the class of an object is derivable from the denotations of the classes of the objects of which it is composed. A semantics of classes with this property is described as being compositional.

Another desirable property of a semantics of classes is that the denotation of any class is only as detailed as necessary for the semantics to be compositional. A semantics with this property is described as being fully abstract [Plo77, Sto88, McC88]. Intuitively, the denotation of a class in a fully abstract semantics contains no unnecessary implementational details and, therefore, describes only the external behaviour of its objects. Consequently, only objects of classes with equal denotations in such a semantics will be behaviourally equivalent, i.e., will behave identically in any context, or environment².

Full abstraction has both theoretical and practical significance. Theoretically, a fully abstract semantics of classes captures the precise meaning of a class independent of its syntactic representation. Practically, by describing the external behaviour of the objects of a class, it enables a concise definition of behavioural compatibility, or subtyping, to be developed. A class is said to be a subtype of another class when an object of that class can be substituted for an object of the other class in any system so that the system, after the substitution has occurred, can only behave in ways that it could have behaved before the substitution. Such a definition is at the heart of a theory of class refinement [Ulr91].

This paper presents a fully abstract semantics of classes for the formal specification language Object-Z [DKR91b]: an object-oriented extension of the formal specification language Z³. Section 2 introduces Object-Z and presents a set-theoretic model of classes which is used to explain the meaning of object instantiation and the initialisation of, and application of operations to, objects of a class. Section 3 reviews existing approaches to proving that a semantics is fully abstract and presents an alternative approach for object-oriented languages. This approach can be used to show that a model of classes in Object-Z is fully abstract. Section 4 presents some preliminary models of classes based on existing process models which are not fully abstract but are used to motivate the definition of a fully abstract model. Section 5 presents the fully abstract model of classes along with associated proofs of composiionality and full abstraction. Section 6 evaluates the contributions of this paper with respect to related work on the semantics of processes.

2. Object-Z

Object-Z has been developed over a number of years by a team of researchers and various versions of the language have been previously published [CDD90, DKR91b, SBC92]. The most complete and well-known of these is [DKR91b] (a shorter version of [DKR91b] is published as [DKR91a]). The version of Object-Z described in this paper is identical to [DKR91b] except for the following points.

² This notion of behavioural equivalence is often referred to as observational equivalence in the literature.
³ A familiarity with Z is assumed in this paper. Readers not so familiar may wish to consult [Spi89] or [PST90].
The use of the language is restricted to reflect the philosophy that the state of an object is hidden from its environment. That is, an object may only be accessed through the procedural interface defined by its class. This philosophy is consistent with that adopted in many object-oriented programming languages (e.g., [GoF83]) and is central to the fully abstract semantics developed in this paper. It may be argued that in a specification language the state of an object should not be hidden so that the effect of operations can be specified in a more direct and abstract way. However, allowing an object's state to be changed other than as directly specified by its operations makes it difficult to verify systems in a modular fashion.

The major restriction resulting from state hiding is that the attributes of objects cannot be accessed directly. If the value of an attribute is required in the environment of an object then an operation must be included in the object's class to return this value. Other restrictions will be detailed in the following sections and are described in detail in [Smi92].

History invariants are not included in the definition of classes. This is for practical rather than technical reasons: history invariants add nothing to the development of the fully abstract model of classes but complicate the associated proofs. However, so that the model is applicable to Object-Z with history invariants, object behaviours are allowed to be infinite. A proof that the model is also fully abstract for Object-Z classes with history invariants can be found in [Smi92].

To enable classes in Object-Z to be used as types, the meaning of a class is taken to be a set of values. Each value corresponds to a potential object of the class at some stage of its evolution. Following the work of Duke and Duke [DuD90], the value chosen to represent an object is the sequence of states the object has passed through together with the corresponding sequence of events the object has undergone. This value is referred to as the history of the object.

Section 2.1 introduces Object-Z with an example specification of a simple class and system. Section 2.2 presents a model of classes based upon the syntactic structure of classes in Object-Z. This model is used to derive the history model of classes in Section 2.3. Both models are defined using the specification language Z.

2.1. Introduction to Object-Z

An Object-Z class schema, often referred to simply as a class, is represented syntactically as a named box with zero or more generic parameters. In this box there may be local type and constant definitions, at most one state and associated initial state schema and zero or more operations. A class may also include the names of inherited classes, however, this is not important in the discussion of the semantics of classes because for any class specified using inheritance there is a semantically equivalent class specified without using inheritance (see [DKR91b] or [DKR91a] for details). For the purposes of this paper, the basic structure of a class is as follows.

---

4 History invariants may specify liveness properties which restrict the infinite behaviours of objects. See [Smi92] for details.
As an example, consider the following Object-Z specification of a simple vending machine.

```
Class: VendingMachine

credit : {0, 50, 100, 150}

I
credit = 0

Coin
Δ(credit)
coin? : {50, 100}

credit < 100
credit' = credit + coin?

Choc
Δ(credit)
change! : {0, 50}

credit ≥ 100
change! = credit - 100
credit' = 0
```

The class `VendingMachine` consists of a state schema, an initial state schema and two operations (there are no generic parameters, or type or constant definitions). The state schema has no name and the initial state schema is identified by the name `I`. The state schema is implicitly included in the initial state schema and in each operation schema in both primed and unprimed form. The operations differ from Z operation schemas in that they have a Δ-list. The Δ-list contains those state variables which may change when the operation is applied to an object of the class. All other state variables are unchanged.

Informally, the class describes a vending machine which allows a customer to purchase a chocolate for one dollar by inserting either one dollar or 50 cent coins. The state variable `credit` denotes the amount of money inserted by a customer. Initially, the credit is zero. The operation `Coin` represents the customer inserting a coin, denoted by the input parameter `coin?`, whose value is added to `credit`. The precondition of `Coin` prevents a customer inserting another coin when the credit already exceeds one dollar. The operation `Choc` represents the customer receiving a chocolate. The precondition allows this operation to occur only when the credit is at least one dollar. The customer also receives change, denoted by the output parameter `change!`, and the credit is returned to zero.
In order to specify objects composed of other objects, a class may have objects of other classes as state variables. As an example, consider the following class which specifies a system composed of two independent vending machines.

\[
\text{VendingMachinePair}
\]
\[
\text{vm}_1, \text{vm}_2 : \text{VendingMachine}
\]
\[
I
\]
\[
\text{vm}_1, I
\]
\[
\text{vm}_2, I
\]
\[
\text{Coin}_1 \equiv \text{vm}_1, \text{Coin}
\]
\[
\text{Choc}_1 \equiv \text{vm}_1, \text{Choc}
\]
\[
\text{Coin}_2 \equiv \text{vm}_2, \text{Coin}
\]
\[
\text{Choc}_2 \equiv \text{vm}_2, \text{Choc}
\]

The class \textit{VendingMachinePair} has two state variables, \text{vm}_1 and \text{vm}_2, representing objects of the class \textit{VendingMachine}. The objects are initialised and have operations applied to them using the dot notation. The precise meaning of this notation will be given in terms of the history model of classes in Section 2.3.

More complex systems can also be specified. A binary schema operator \texttt{||} is available to facilitate the specification of inter-object communication and a schema nesting operator \texttt{•} to facilitate the specification of operations on objects in aggregates. Details of these operators can be found in [DKR91b] or [DKR91a]. All existing Z schema operators are also available in a class with the exception of the schema composition operator \texttt{∃}. The reason for the exclusion of this operator is discussed in Section 6.

2.2. Structural Model

Structurally, a class in Object-Z consists of a set of attributes and a set of operations which act upon those attributes. The set of attributes includes, as well as all local constants and state variables of the class, any global constants to which the class may refer. Each operation has a set of parameters for the purpose of input and output.

Attributes and operations within a class are given unique names, or identifiers. Also, parameters within a particular operation are identified uniquely with respect to each other and with respect to the attributes and operations of the class.

Let \textit{Id} denote the set of all possible identifiers. In Object-Z, this would be the set of all strings of alphanumerics, underscores and symbols excluding certain reserved symbols. Such details will not be elaborated upon formally here.

Let \textit{Value} denote the set of all possible values that could be assigned to any identifier of any type. A \textit{state} is an assignment of values to a set of identifiers rep-
resenting attributes. It can be defined as a finite partial function from identifiers to values as follows.

\[ \text{State} \quad \text{Id} \rightarrow \text{Value} \]

An \textit{event} can be defined as a tuple consisting of the operation’s name and a finite partial function defining the values of the operation’s parameters.

\[ \text{Event} \quad \text{Id} \times (\text{Id} \rightarrow \text{Value}) \]

The auxiliary functions \textit{op} and \textit{params} are defined to enable access to an event’s associated operation name and parameter values respectively.

\[
\begin{align*}
\text{op} &: \text{Event} \to \text{Id} \\
\text{params} &: \text{Event} \to (\text{Id} \to \text{Value}) \\
\forall e : \text{Event} \quad e = (\text{op}(e), \text{params}(e))
\end{align*}
\]

The structural model of a class defines a set of objects in terms of the states they can be in and the events they can undergo. It includes as state variables the following.

\textit{states} - the set of possible states of an object of the class. This includes those states which are composed of the attributes of the class (including any global constants) and which satisfy the state invariant of the class.

\textit{initial} - the set of possible initial states of an object of the class. This includes those states from the set of possible states of an object of the class which satisfy the predicate of the initial state schema of the class.

\textit{trans} - a function from the set of possible events an object of the class may undergo to the associated set of state transitions. This includes, for each event, those pairs of states from the set of possible states of an object of the class which satisfy the precondition and resulting postcondition of the operation associated with the event. The interpretation of operations in an Object-Z class differs from that in Z in that an operation cannot occur when its precondition is not enabled. In Z, the operation would be able to occur but the outcome would be unspecified.

<table>
<thead>
<tr>
<th>ClassStruct</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{states} : \text{State}</td>
</tr>
<tr>
<td>\textit{initial} : \text{State}</td>
</tr>
<tr>
<td>\textit{trans} : \text{Event} \to (\text{State} \leftrightarrow \text{State})</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\forall s_1, s_2 : \text{states} \quad & \text{dom } s_1 = \text{dom } s_2 \\
\text{initial} & \subseteq \text{states} \\
\forall e_1, e_2 : \text{dom trans} \quad & (\text{op}(e_1) = \text{op}(e_2)) \Rightarrow \text{dom params}(e_1) = \text{dom params}(e_2) \\
\forall e : \text{dom trans} \quad & \text{trans}(e) \subseteq \text{states} \leftrightarrow \text{states}
\end{align*}
\]

The predicate of \textit{ClassStruct} ensures that all states of the class have the
same set of attributes and that all events corresponding to a given operation name have the same parameters.

The schema ClassStruct effectively defines a state transition system. The same information can, therefore, also be represented graphically as a state transition diagram. As an example, consider the class VendingMachine of Section 2.1.

The sets attr, ops and the function op_params for this class are as follows.

\[\text{attr} = \{\text{'}credit\text{'}\}\]
\[\text{ops} = \{\text{'}Coin\text{',} \text{'}Choc\text{'}\}\]
\[\text{op_params} = \{\text{'}Coin\text{'} \mapsto \{\text{'}coin?'\text{'}\}, \text{'}Choc\text{'} \mapsto \{\text{'}change!'\text{'}\}\}\]

The notation ‘credit’ denotes the name of attribute credit as opposed to its semantic value. Similarly, the notations ‘Coin’, ‘Choc’, ‘coin?’ and ‘change!’ denote the names of the corresponding operations and parameters.

The sets states, initial and the function trans for the class VendingMachine are as follows.

\[\text{states} = \{\{\text{'}credit\text{'} \mapsto 0\}, \{\text{'}credit\text{'} \mapsto 50\}, \{\text{'}credit\text{'} \mapsto 100\}, \{\text{'}credit\text{'} \mapsto 150\}\}\]
\[\text{initial} = \{\{\text{'}credit\text{'} \mapsto 0\}\}\]
\[\text{trans} = \{\{\text{'}Coin\text{'}\}, \{\text{'}coin?'\mapsto 50\}\} \mapsto \{\{\{\text{'}credit\text{'} \mapsto 0\}, \{\text{'}credit\text{'} \mapsto 50\}\}, \{\text{'}credit\text{'} \mapsto 100\}\}\]
\[\{\text{'}Coin\text{'}\}, \{\text{'}coin?'\mapsto 100\}\} \mapsto \{\{\{\text{'}credit\text{'} \mapsto 0\}, \{\text{'}credit\text{'} \mapsto 100\}\}, \{\text{'}credit\text{'} \mapsto 50\}, \{\text{'}credit\text{' }\mapsto 150\}\}\]
\[\{\text{'}Choc\text{'}\}, \{\text{'}change!'\mapsto 0\}\} \mapsto \{\{\{\text{'}credit\text{'} \mapsto 100\}, \{\text{'}credit\text{'} \mapsto 0\}\}\]
\[\{\text{'}Choc\text{'}\}, \{\text{'}change!'\mapsto 50\}\} \mapsto \{\{\{\text{'}credit\text{'} \mapsto 150\}, \{\text{'}credit\text{'} \mapsto 0\}\}\}\]

An equivalent state transition diagram is shown in Fig. 1. Circles represent states and labelled arcs represent events. A circle with an unlabelled arc entering it is an initial state.

![State transition diagram of the VendingMachine class.](image-url)
2.3. History Model

The history model defines the type of a class in Object-Z as opposed to its syntactic structure. This type is defined to be the set of histories that objects of the class can undergo. Intuitively, the history of an object represents the sequence of states the object has passed through together with the sequence of operations it has undergone. It can be modelled as a non-empty sequence of states $ss$, where each state assigns values to a common set of identifiers, and a sequence of events $es$ such that the number of states in $ss$ is one more than the number of events in $es$ unless $ss$ is infinite in which case $es$ is also infinite. The relationship between the sequence of states $ss$ and the corresponding sequence of events $es$ of a history can be represented by a state transition diagram as shown in Fig. 2.

Building on the definitions in Section 2.2, a history is defined as follows\(^5\).

$$
\textit{History} \\
\textit{states} : \text{seq}_\infty \text{State} \\
\textit{events} : \text{seq}_\infty \text{Event}
$$

\(\forall i, j : \text{dom states} \cdot \text{dom states}(i) = \text{dom states}(j)\)

\(\forall i : 1 \cdot i \in \text{dom events} \iff i + 1 \in \text{dom states}\)

Any history whose sequences of states and events are prefixes of or equal to those of another history is referred to as a pre-history of that history. Intuitively, a pre-history of an object's history represents the history of that object at some stage of its evolution.

$$
\textit{prehist} : \text{History} \rightarrow \text{History}
$$

\(\forall h : \text{History} \cdot \textit{prehist}(h) = \{ \textit{ph} : \text{History} \mid \text{ph.states} \subseteq h.\text{states} \land \text{ph.events} \subseteq h.\text{events} \}\)

The set of histories representing a class can be derived from the structural model of the class using the function $H$ defined below.

$$
H : \text{ClassStruct} \rightarrow \text{History}
$$

\(\forall c : \text{ClassStruct} \cdot H(c) = \{ h : \text{History} \mid h.\text{states}(1) \in c.\text{initial} \land \forall i : \text{dom h.events} \cdot h.\text{events}(i) \in \text{dom c.trans} \land (h.\text{states}(i), h.\text{states}(i + 1)) \in c.\text{trans}(h.\text{events}(i)) \}\)

---

\(^5\) $\text{seq}_\infty X$ denotes the set of finite and infinite sequences whose elements are drawn from $X$, i.e. $\text{seq}_\infty X = \text{seq}X \cup 1 \rightarrow X$. 
The first state in the sequence of states of any history of a class is an initial state of the class and each pair of consecutive states is a possible state transition of the corresponding event in the sequence of events.

An important property of the set of histories of a class is that it is pre-history closed, i.e. given any history in the set, all pre-histories of that history are also in the set. This is necessary as any pre-history of an object's history is the history of that object at some earlier stage in its evolution and hence represents a possible history of the object’s class.

Adopting the history model of classes, a class in Object-Z may be used as a type. For example, the declaration \( \text{vm}_1 : \text{VendingMachine} \) in the class \( \text{VendingMachinePair} \) of Section 2.1 is semantically identical to \( \text{vm}_1 : \mathcal{H}(\text{VM}) \) where \( \text{VM} : \text{ClassStruct} \) is the structural model of the class \( \text{VendingMachine} \).

The dot notation for initialising and applying operations to objects can also be defined in terms of the history model. Semantically, the notation \( \text{vm}_1.I \) is identical to the schema which states that the object \( \text{vm} \) has not undergone any events.

\[
\text{vm}_1.I
\]
\[
\text{vm}_1.events = \langle \rangle
\]

It can be deduced, from the definition of \( \mathcal{H} \), that the state of \( \text{vm}_1 \), i.e. the final state in \( \text{vm}_1 \)'s state sequence, satisfies the predicate of the \( I \) schema of its class. That is, the predicate \( \text{credit} = 0 \) is true. Similar meaning can be given to the notation \( \text{vm}_1.I \).

The notation \( \text{vm}_1.Coin \) can also be represented semantically as a schema which states that the object \( \text{vm}_1 \) undergoes an event associated with the operation \( \text{Coin} \) provided the sequence of events \( \text{vm}_1 \) has undergone is finite. The condition of finiteness is required as only appending to a finite sequence is defined. The input variable \( \text{coin}? \) declared in \( \text{Coin} \) is included in the signature of this schema. In general, all input and output variables declared in the corresponding operation’s signature would be similarly included.

\[
\text{vm}_1.Coin
\]
\[
\Delta(\text{vm}_1)
\]
\[
\text{coin}? : \{50, 100\}
\]
\[
\text{vm}_1.events \in \text{seq Event}
\]
\[
\text{from} \; \text{vm}_1, \text{states} = \text{vm}_1, \text{states}
\]
\[
\text{vm}_1.events = \text{vm}_1.events \cap (\{'\text{Coin}', \{'\text{coin}?\} \mapsto \text{coin}?\})
\]

It can be deduced that if \( \text{vm}_1.Coin \) occurs then the state of \( \text{vm}_1 \) before the operation satisfies the precondition of the operation \( \text{Coin} \) and the state of \( \text{vm}_1 \) after the operation is related to the state before by a state transition defined by \( \text{Coin} \). Similar meanings can be given to the notations \( \text{vm}_1, \text{Choc}, \text{vm}_2, \text{Coin} \) and \( \text{vm}_2, \text{Choc} \).

The above schema expansions of expressions involving the dot notation are given only to clarify the meaning of such expressions with respect to the history model. In practice, expressions which equate an object to a history are not allowed in Object-Z specifications as they allow objects to be used in ways other than specified by their class. Expressions such as \( a = b \) and \( a \in A \), where \( a \) and \( b \) are objects and \( A \) is a set of objects, are also not allowed for similar reasons discussed in Section 6. In brief, an object may only be referred to in the \( \Delta \)-list.
3. Full Abstraction

A language may be thought of as consisting of components which may be used to construct programs, or systems, in the language, e.g. assignment statements in a sequential programming language or classes in an object-oriented language. A semantics of a language is fully abstract precisely when any two such components with identical semantic denotations are behaviourally equivalent.

To prove a semantics is fully abstract, it is necessary to have another semantics of the language from which a definition of behavioural equivalence can be derived. Traditionally, this semantics describes the input/output behaviour of programs, or systems, in the language. Classes in an object-oriented language, however, cannot always be easily described as a relation between input and output. Alternative means of proving a semantics is fully abstract, therefore, have had to be developed (e.g. see [Yel89]).

Section 3.1 examines existing approaches to proving that a semantics is fully abstract. Section 3.2 presents an alternative approach for object-oriented languages and outlines how this approach may be used to show that a model of classes in Object-Z is fully abstract.

3.1. Existing Approaches to Full Abstraction

The most common approach to proving that a semantics of a language is fully abstract relies on the existence of another semantics of the language which describes the observable behaviour of programs, or systems, in the language. The observable behaviour of a system describes only what the system can be observed to do. It reflects the view of a passive observer who has no control over the system. The external behaviour, as described by a fully abstract semantics, reflects the view of a client, i.e. another system which may be composed with the system in question and direct it in the events it undergoes. Such a client may, for example, be able to detect failure of the system to perform particular operations. This failure, however, would not be seen by an observer and so is not a part of the observable behaviour.

Traditionally, the observable behaviour is defined as the input/output behaviour of programs, or systems, in the language. Two language components \( c_1 \) and \( c_2 \) are said to be behaviourally equivalent if the input/output behaviour of the program, or system, formed by placing \( c_1 \) in any context \( C \) is identical to the input/output behaviour of the program, or system, formed by placing \( c_2 \) in \( C \).

To state this more formally, let the notation \( C[c] \) represent the program, or system, formed by placing the language component \( c \) in the context \( C \). Given a semantics \( \mathcal{IO} \) which describes the input/output behaviour of programs, or systems, in a language, two language components \( c_1 \) and \( c_2 \) are said to be behaviourally equivalent if, for any context \( C \), the following holds.

\[
\mathcal{IO}(C[c_1]) = \mathcal{IO}(C[c_2])
\]
A semantics $D$ is said to be fully abstract with respect to $IO$ if, for any language components $c_1$ and $c_2$ and for any context $C$, the following holds.

$$D(c_1) = D(c_2) \Leftrightarrow IO(C[c_1]) = IO(C[c_2])$$

This approach has been used for many languages. For example, it has been used to define fully abstract models of nondeterministic dataflow networks in [Jon89] and [Rus89]. It is not, however, applicable for object-oriented languages as it is not always easy to describe a class in terms of a relation between input and output. Classes do not necessarily have any inputs or outputs, in the traditional sense, nor do they necessarily have a final state which could be regarded as an ‘output’ state. Classes, in fact, fall into the category of reactive systems defined by Pnueli in [Pnu85]. Such systems are best described in terms of their interaction with their environment.

Yelland [Yel89] proposes a method of proving that a semantics of an object-oriented language is fully abstract. The approach relies on the existence of another semantics of the language which is compositional but not necessarily fully abstract. A notion of behavioural equivalence of systems, where systems are collections of interacting objects, is defined in terms of this semantics. This notion is based on the idea that a system can be observed by a “new” object introduced into the system. An observation is made by initialising the variables of the new object with a set of values, executing a sequence of statements and examining the resulting contents of the variables. Two systems are said to be behaviourally equivalent when no observation can distinguish them.

This approach can only be used to develop a semantics which gives fully abstract denotations to systems and, unlike the approach presented in this paper, not necessarily to the classes of the objects of which those systems are composed. Yelland briefly describes an extension to the approach which would allow classes to also be given fully abstract denotations. This extension relies on having classes identified with systems within the existing semantics.

### 3.2. An Alternative Approach to Full Abstraction

This section presents an alternative approach for proving a semantics of an object-oriented language is fully abstract. The approach is based on the approach presented in Section 3.1 which relies on the existence of another semantics describing observable behaviour.

Adopting the point of view that an object’s state is hidden, its observable behaviour consists of the sequence of events it can undergo. Such a sequence of events is referred to as a trace of the object. Let $C[c]$ denote the class formed by instantiating an object of the class $c$ in the context, or environment, $C$. Given a semantics $T$ which denotes a class by the set of traces its objects can undergo, a semantics $D$ is fully abstract with respect to $T$ if, for any context $C$, the following holds.

$$D(c_1) = D(c_2) \Leftrightarrow T(C[c_1]) = T(C[c_2])$$

---

6. Yelland's understanding of full abstraction is similar to that of Stoughton [Sto88] which, as pointed out by Meyer in [McC88], is actually equivalent to compositionality. Stoughton's notion of contextual full abstraction is equivalent to the definition of full abstraction adopted in this paper.
This approach can be used to show that a model of classes in Object-Z is fully abstract. Adopting the definition of Event from Section 2.2, a trace can be defined as a possibly infinite sequence of events as follows.

\[ \text{Trace} \quad \text{seq}_{\infty} \text{Event} \]

Let \( \text{ClassStruct} \) and the function \( \mathcal{H} \) be defined as in Section 2. The set of traces of a class can be derived from its histories as defined below.

\[ \mathcal{T} : \text{ClassStruct} \rightarrow \text{Trace} \]

\[ \forall c : \text{ClassStruct} \bullet \mathcal{T}(c) = \{ h : \mathcal{H}(c) \bullet h.\text{events} \} \]

A context in Object-Z can be thought of as an incomplete class schema with all occurrences of a class, used in the declaration of one or more objects, elided. To represent a context, a notation similar to the notation used for specifying Object-Z classes is adopted but with occurrences of the elided class represented by the symbol \( \square \). For example, the following is a possible context.

\[
\begin{align*}
\mathcal{C} = \\
\begin{array}{l}
\quad a, b : \square \\
\quad I \\
\quad a.1 \\
\quad b.1
\end{array} \\
Aop \equiv a.\text{Op} \\
Bop \equiv b.\text{Op}
\end{align*}
\]

A class can be placed in a context to form a new class by replacing all occurrences of \( \square \) with the class. For example, a class \( A \), which includes an operation \( \text{Op} \), can be placed in the above context to yield the class \( \mathcal{C}[A] \) defined below.

\[
\begin{align*}
\mathcal{C}[A] = \\
\begin{array}{l}
\quad a, b : A \\
\quad I \\
\quad a.1 \\
\quad b.1
\end{array} \\
Aop \equiv a.\text{Op} \\
Bop \equiv b.\text{Op}
\end{align*}
\]

Two classes which can be placed in exactly the same contexts are said to be signature equivalent. Intuitively, two classes are signature equivalent if they can undergo exactly the same events.

\[
\begin{align*}
\equiv = \_ : \text{ClassStruct} \leftrightarrow \text{ClassStruct} \\
\forall c_1, c_2 : \text{ClassStruct} \\
\quad c_1 = c_2 \iff \text{dom}(c_1.\text{trans}) = \text{dom}(c_2.\text{trans})
\end{align*}
\]
Given a class \( A \) whose structural model is denoted by \( A \), let the structural model of the class \( C[A] \), where \( C \) is a context, be denoted by \( C[A] \). A model \( D \) of classes is fully abstract with respect to the trace model if, for all structural models \( c_1 \) and \( c_2 \) such that \( c_1 \equiv c_2 \), the following holds for all contexts \( C \) in which the classes corresponding to \( c_1 \) and \( c_2 \) can be placed.

\[
D(c_1) = D(c_2) \Leftrightarrow T(C[c_1]) = T(C[c_2])
\]

4. Preliminary Models

Intuitively, the history model of classes presented in Section 2.3 is not fully abstract with respect to the trace model of Section 3.2 as it contains information about the internal state of objects which cannot be accessed within any context. For example, two classes which are identical except for the name of a particular state variable could not be distinguished by any context but would be given different semantic denotations under the history model.

The history model is, however, compositional with respect to the trace model. The ways in which a component object can be referred to in a class are limited by the syntax of Object-Z to \( a.1 \), \( \text{pre} \ a.\text{op} \) and \( a.\text{op} \) where \( a \) refers to the object and \( \text{op} \) is an operation. Since each of these constructs are defined only in terms of the histories of the object’s class (as detailed in Section 2.3), the set of histories, and hence the set of traces, of any class can be derived from the sets of histories of its components.

A fully abstract model of classes, therefore, can be derived from the history model by removing exactly that internal state information not required for composition. In this section, two models of classes derived from the history model are presented as motivation for a fully abstract model of classes presented in Section 5. The first model, presented in Section 4.1, is the trace model of Section 3.2.

It is shown, by means of a counter-example, that this model is not compositional for classes with nondeterministic operations. The second model, presented in Section 4.2, attempts to overcome this problem by associating with each trace the operations which are enabled immediately after an object has undergone the sequence of events of the trace. This model is similar to the readiness model of Olderog and Hoare [OlH86] and the closely related failures model of Brookes et al. [BRH84] which has been adopted as the semantics of CSP [Hoa85]. It is shown, however, that this model is also not compositional with respect to the trace model for Object-Z classes.

4.1. Trace Model

A model \( D \) of classes is defined to be compositional if, for all structural models \( c_1 \) and \( c_2 \) such that \( c_1 \equiv c_2 \), the following holds for all contexts \( C \) in which the classes corresponding to \( c_1 \) and \( c_2 \) can be placed.

\[
D(c_1) = D(c_2) \Rightarrow D(C[c_1]) = D(C[c_2])
\]

A model \( D \) of classes is defined to be compositional with respect to the trace model of classes if, for all structural models \( c_1 \) and \( c_2 \) such that \( c_1 \equiv c_2 \), the following holds for all contexts \( C \) in which the classes corresponding to \( c_1 \) and \( c_2 \) can be placed.

\[
D(c_1) = D(c_2) \Rightarrow T(C[c_1]) = T(C[c_2])
\]
Equivallently, the following two predicates must hold (see [Jon89]).

\[ D(c_1) = D(c_2) \Rightarrow D(C[c_1]) = D(C[c_2]) \]
\[ D(c_1) = D(c_2) \Rightarrow T(c_1) = T(c_2) \]

The first predicate states that the model \( D \) is compositional. The second predicate states that the model \( D \) is at least as distinguishing as the trace model \( T \). Therefore, a model of classes which is fully abstract with respect to the trace model must be at least as distinguishing as the trace model. The trace model itself, therefore, provides the minimum candidate for a fully abstract model.

The trace model, however, can be shown not to be compositional when non-determinism is allowed within classes. For example, consider the following signature equivalent Object-Z classes. The operations \( X \) and \( Y \) in class \( B \) are non-deterministic as they have more than one post-state for a given pre-state.

State transition diagrams of these classes are shown in Fig. 3.

The trace models of these classes are identical. That is, each class is represented by the set of all traces made up of events corresponding to the operations \( X \) and \( Y \). Let \( A \) and \( B \) denote the structural models of classes \( A \) and \( B \) respectively. The traces of \( A \) and \( B \) can be defined formally as follows\(^7\).

\[ T(A) = T(B) = \{ t : \text{Trace} \mid \text{ran} t \subseteq \{ (X', 0), (Y', 0) \} \} \]

\(^7\) The events corresponding to the operations \( X \) and \( Y \) are represented as tuples consisting of the operation’s name and an assignment of values to the operation’s parameters (as detailed...
The operation $X$, however, is always enabled for objects of class $A$ and only sometimes enabled for objects of class $B$. Therefore, a context which allows, for example, operation $Y$ to occur only when operation $X$ is not enabled and allows operation $X$ to occur otherwise can be used to distinguish classes $A$ and $B$. This context can be represented as follows.

$$
\begin{array}{c}
C \\
\begin{array}{c}
\text{a : } \square \\
\hline \\
I \\
\text{a, I} \\
\hline \\
\text{OpX} \equiv a.X \\
\text{OpY} \\
\text{a, Y} \\
\hline \\
\sim \text{ pre OpX}
\end{array}
\end{array}
$$

The traces of the class $C[A]$ consist of events corresponding to the operation $OpX$ only.

$$
T(C[A]) = \{ t : \text{Trace} \mid \text{ran } t \subseteq \{ (\text{OpX}, ) \} \}
$$

The traces of the class $C[B]$, however, begin with an event corresponding to the operation $OpX$ and then continue with events corresponding to either $OpX$ or $OpY$.

$$
T(C[B]) = \{ t : \text{Trace} \mid t \neq \{ \} \Rightarrow \text{head } t = (\text{OpX}, ) \land \\
\text{ran } t \subseteq \{ (\text{OpX}, ), (\text{OpY}, ) \} \}
$$

Since $A$ and $B$ can be distinguished by the context $C$, the trace model of classes is not compositional.

### 4.2. Readiness Model

The model of classes presented in this section represents an object by the sequence of events it has undergone together with the set of events that it is now ready to perform. The inclusion of this set of events, called the ready set of the object, allows the model to distinguish between classes such as $A$ and $B$ of Section 4.1.

The model is similar to the readiness model of Oldberg and Hoare [OIH86] and the closely related failures model of Brookes et al. [BHR84]. The failures model, however, associates with a trace a set of events which can be refused after the trace, rather than the set of events which are enabled after the trace. It is also less distinguishing than the readiness model of Oldberg and Hoare and the model presented in this section as it associates with each trace of an object (or process, using the terminology of [BHR84]) not only the sets of events it can

\( \text{in Section 2.2). Since } X \text{ and } Y \text{ have no parameters, the associated assignment of values is denoted by the empty set.} \)
next refuse, but all subsets of these sets. Therefore, while the failures model of an object can be derived from its readiness model, the readiness model of an object cannot necessarily be derived from its failures model.

The failures model and the readiness model of Olderog and Hoare do not include traces corresponding to infinite sequences of events\(^8\) and cannot, therefore, distinguish between systems whose finite behaviour is the same, but whose liveness properties are different. The readiness model presented in this section, however, does include infinite traces. As mentioned, this is necessary for the model to be also fully abstract for Object-Z classes with history invariants (see [Smi92] for details).

A *ready-behaviour* is modelled as a (possibly infinite) sequence of events \( \text{events} \) and a set of events \( \text{ready} \) representing the events which an object is ready to perform after undergoing the events in \( \text{events} \). If \( \text{events} \) is infinite then \( \text{ready} \) is the empty set. Adopting the definition of \( \text{Event} \) from Section 2.2, a ready-behaviour can be specified as follows.

\[
\begin{align*}
\text{ReadyBehaviour} \\
\text{events} : \text{seq} \: \text{Event} \\
\text{ready} : \text{Event} \\
\text{events} \notin \text{seq Event} \Rightarrow \text{ready} =
\end{align*}
\]

The set of events which are ready to occur after an object of a class with structural model \( c \) has undergone a history \( h \) is given by the function \( \text{next} \) defined below.

\[
\begin{align*}
\text{next} : \text{ClassStruct} \times \text{History} \rightarrow \text{Event} \\
\text{dom} \: \text{next} &= \{(c, h) : \text{ClassStruct} \times \text{History} \mid h \in \mathcal{H}(c)\} \\
\forall (c, h) : \text{dom} \: \text{next} \bullet \\
&\quad h.\text{events} \notin \text{seq Event} \Rightarrow \text{next}(c, h) = \text{\texttt{empty set}} \\
&\quad h.\text{events} \in \text{seq Event} \Rightarrow \\
&\quad \text{next}(c, h) = \{e : \text{Event} \mid \exists h' : \mathcal{H}(c) \bullet \\
&\quad \quad \text{front} h'.\text{states} = h.\text{states} \land \\
&\quad \quad h'.\text{events} = h.\text{events} \cup \{e\}\}
\end{align*}
\]

The set of ready-behaviours representing a class can be derived from the histories of the class using the function \( \mathcal{R} \) defined below.

\[
\begin{align*}
\mathcal{R} : \text{ClassStruct} \rightarrow \text{ReadyBehaviour} \\
\forall c : \text{ClassStruct} \bullet \\
\mathcal{R}(c) &= \{r : \text{ReadyBehaviour} \mid \exists h : \mathcal{H}(c) \bullet \\
&\quad r.\text{events} = h.\text{events} \land \\
&\quad r.\text{ready} = \text{next}(c, h)\}
\end{align*}
\]

The ready-behaviours of the class \( A \) of Section 4.1 can be defined formally as follows. (\( A \) denotes the structural model of class \( A \).)

\[
\mathcal{R}(A) = \{r : \text{ReadyBehaviour} \mid \text{ran} \: r.\text{events} \subseteq \{(X', \_), (Y', \_), 0\} \land \\
&\quad r.\text{events} \in \text{seq Event} \Rightarrow \\
&\quad r.\text{ready} = \{(X', \_), (Y', \_), 0\}\}
\]

\(^8\) The failures model has been extended to include a component of infinite traces by Roscoe and Barrett [Bol90].
This is different from the ready-behaviours of class \(B\) of Section 4.1 which can be defined formally as follows. (\(B\) denotes the structural model of class \(B\).)

\[
\mathcal{R}(B) = \{ r : \text{ReadyBehaviour} \mid \text{ran } r.\text{events} \subseteq \{(X', ), (Y', )\} \land \\
( r.\text{events} \in \text{seq Event} \Rightarrow \\
\forall r.\text{ready} \in \left\{ \{(X', ), (Y', )\}, \{(Y', )\} \right\} \land \\
 r.\text{events} = \{} \Rightarrow r.\text{ready} = \{(X', ), (Y', )\} \}\}
\]

Hence, the readiness model can distinguish between these classes as desired. The readiness model, however, can be shown not to be compositional. For example, consider the following signature equivalent Object-Z classes.

\[
\begin{align*}
\mathcal{D} = & \{ s, X, Y \} \\
\mathcal{E} = & \{ s, X, Y \}
\end{align*}
\]

\[
\begin{align*}
s : & \{0, 1, 2\} \\
I = & \{ s = 0 \} \\
\Delta(s) = & \begin{cases} 
\{0, 2\} & s = 0 \\
\{2\} & s = 2
\end{cases} \\
\Delta(s) = & \begin{cases} 
\{0, 1, 2\} & s = 0 \\
\{1, 2\} & s = 2
\end{cases}
\end{align*}
\]

\[
\begin{align*}
s : & \{3, 4\} \\
I = & \{ s = 3 \} \\
\Delta(s) = & \begin{cases} 
\{3\} & s = 3 \\
\{3, 4\} & s = 3
\end{cases} \\
\Delta(s) = & \begin{cases} 
\{3, 4\} & s = 3 \\
\{3, 4\} & s = 3
\end{cases}
\end{align*}
\]

State transition diagrams of these classes are shown in Fig. 4.

The readiness models of these classes are identical. That is, initially an object of either class is ready to perform an \(X\) and a \(Y\) operation and, after performing any sequence of events, is ready to perform either an \(X\) and a \(Y\) operation or just a \(Y\) operation. Let \(D\) and \(E\) denote the structural models of classes \(D\) and \(E\).
$E$ respectively. The ready-behaviours of $D$ and $E$ can be defined formally as follows.

$$R(D) = R(E) = \{ r : \text{ReadyBehaviour} \mid \begin{array}{l}
\text{ran } r.\text{events} \subseteq \{(\langle X', \rangle, \langle Y', \rangle)\} \land \\
\text{ran } r.\text{events} \subseteq \text{seq Event} \Rightarrow \\
\text{r.ready} \in \{(\langle X', \rangle, \langle Y', \rangle), \langle \langle Y', \rangle \rangle, \langle \langle Y', \rangle \rangle \}) \land \\
r.\text{events} = \langle \rangle \Rightarrow \text{r.ready} = \{(\langle X', \rangle, \langle Y', \rangle)\}\}
\end{array} \}$$

The operation $X$, however, can only be refused at most once for an object of class $D$ but can be refused many times for an object of class $E$. Therefore, the context $C$ of Section 4.1, which allows operation $Y$ to occur only when operation $X$ is not enabled and allows operation $X$ to occur otherwise, can be used to distinguish classes $D$ and $E$.

The traces of $C[D]$ begin with an event corresponding to an occurrence of the operation $OpX$ and may have at most one subsequent event corresponding to an occurrence of the operation $OpY$.

$$T(C[D]) = \{ t : \text{Trace} \mid t \neq \langle \rangle \Rightarrow \text{head } t = (\langle OpX', \rangle) \land \begin{array}{l}
\text{ran } t \subseteq \{(\langle OpX', \rangle, \langle OpY', \rangle)\} \land \\
\#(t) \geq 1 \}
\end{array} \}$$

The traces of $C[E]$, however, begin with an event corresponding to the occurrence of the operation $OpX$ and may have (possibly infinitely) many subsequent events corresponding to occurrences of the operation $OpY$.

$$T(C[E]) = \{ t : \text{Trace} \mid t \neq \langle \rangle \Rightarrow \text{head } t = (\langle OpX', \rangle) \land \begin{array}{l}
\text{ran } t \subseteq \{(\langle OpX', \rangle, \langle OpY', \rangle)\} \}
\end{array} \}$$

Since $D$ and $E$ can be distinguished by the context $C$, the readiness model of classes is not compositional with respect to the trace model.

5. Fully Abstract Model

A fully abstract model of classes describes a class in terms of the external behaviour of its objects. It therefore captures the meaning of a class independent of its syntactic representation. In this section, a model of classes in Object-Z is presented which is fully abstract with respect to the trace model of Section 3.2.

The model, called the complete-readiness model, represents an object by the sequence of events it has undergone together with the sequence of ready sets at each stage of its evolution. The inclusion of the past ready sets, as well as the current ready set, of the object allows the model to distinguish between classes such as $D$ and $E$ of Section 4.2.

The model is presented in Section 5.1. It is proved, in Section 5.2, to be compositional with respect to the trace model and, in Section 5.3, to be fully abstract with respect to the trace model.

5.1. Complete-Readiness Model

The complete-readiness model represents a class by a set of behaviours. A behaviour represents the sequence of events an object has undergone together with the sequence of ready sets at each stage of its evolution. It can be modelled as
a sequence of events \( events \) and a non-empty sequence of ready sets \( readys \) such that the number of ready sets in \( readys \) is one more than the number of events in \( events \) unless \( readys \) is infinite in which case \( events \) is also infinite. Adopting the definition of \( Event \) from Section 2.2, a behaviour can be specified as follows.

Let \( \text{prehist} \) be defined as in Section 2.3 and the function \( \text{next} \) as in Section 4.2. The behaviour of an object of a class with structural model \( c \) can be derived from its history using the function \( \text{behav}(c) \) defined below.

\[
\begin{align*}
\text{behav}: \text{ClassStruct} & \to (\text{History} \to \text{Behaviour}) \\
\forall c : \text{ClassStruct} & \bullet \\
\text{dom}\, \text{behav}(c) & = \mathcal{H}(c) \\
\forall h : \text{dom}\, \text{behav}(c); b : \text{Behaviour} & \bullet \\
\text{behav}(c)(h) & = b \iff \\
b.\, \text{events} & = h.\, \text{events} \\
\forall i : \text{dom}\, b.\, \text{readys}; ph : \text{prehist}(h) & \bullet \\
\#ph.\, \text{states} & = i \Rightarrow \\
b.\, \text{readys}(i) & = \text{next}(c, ph)
\end{align*}
\]

The set of behaviours representing a class can be derived from the histories of the class using the function \( \mathcal{CR} \) defined below.

\[
\mathcal{CR}: \text{ClassStruct} \to \text{Behaviour} \\
\forall c : \text{ClassStruct} \bullet \mathcal{CR}(c) = \text{behav}(c)[\mathcal{H}(c)]
\]

The behaviours of the class \( D \) of Section 4.2 can be defined formally as follows. (\( D \) denotes the structural model of class \( D \).)

\[
\mathcal{CR}(D) = \{ b : \text{Behaviour} \mid \text{ran} \, b.\, \text{events} \subseteq \{ (X^*, ) , (Y^*, ) \} \land \\
\text{ran} \, b.\, \text{readys} \subseteq \{ (X^*, ) , (Y^*, ) , (Y^*, ) \} \land \\
b.\, \text{readys}(1) = \{ (X^*, ) , (Y^*, ) \} \land \\
\#(b.\, \text{readys} \triangleright \{ (Y^*, ) \}) \leq 1 \}
\]

This is different from the behaviours of class \( E \) of Section 4.2 which can be defined formally as follows. (\( E \) denotes the structural model of class \( E \).)

\[
\mathcal{CR}(E) = \{ b : \text{Behaviour} \mid \text{ran} \, b.\, \text{events} \subseteq \{ (X^*, ) , (Y^*, ) \} \land \\
\text{ran} \, b.\, \text{readys} \subseteq \{ (X^*, ) , (Y^*, ) , (Y^*, ) \} \land \\
b.\, \text{readys}(1) = \{ (X^*, ) , (Y^*, ) \} \}
\]

That is, an object of class \( D \) can refuse to perform an \( X \) operation only once whereas an object of class \( E \) can refuse to perform an \( X \) operation (possibly infinitely) many times. Hence, the complete-readiness model can distinguish these classes as desired.

The proof of full abstraction of the complete-readiness model in the following sections relies on the fact that the precondition of an operation \( op \) in Object-Z
can be tested at any time, i.e. by a statement of the form pre \( op \). This ability, inherited from Z, allows the specification of a much wider range of systems than would otherwise be possible.

For example, Object-Z allows an object to be placed in an environment which operates it according to some priority. The context \( C \) of Section 4.1 which allows operation \( Y \) to occur only when operation \( X \) is not enabled and allows operation \( X \) to occur otherwise is an example of such an environment. Since priority constructs are included in some programming languages, such as occam [JoG88], it is desirable to be able to capture the notion of priority in a specification language. Indeed, alternatives to the failures semantics of CSP have been proposed for this purpose (see e.g. [Fid92]).

The ability to test the precondition of an operation also allows the specification of systems in which an object can perform an operation at any time but must perform it in synchronisation with another object performing an operation whenever that other object can also perform its operation. This notion of composition has been suggested as an alternative to the standard parallel composition operator of process algebras by Pnueli [Pnu85]. Pnueli shows that the failures model is not compositional when this type of composition is allowed and suggests a model similar to the complete-readiness model as an alternative.

### 5.2. Proof of Compositionality

The complete-readiness model of classes is compositional with respect to the trace model if, for all structural models \( c_1 \) and \( c_2 \) such that \( c_1 \neq c_2 \), the following holds for all contexts \( C \) in which the classes corresponding to \( c_1 \) and \( c_2 \) can be placed.

\[
C \mathcal{R}(c_1) = C \mathcal{R}(c_2) \Rightarrow T(C[c_1]) = T(C[c_2])
\]

In this section, a proof of compositionality is given for any context \( C \) which includes a single object \( a \) of its elided class as a state variable. The proof could be generalised to also include contexts with multiple objects of the elided class or with aggregates of objects of the elided class. These generalisations are not, however, discussed in this paper.

The proof relies on the fact that the traces of \( C[A] \), for any class \( A \), can be derived from the set of behaviours of \( A \). The ways in which the object \( a \) can be referred to within \( C[A] \) are limited by the syntax of Object-Z to \( a.I \), pre \( a.op \) and \( a.op \) where \( op \) is an operation of class \( A \). The proof, therefore, requires the meanings of these notations to be defined in terms of the complete-readiness model in such a way that the set of traces of \( C[A] \) can be derived.

Schema definitions of the notations \( a.I \) and \( a.op \) are given in Section 2.3. Based on these definitions, the meanings of the notations \( a.I \), pre \( a.op \) and \( a.op \) in terms of the history model are as follows.

- An object \( a \) of a class with structural model \( c \) satisfies the predicate of \( a.I \) if and only if its history is in the set \( h_{\text{init}}(c) \) defined below.

\[
h_{\text{init}}(c) = \{ h : \mathcal{H}(c) \mid h.\text{events} = \langle \rangle \}
\]

- An object \( a \) of a class with structural model \( c \) satisfies the predicate pre \( a.op \), for a particular assignment of values to the parameters of the operation \( op \),
if and only if its history is in the set \( h_{\text{pre}}(c, e) \), where \( e \) is the event corresponding to the occurrence of \( op \) given the parameter values.

\[
h_{\text{pre}}(c, e) = \{ h : \mathcal{H}(c) \mid h.\text{events} \in \text{seq Event} \land \exists h' : \mathcal{H}(c) \cdot \text{front h'.states} = h.\text{states} \land h'.\text{events} = h.\text{events} \cup \langle e \rangle \}
\]

- The objects \( a \) and \( a' \) of a class with structural model \( c \) satisfy the pre-state and associated post-state of an operation \( a.\text{op} \), for a particular assignment of values to the parameters of the operation \( op \), if and only if the tuple consisting of the histories of \( a \) and \( a' \) is in the set \( h_{\text{trans}}(c, e) \), where \( e \) is the event corresponding to the occurrence of \( op \) given the parameter values.

\[
h_{\text{trans}}(c, e) = \{(h, h') : \mathcal{H}(c) \times \mathcal{H}(c) \mid h.\text{events} \in \text{seq Event} \land \text{front h'.states} = h.\text{states} \land h'.\text{events} = h.\text{events} \cup \langle e \rangle \}
\]

The meanings of the notations \( a.\text{I} \), \( \text{pre a.\text{op}} \) and \( a.\text{op} \) can also be defined in terms of the complete-readiness model of classes. Given that an object is instantiated from the behaviours of its class, the notation \( a.\text{I} \), where \( a \) is an object, can be represented semantically by the following schema.

\[
\begin{array}{l}
\text{a.} \text{I} \\
\text{a.\text{events} = \langle \rangle }
\end{array}
\]

Similarly the notation \( a.\text{op} \), where \( a \) is an object and \( op \) an operation in \( a \)'s class with an input parameter \( \text{in?} : \text{In} \) and an output parameter \( \text{out!} : \text{Out} \), can be represented semantically by the following schema.

\[
\begin{array}{l}
\text{a.\text{op}} \\
\Delta(a) \\
\text{\text{in?} : \text{In}} \\
\text{\text{out!} : \text{Out}} \\
\text{a.\text{events} \in \text{seq Event}} \\
\text{\text{front a'.readys} = a.\text{readys}} \\
\text{a'.\text{events} = a.\text{events} \cup \langle \langle '\text{op}', \text{\{'in?' \mapsto \text{in?}, '\text{out!}' \mapsto \text{out!}'} \rangle \rangle \rangle }
\end{array}
\]

Therefore, the meanings of the notations \( a.\text{I} \), \( \text{pre a.\text{op}} \) and \( a.\text{op} \) in terms of the complete-readiness model are as follows.

- An object \( a \) of a class with structural model \( c \) satisfies the predicate of \( a.\text{I} \) if and only if its behaviour is in the set \( h_{\text{init}}(c) \) defined below.

\[
h_{\text{init}}(c) = \{ b : \text{CR}(c) \mid b.\text{events} = \langle \rangle \}
\]

- An object \( a \) of a class with structural model \( c \) satisfies the predicate \( \text{pre a.\text{op}} \), for a particular assignment of values to the parameters of the operation \( op \),
if and only if its behaviour is in the set \( b_{\text{pre}}(c, e) \), where \( e \) is the event corresponding to the occurrence of \( op \) given the parameter values.

\[
b_{\text{pre}}(c, e) = \{ b : CR(c) \mid b.\text{events} \in \text{seq Event} \land \\
\exists b' : CR(c) \cdot \\
\text{front } b'.\text{readys} = b.\text{readys} \land \\
b'.\text{events} = b.\text{events} \setminus \{ e \} \}
\]

• The objects \( a \) and \( a' \) of a class with structural model \( c \) satisfy the pre-state and associated post-state of an operation \( a, op \), for a particular assignment of values to the parameters of the operation \( op \), if and only if the tuple consisting of the behaviours of \( a \) and \( a' \) is in the set \( b_{\text{trans}}(c, e) \), where \( e \) is the event corresponding to the occurrence of \( op \) given the parameter values.

\[
b_{\text{trans}}(c, e) = \{ (b, b') : CR(c) \times CR(c) \mid b.\text{events} \in \text{seq Event} \land \\
\text{front } b'.\text{readys} = b.\text{readys} \land \\
b'.\text{events} = b.\text{events} \setminus \{ e \} \}
\]

Given a class \( A \) with structural model \( A \) and a context \( C \) which includes a single object \( a \) of its elided class as a state variable, consider the following two methods for interpreting constructs in \( C[A] \).

**Method 1** All component objects of \( C[A] \), including \( a \), are instantiated from the set of histories of their classes. All constructs involving these objects are interpreted using the meanings of the constructs in terms of the history model.

**Method 2** The object \( a \) is instantiated from the set of behaviours of its class. All other component objects of \( C[A] \) are instantiated from the set of histories of their classes. All constructs involving \( a \) are interpreted using the meanings of the constructs in terms of the complete-readiness model. All constructs involving other component objects are interpreted using the meanings of the constructs in terms of the history model.

Notice that the set of histories of \( C[A] \) derived using Method 1 will be \( \mathcal{H}(C[A]) \) and, hence, the set of traces of \( C[A] \) derived using Method 1 will be \( \mathcal{T}(C[A]) \). To show that the complete-readiness model is compositional with respect to the trace model, therefore, it is sufficient to show that the set of traces of \( C[A] \) derived using Method 2 is identical to the set of traces of \( C[A] \) derived using Method 1. In order to do this, consider the following preliminary definitions which allow a history of \( C[A] \) derived using Method 1 to be related to a history of \( C[A] \) derived using Method 2.

Given a state \( s \) of \( C[c] \), where \( c \) is the structural model of a class, the state which is identical to \( s \) but with the history of an object \( a \) replaced with the behaviour of \( a \) can be derived using the function \( s_{\text{map}}(c) \) defined below.

\[
s_{\text{map}} : \text{ClassStruct} \to (\text{State} \to \text{State})
\]

\[
\forall c : \text{ClassStruct} \cdot \\
\text{dom } s_{\text{map}}(c) = \{ s : \text{State} \mid 'a' \in \text{dom } s \land s('a') \in \mathcal{H}(c) \} \\
\forall s : \text{dom } s_{\text{map}}(c) \cdot \\
s_{\text{map}}(c)(s) = s \oplus \{ 'a' \mapsto \text{behav}(c)(s('a')) \}
\]

Given a history \( h \) of \( C[c] \), the history which is identical to \( h \) but with the
history of a in each state replaced with the behaviour of a can be derived using
the function \( h_{\text{map}}(c) \) defined below:

\[
\begin{align*}
\forall c : \text{ClassStruct} & \quad \bullet \\
\quad \text{dom} \ h_{\text{map}}(c) &= \{ h : \text{History} \mid \text{ran} h.\text{states} \subseteq \text{dom} s_{\text{map}}(c) \} \\
\quad \forall h : \text{dom} \ h_{\text{map}}(c) & \quad \bullet \\
\quad h_{\text{map}}(c)(h).\text{events} &= h.\text{events} \\
\quad \forall i : \text{dom} h.\text{states} & \quad \bullet \\
\quad h_{\text{map}}(c)(h).\text{states}(i) &= s_{\text{map}}(c)(h.\text{states}(i))
\end{align*}
\]

To prove that the complete-readiness model is compositional with respect
to the trace model, it is sufficient to prove that for every history \( h_1 \) of \( C[A] \)
derived using Method 1, the history \( h_{\text{map}}(A)(h_1) \) is a history of \( C[A] \) derived
using Method 2 and, for every history \( h_2 \) of \( C[A] \) derived using Method 2, there
exists a history in \( h_{\text{map}}(A)^- = \{ \{ h_2 \} \} \) which is a history of \( C[A] \) derived using
Method 1. Since \( h_{\text{map}}(A) \) preserves the trace, i.e. the sequence of events, of a
history, it follows that the set of traces derived using Method 2 are the same as
those derived using Method 1.

A proof of compositionality is given below.

**Theorem 5.1.** Let \( A \) be a class and \( A \) denote its structural model. Let \( C \) be a
context which includes a single object \( a \) of its elided class as a state variable such
that \( A \) can be placed in \( C \). Let \( H_1 \) be the set of histories of \( C[A] \) derived using
Method 1, i.e. \( H_1 = \mathcal{H}(C[A]) \), and \( H_2 \) be the set of histories of \( C[A] \) derived
using Method 2.

The following predicates are true.

(a) \( \forall h_1 : H_1 \bullet h_{\text{map}}(A)(h_1) \in H_2 \)
(b) \( \forall h_2 : H_2 \bullet \exists h_1 : H_1 \bullet h_1 \in h_{\text{map}}(A)^- = \{ \{ h_2 \} \} \)

**Proof.**
(a) The proof is by induction over the length of \( h_1.\text{events} \).

(i) If \( \# h_1.\text{events} = 0 \) then the state \( h_1.\text{states}(1) \) satisfies the predicate of
the initial state schema of \( C[A] \) using Method 1. Hence, \( s_{\text{map}}(A)(h_1.\text{states}(1)) \)
satisfies the predicate of the initial state schema of \( C[A] \) using Method 2 by
Lemma 3.9 Therefore, there exists a history \( h_2 \) in \( H_2 \) such that \# \( h_2.\text{events} = 0 \)
and \( h_2.\text{states}(1) = s_{\text{map}}(A)(h_1.\text{states}(1)) \). That is, \( h_{\text{map}}(A)(h_1) \in H_2 \).

(ii) Assume \( h_{\text{map}}(A)(h_1) \in H_2 \) for all \( h_1 \) such that \# \( h_1.\text{events} = n \) for some
\( n > 0 \).

If \( \# h_1.\text{events} = n + 1 \) then \( h_1.\text{states}(n + 1), h_1.\text{states}(n + 2) \) is a state
transition of the event \( h_1.\text{events}(n + 1) \) using Method 1. Hence, the state transition
\( s_{\text{map}}(A)(h_1.\text{states}(n + 1)), s_{\text{map}}(A)(h_1.\text{states}(n + 2)) \) is a transition of
\( h_1.\text{events}(n + 1) \) using Method 2 by Lemma 6(a).

Since all pre-histories of \( h_1 \) are in \( H_1 \), there exists a history \( ph_1 \) in \( H_1 \) such
that \( ph_1 \in \text{prehist}(h_1) \) and \# \( ph_1.\text{events} = n \). Therefore, there exists a history
\( ph_2 \) in \( H_2 \) such that \( ph_2 = h_{\text{map}}(A)(\{ ph_1 \}) \) by the above assumption. Therefore,
there exists a \( h_2 \) (extending \( ph_2 \) in \( H_2 \) such that \( h_2 = h_{\text{map}}(A)(h_1) \). Hence,
\( h_{\text{map}}(A)(h_1) \in H_2 \) for all \( h_1 \) such that \# \( h_1.\text{events} = n + 1 \).

The lemmas required for this proof are included in the Appendix.
(b) The proof is by induction over the length of \( h_2.event \).

(i) If \( \# h_2.event = 0 \) then the state \( h_2.states(1) \) satisfies the predicate of the initial state schema of \( C[A] \) using Method 2. Hence, all states in the set of states \( s_map(A)^-\{h_2.states(1)\} \| \) satisfy the predicate of the initial state schema of \( C[A] \) using Method 3. Also, since \( CR(A) = behave(A) \| H \| \) and the behaviour of \( a \) in \( h_2.states(1) \) is in \( CR(A) \), \( s_map(A)^-\{h_2.states(1)\} \| \) is not the empty set. Therefore, there exists a \( h_1 \) in \( H_1 \) such that \( \# h_1.event = 0 \) and \( h_1.states(1) \) is in \( s_map(A)^-\{h_2.states(1)\} \| \). That is, there is a \( h_1 \) in \( H_1 \) such that \( h_1 \in h_map(A)^-\{h_2\} \| \).

(ii) Assume there is a \( h_1 \) in \( H_1 \) such that \( h_1 \in h_map(A)^-\{h_2\} \| \) for all \( h_1 \) such that \( \# h_1.event \geq n \) for some \( n \geq 0 \).

If \( \# h_2.event = n + 1 \) then \( (h_2.states(n + 1), h_2.states(n + 2)) \) is a state transition of the event \( h_2.events(n + 1) \) using Method 2. Hence, for a given state \( s' \) in the set of states \( s_map(A)^-\{h_2.states(n + 2)\} \| \), there exists a state \( s \) in the set of states \( s_map(A)^-\{h_2.states(n + 1)\} \| \) such that \( (s, s') \) is a transition of \( h_2.events(n + 1) \) using Method 1 by Lemma 6(b). Also, since the behaviour of \( a \) in \( h_2.states(n + 2) \) is in \( CR(A) \), \( s_map(A)^-\{h_2.states(n + 2)\} \| \) is not the empty set.

Since all pre-histories of \( h_2 \) are in \( H_2 \), there exists a history \( ph_2 \) in \( H_2 \) such that \( ph_2 \in prehist(h_2) \) and \( \# ph_2.event = n \). Therefore, by the assumption, there exists a history \( ph_1 \) in \( H_1 \) such that \( ph_1 \in h_map(A)^-\{ph_2\} \| \). Hence, by Lemma 7, there exists a history \( ph_1' \) in \( H_1 \) such that \( ph_1' \in h_map(A)^-\{ph_1\} \| \) and \( h_map(A)(ph_1') = h_map(A)(ph_1) \). Therefore, there exists an \( h_1 \) (which extends \( ph_1' \)) in \( H_1 \) such that \( h_1 \in h_map(A)^-\{h_2\} \| \). Therefore, there is a \( h_1 \) in \( H_1 \) such that \( h_1 \in h_map(A)^-\{h_2\} \| \) for all \( h_2 \) such that \( \# h_2.event = n + 1 \).

5.3. Proof of Full Abstraction

Given that the complete-readiness model is compositional with respect to the trace model, it is also fully abstract with respect to the trace model if, for all structural models \( c_1 \) and \( c_2 \) such that \( c_1 = c_2 \), if \( T(C[c_1]) = T(C[c_2]) \), for all contexts \( C \) in which the classes corresponding to \( c_1 \) and \( c_2 \) can be placed, then \( CR(c_1) = CR(c_2) \).

This property states that the complete-readiness model only distinguishes classes when that distinction is necessary for compositionality. Its proof relies on the fact that given any two signature equivalent classes \( A \) and \( B \) with structural models \( A \) and \( B \) respectively, if \( CR(A) \neq CR(B) \) then a context \( C \) can be constructed such that the traces of \( C[A] \) are different from those of \( C[B] \).

To motivate the following proof, consider again the classes \( D \) and \( E \) of Section 4.2. An object of class \( E \) can undergo particular behaviours which an object of class \( D \) cannot undergo. One such behaviour is the behaviour \( b \) whose sequences of events and ready sets are as follows.

\[ b.events = \{(Y',_1), (Y',_2)\} \]
\[ b.readys = \{\{(X',_1), (Y',_2)\}, \{(Y',_1), (Y',_2)\}\} \]

Consider constructing a context \( C \) which has an operation corresponding to each event in \( b.events \) and each ready set in \( b.readys \) as follows.
The operation \( OP1 \) corresponds to the occurrence of the events in \( b.events \). The operation \( OP2 \) is enabled when the object \( a \) (assumed to be of class \( D \) or \( E \)) is ready to perform exactly those events in the first ready set of \( b.readys \). Similarly, the operation \( OP3 \) is enabled when \( a \) is ready to perform exactly those events in the second and third ready sets of \( b.readys \).

The trace \( (\langle OP1, \rangle, \langle OP1, \rangle, \langle OP3, \rangle, \langle OP1, \rangle, \langle OP3, \rangle) \) is a possible trace of \( C[E] \) since an object of class \( E \) can undergo the behaviour \( b \). The trace is not, however, a possible trace of \( C[D] \). The context \( C \) can, therefore, be used to distinguish classes \( D \) and \( E \).

This method can be generalised for any signature equivalent classes with different complete-readiness models as shown in Theorem 5.2 below.

**Theorem 5.2.** Given two signature equivalent classes \( A \) and \( B \) with structural models \( A \) and \( B \) respectively, if \( CR(A) \neq CR(B) \) then there exists a context \( C \) such that \( T(C[A]) \neq T(C[B]) \).

**Proof.**

1) Let \( C \) have a single state variable \( a \) which is an object of its elided class and which is initialised in its initial state schema.

2) Without loss of generality, assume there is a behaviour \( b \) in \( CR(A) \) which is not in \( CR(B) \).

3) For each event \( e \) in the range of \( b.events \), let \( C \) have an operation which corresponds to \( a \) undergoing that event.

4) For each ready set \( r \) in the range of \( b.readys \), let \( C \) have an operation which has a true postcondition and a precondition that states that each event in \( r \) is enabled and each event of \( A \) not in \( r \) is not enabled.

Let \( t \) be a trace such that

- if \( b.events \) is finite then \( t \) is finite and \( \#t = \#b.readys + \#b.events \), otherwise \( t \) is infinite, and

- for all \( i : \text{dom} \ t \), if \( i \) is odd then \( t(i) \) is the event corresponding to the operation associated with \( b.readys(i + 1)/2 \) (as described in step 4 above) and if \( i \) is even then \( t(i) \) is the event corresponding to the operation associated with \( b.events(i/2) \) (as described in step 3 above).
The trace $t$ will be in $T(C[A])$ but will not be in $T(C[B])$. □

6. Discussion

The fully abstract model of classes in Section 5.1 defines the notion of behavioural equivalence of classes in Object-Z. That is, two classes are behaviourally equivalent when they have exactly the same complete-readiness model. This was shown to be a stronger form of behavioural equivalence than that of CSP which is based on processes having the same failures model. This enables Object-Z to capture notions of priority and alternative notions of composition which cannot be captured in CSP.

The notion of behavioural equivalence is, however, weaker than bisimulation which is the primitive notion of equivalence between processes in CCS [Mil89]. To detect that two processes are not bisimilar, it may be necessary for the environment to make multiple copies of a process at some stage of its evolution and to 'force' the copies to undergo every possible transition which the process could have undergone as its next transition. For example, the processes $P$ and $Q$ in Fig. 5 are not bisimilar. They are, however, equivalent in terms of the complete-readiness model.

Whether bisimulation provides an intuitive notion of behavioural equivalence has been the subject of some debate. The notion of 'forcing' a process to exhibit every possible transition has been criticised by Bloom et al. [BIM88] as it assumes the environment of a process can control its internal nondeterminism. On the other hand, Larsen and Skou [LaS89] argue that non-bisimilar processes can be detected using a notion of probabilistic testing.

An alternative notion of behavioural equivalence is proposed by Bloom et al. which allows multiple copies to be made of processes but does not allow the internal nondeterminism of a process to be controlled. Bloom and Meyer [BIM90] argue that this equivalence is the finest "reasonable" process equivalence. The notion of behavioural equivalence in terms of the complete-readiness model is, however, still weaker than this equivalence. For example, consider the processes $R$ and $S$ in Fig. 6.

These processes are not equivalent according to the notion of behavioural equivalence of Bloom et al. but are equivalent in terms of the complete-readiness model. The discrepancy arises because the environment of an object in Object-Z cannot make copies of the object. Allowing the environment in Object-Z to do this would violate the notion that an object's state is hidden.

Therefore, classes corresponding to the processes $R$ and $S$ need to be semantically identified in a fully abstract semantics of Object-Z. This is precisely the
reason why expressions such as \(a = b\) and \(a \in A\), where \(a\) and \(b\) are objects and \(A\) is a set of objects, are not allowed in the version of Object-Z in this paper. It is also the reason why schema expressions involving the \(Z\) schema operator \(\gamma\) are not allowed in classes. Allowing such expressions enables the following contexts, which distinguish classes corresponding to the processes \(R\) and \(S\), to be constructed.

\[
C
\begin{align*}
&x, y : \Box \\
x &\downarrow I \\
y &\downarrow I \\
OP_1 &\equiv x.a \\
OP_2 &\equiv y.a \\
OP_3 &\equiv \neg (x = y)
\end{align*}
\]

\[
D
\begin{align*}
&x : \Box \\
x &\downarrow I \\
OP_1 &\equiv x.a \\
OP_2 &\equiv x.b \gamma x.c \\
OP_3 &\equiv \neg \text{pre } OP_2
\end{align*}
\]

When an object of a class corresponding to the process \(R\) is placed in the context \(C\) the trace \(\langle \langle OP_1 \rangle , \langle OP_2 \rangle , \langle OP_3 \rangle \rangle\) is possible. This trace is not possible, however, when a class corresponding to the process \(S\) is placed in \(C\).

Similarly, the trace \(\langle \langle OP_1 \rangle , \langle OP_3 \rangle \rangle\) is possible when a class corresponding to the process \(R\) is placed in the context \(D\) but not when a class corresponding to the process \(S\) is placed in \(D\).

7. Conclusion

This paper has presented a fully abstract model of classes for Object-Z. The model contains the minimum amount of information required to enable the denotation of a class to be derived from the denotations of the classes of the objects of which it is composed. Intuitively, the model describes the external behaviour of a class, and, hence, captures its precise meaning independent of its syntactic representation.

The model, called the complete-readiness model, represents an object by the
A Fully Abstract Semantics of Classes for Object-Z

sequence of events it has undergone together with the sequence of sets of events representing the enabled events at each stage of its evolution. The notion of behavioural equivalence it defines is stronger than that of CSP and weaker than that of CCS.

Such a model, by identifying the external behaviour of a class, can aid in developing a concise definition of behavioural compatibility or subtyping in Object-Z. Such a definition could be used as the basis for a theory of class refinement enabling specifications to be refined by separately refining the classes of each of their component objects. Preliminary work in this area can be found in [Smü92].

Acknowledgements

This work was carried out while the author was a PhD student at the University of Queensland, Australia. The author would like to acknowledge the generous financial support provided by the Overseas Telecommunications Corporation (OTC) of Australia and the Commonwealth Government of Australia in the form of postgraduate awards. The author would also like to thank Roger Duke and Cecily Bailes for many stimulating and insightful discussions on this work.

References


Appendix: Proof of Lemmas

This appendix includes the proofs of the lemmas required for the proof of compositionality of the complete-readiness model in Section 5. A denotes a class and
A its structural model. $C$ denotes a context which includes a single object $a$ of its elided class as a state variable.

**Lemma 1.** The history of an object of class $A$ is in the set $h_{\text{init}}(A)$ if and only if its behaviour is in $b_{\text{init}}(A)$.

$$\text{behav}(A) \models h_{\text{init}}(A) \iff b_{\text{init}}(A)$$

**Proof.** From the definition of $\text{behav}(A)$,

$$\text{behav}(A) \models h_{\text{init}}(A) \iff \{ b : CR(A) \mid \exists h : h_{\text{init}}(A) \land \begin{array}{l} b.\text{events} = h.\text{events} \\
\forall i : b.\text{ready} : \text{ph : prehist}(h) \land \#\text{ph.states} = i \Rightarrow \end{array} b.\text{ready}(i) = \text{next}(A, \text{ph}) \}$$

From the definition of $h_{\text{init}}(A)$,

$$\text{behav}(A) \models h_{\text{init}}(A) \iff \{ b : CR(A) \mid \exists h : H(A) \land \begin{array}{l} b.\text{events} = \langle \rangle \land \\
b.\text{events} = h.\text{events} \land \\
\forall i : \text{dom} b.\text{ready} : \text{ph : prehist}(h) \land \#\text{ph.states} = i \Rightarrow \\
b.\text{ready}(i) = \text{next}(A, \text{ph}) \}$$

Since $b.\text{events} = \langle \rangle \land h.\text{events} = \langle \rangle$ is equivalent to $b.\text{events} = \langle \rangle \land b.\text{events} = h.\text{events}$,

$$\text{behav}(A) \models h_{\text{init}}(A) \iff \{ b : CR(A) \mid \exists h : H(A) \land \begin{array}{l} b.\text{events} = \langle \rangle \land \\
b.\text{events} = h.\text{events} \land \\
\forall i : \text{dom} b.\text{ready} : \text{ph : prehist}(h) \land \#\text{ph.states} = i \Rightarrow \\
b.\text{ready}(i) = \text{next}(A, \text{ph}) \}$$

From the definition of $\text{behav}(A)$,

$$\text{behav}(A) \models h_{\text{init}}(A) \iff \{ b : CR(A) \mid b.\text{events} = \langle \rangle \land \\
\exists h : H(A) \land b = \text{behav}(A)(h) \}$$

Since $CR(A) = \text{behav}(A) \models H(A)$,

$$\text{behav}(A) \models h_{\text{init}}(A) \iff \{ b : CR(A) \mid b.\text{events} = \langle \rangle \}$$

From the definition of $h_{\text{init}}(A)$,

$$\text{behav}(A) \models h_{\text{init}}(A) \iff b_{\text{init}}(A)$$

**Lemma 2.** The history of an object of class $A$ is in $h_{\text{pre}}(A, e)$, for a particular event $e$, if and only if its behaviour is in $b_{\text{pre}}(A, e)$.

$$\text{behav}(A) \models h_{\text{pre}}(A, e) \iff b_{\text{pre}}(A, e)$$
Proof. From the definition of $\text{behav}(A)$,

$$\text{behav}(A) \| b_{\text{pre}}(A, e) \| = \{ b : \mathcal{CR}(A) \mid \exists h : b_{\text{pre}}(A, e) \cdot b.\text{events} = h.\text{events} \land \forall i : \text{dom} b.\text{readys}; \; ph : \text{prehist}(h) \cdot \#ph.\text{states} = i \Rightarrow b.\text{readys}(i) = \text{next}(A, ph) \}$$

From the definition of $b_{\text{pre}}(A, e)$,

$$\text{behav}(A) \| b_{\text{pre}}(A, e) \| = \{ b : \mathcal{CR}(A) \mid \exists h : \mathcal{H}(A) \cdot b.\text{events} = h.\text{events} \land \forall i : \text{dom} b.\text{readys}; \; ph : \text{prehist}(h) \cdot \#ph.\text{states} = i \Rightarrow b.\text{readys}(i) = \text{next}(A, ph) \land h.\text{events} \in \text{seq} \; \text{Event} \land \exists h' : \mathcal{H}(A) \cdot \text{behav}(A)(h') = b' \land \text{front} h'.\text{states} = h.\text{states} \land h'.\text{events} = h.\text{events} \land \{ e \} \}$$

Since $\mathcal{CR}(A) = \text{behav}(A) \| \mathcal{H}(A) \|$,

$$\text{behav}(A) \| b_{\text{pre}}(A, e) \| = \{ b : \mathcal{CR}(A) \mid \exists b' : \mathcal{CR}(A) \cdot \exists h : \mathcal{H}(A) \cdot b.\text{events} = h.\text{events} \land \forall i : \text{dom} b.\text{readys}; \; ph : \text{prehist}(h) \cdot \#ph.\text{states} = i \Rightarrow b.\text{readys}(i) = \text{next}(A, ph) \land h.\text{events} \in \text{seq} \; \text{Event} \land \exists h' : \mathcal{H}(A) \cdot \text{behav}(A)(h') = b' \land \text{front} h'.\text{states} = h.\text{states} \land h'.\text{events} = h.\text{events} \land \{ e \} \}$$

From the definition of $\text{behav}(A)$,

$$\text{behav}(A) \| b_{\text{pre}}(A, e) \| = \{ b : \mathcal{CR}(A) \mid \exists b' : \mathcal{CR}(A) \cdot \exists h : \mathcal{H}(A) \cdot b.\text{events} = h.\text{events} \land \forall i : \text{dom} b.\text{readys}; \; ph : \text{prehist}(h) \cdot \#ph.\text{states} = i \Rightarrow b.\text{readys}(i) = \text{next}(A, ph) \land h.\text{events} \in \text{seq} \; \text{Event} \land \exists h' : \mathcal{H}(A) \cdot b'.\text{events} = h'.\text{events} \land \forall i : \text{dom} b'.\text{readys}; \; ph : \text{prehist}(h') \cdot \#ph.\text{states} = i \Rightarrow b'.\text{readys}(i) = \text{next}(A, ph) \land \text{front} h'.\text{states} = h.\text{states} \land h'.\text{events} = h.\text{events} \land \{ e \} \}$$
Since $b'.events = b'.events \land b.events = h.events \land b'.events = h.events \land (e)$ is equivalent to $b'.events = b'.events \land b.events = h.events \land b'.events = b.events \land (e)$, and $b.events = h.events \land b.events \in \text{seq Event}$ is equivalent to $b.events = h.events \land b.events \in \text{seq Event}$.

$$\text{behav}(A) \parallel h.\text{pre}(A, e) \parallel = \{ b : CR(A) \mid b.events \in \text{seq Event} \}$$

$$\exists \theta : CR(A) \bullet$$

$$b'.events = b.events \land (e) \land$$

$$\exists h : H(A) \bullet$$

$$b.events = h.events \land$$

$$\forall i : \text{dom} b.\text{readys}; \text{ph} : \text{prehist}(h) \bullet$$

$$\#\text{ph. states} = i \Rightarrow$$

$$b.\text{readys}(i) = \text{next}(A, \text{ph}) \land$$

$$\exists h' : H(A) \bullet$$

$$\theta.\text{events} = h'.\text{events} \land$$

$$\forall i : \text{dom} b'.\text{readys}; \text{ph} : \text{prehist}(h') \bullet$$

$$\#\text{ph. states} = i \Rightarrow$$

$$b'.\text{readys}(i) = \text{next}(A, \text{ph}) \land$$

$$\text{front } h'.\text{states} = h.\text{states}$$

From the definition of $\text{behav}(A)$,

$$\text{behav}(A) \parallel h.\text{pre}(A, e) \parallel = \{ b : CR(A) \mid b.\text{events} \in \text{seq Event} \}$$

$$\exists \theta : CR(A) \bullet$$

$$b'.\text{events} = b.\text{events} \land (e) \land$$

$$\exists h : H(A) \bullet$$

$$\text{behav}(A)(h) = b \land$$

$$\exists h' : H(A) \bullet$$

$$\text{behav}(A)(h') = b' \land$$

$$\text{front } h'.\text{states} = h.\text{states}$$

Since $\exists h, h' : H(A) \bullet \text{front } h'.\text{states} = h.\text{states} \land \text{behav}(A)(h) = b \land \text{behav}(A)(h') = b' \implies \text{front } \theta.\text{readys} = b.\text{readys}$ (from the definition of $\text{behav}(A)$),

$$\text{behav}(A) \parallel h.\text{pre}(A, e) \parallel = \{ b : CR(A) \mid b.\text{events} \in \text{seq Event} \}$$

$$\exists \theta : CR(A) \bullet$$

$$\text{front } \theta.\text{readys} = b.\text{readys} \land$$

$$b'.\text{events} = b.\text{events} \land (e) \land$$

$$\exists h : H(A) \bullet$$

$$\text{behav}(A)(h) = b \land$$

$$\exists h' : H(A) \bullet$$

$$\text{behav}(A)(h') = b' \land$$

$$\text{front } h'.\text{states} = h.\text{states}$$

Since $H(A)$ includes all pre-histories of any history it contains, for all $h'$ in $H(A)$, there exists a $h : H(A)$ such that $\text{front } h'.\text{states} = h.\text{states}$. Therefore, $\text{front } \theta.\text{readys} = b.\text{readys} \land \exists h, h' : H(A) \bullet \text{front } h'.\text{states} = h.\text{states} \land \text{behav}(A)(h) = b \land \text{behav}(A)(h') = b' \implies \theta$ is equivalent to $\text{front } b'.\text{readys} = b.\text{readys} \land \exists h, h' : H(A) \bullet \text{behav}(A)(h) = b \land \text{behav}(A)(h') = b'$. Therefore,
Since by Lemma 2, Using Method 2, the predicate is true when the behaviour of \(a\) is in \(b_{\text{pre}}(A,e)\).

Using Method 2, the predicate is true when the behaviour of \(a\) is in \(b_{\text{pre}}(A,e)\).

Since by Lemma 2, \(\sigma_{\text{map}}(A)(s)\) satisfies the predicate using Method 1 if and only if \(\sigma_{\text{map}}(A)(s)\) satisfies the predicate using Method 2.

All other predicates in \(C[A]\) can be constructed from the predicates above and, using the above results, can be shown to satisfy the lemma.

**Lemma 4.** If the transition \((h, h')\) is in \(b_{\text{trans}}(A,e)\), for a particular event \(e\), then the transition \((\sigma_{\text{map}}(A)(h), \sigma_{\text{map}}(A)(h'))\) is in \(b_{\text{trans}}(A,e)\).
A Fully Abstract Semantics of Classes for Object-Z

\[ \forall (h, h') : h \in \mathcal{H}(A, e) \rightarrow (\text{behav}(A)(h), \text{behav}(A)(h')) \in h \mathcal{R}_\text{trans}(A, e) \]

**Proof.** If \((h, h')\) is in \(h \mathcal{R}_\text{trans}(A, e)\) then from the definition of \(h \mathcal{R}_\text{trans}(A, e)\),

\[ h.e \in \text{seq Event} \land \quad \text{front } h'.s = h.s \land \quad h'.e = h.e \land \langle e \rangle \]

From the definition of \(\text{behav}(A)\),

\[ \forall i : \text{dom } \text{behav}(A)(h).r = h.e \land \quad \# ph.st = i \Rightarrow \quad \text{behav}(A)(h).r(i) = \text{next}(A, ph) \land \quad \text{behav}(A)(h').e = h'.e \land \quad \forall i : \text{dom } \text{behav}(A)(h').r = h.e \land \quad \# ph.st = i \Rightarrow \quad \text{behav}(A)(h').r(i) = \text{next}(A, ph) \]

Therefore,

\[ \text{behav}(A)(h).e \in \text{seq Event} \land \quad \text{front } \text{behav}(A)(h').r = \text{behav}(A)(h).r \land \quad \text{behav}(A)(h').e = \text{behav}(A)(h).e \land \langle e \rangle \]

From the definition of \(h \mathcal{R}_\text{trans}(A, e)\),

\[ (\text{behav}(A)(h), \text{behav}(A)(h')) \in h \mathcal{R}_\text{trans}(A, e) \]

\[ \square \]

**Lemma 5.** If the transition \((b, b')\) is in \(b \mathcal{R}_\text{trans}(A, e)\), for a particular event \(e\), then for all \(h\) such that \(\text{behav}(A)(h') = b\), there exists an \(h\) such that \(\text{behav}(A)(h) = b\) and the transition \((h, h')\) is in \(h \mathcal{R}_\text{trans}(A, e)\).

\[ \forall (b, b') : b \mathcal{R}_\text{trans}(A, e) \rightarrow \quad \forall h' : \text{behav}(A)(h') = b' \rightarrow \exists h : \text{behav}(A)(h) = b \land (h, h') \in h \mathcal{R}_\text{trans}(A, e) \]

**Proof.** If \((b, b')\) is in \(b \mathcal{R}_\text{trans}(A, e)\) and \(\text{behav}(A)(h') = b'\) then from the definition of \(b \mathcal{R}_\text{trans}(A, e)\) and \(\text{behav}(A)\),

\[ b \in \text{seq Event} \land \quad \text{front } b'.r = b.r \land \quad b'.e = b.e \land \langle e \rangle \land \quad b'.e = b'.e \land \quad \forall i : \text{dom } b'.r = b.r \land \quad \# ph.st = i \Rightarrow \quad b'.r(i) = \text{next}(A, ph) \]

Let \(h\) be a history such that \(h.s = \text{front } h'.s\) and \(h.e = \text{front } h'.e\). Since \(h \in \text{prehist}(b')\), \(h \in \mathcal{H}(A)\) and is therefore in the domain of \(\text{behav}(A)\).

From the definition of \(\text{behav}(A)\), \(\text{behav}(A)(h) = b\). Also, from the definition of \(h \mathcal{R}_\text{trans}(A, e)\), \((h, h')\) is in \(h \mathcal{R}_\text{trans}(A, e)\).  \[ \square \]
Lemma 6.  (a) If state tuple \((s, s')\) satisfies an operation in \(C[A]\) using Method 1 then tuple \((s_{map}(A)(s), s_{map}(A)(s'))\) satisfies the operation using Method 2.

(b) If a state tuple \((t, t')\) satisfies an operation in \(C[A]\) using Method 2 then for all states \(s'\) in \(s_{map}(A)^+ \{ t' \}\), there exists a state \(s\) in \(s_{map}(A)^+ \{ t \}\) such that the tuple \((s, s')\) satisfies the operation using Method 1.

Proof.
1) The operation does not change \(a\). It may, however, refer to \(a\), i.e. it may involve \(a, t\) or \(\text{pre}\ a, \text{op}\).

Using Method 1, \((s, s')\) is a transition of the operation if the history of \(a\) in \(s\) is the same as the history of \(a\) in \(s'\) and the precondition and postcondition of the operation are true for the variables in \(s\) and \(s'\) respectively. Using Method 2, \((t, t')\) is a transition of the operation if the behaviour of \(a\) in \(t\) is the same as the behaviour of \(a\) in \(t'\) and the precondition and postcondition of the operation are true for the assignment of values to the variables in \(t\) and \(t'\) respectively.

(a) Since \(s\) satisfies the operation’s precondition using Method 1, \(s_{map}(A)(s)\) satisfies the precondition using Method 2 by Lemma 3. Also, since \(s'\) satisfies the postcondition of the operation using Method 1, \(s_{map}(A)(s')\) satisfies the postcondition using Method 2. From the definition of \(s_{map}\), if the history of \(a\) is the same in \(s\) and \(s'\) then the behaviour of \(a\) is the same in \(s_{map}(A)(s)\) and \(s_{map}(A)(s')\). Therefore, the operation satisfies part (a) of the lemma.

(b) Since \(t\) satisfies the operation’s precondition using Method 2, all states \(s\) in \(s_{map}(A)^+ \{ t \}\) satisfy the precondition using Method 1 by Lemma 3. Also, since \(t'\) satisfies the postcondition of the operation using Method 2, all \(s'\) in \(s_{map}(A)^+ \{ t' \}\) satisfy the postcondition using Method 1. From the definition of \(s_{map}\), if the behaviour of \(a\) in \(t\) is the same as the behaviour of \(a\) in \(t'\) then given an \(s'\) in \(s_{map}(A)^+ \{ t' \}\), there exists a \(s\) in \(s_{map}(A)^+ \{ t \}\) such that the history of \(a\) in \(s\) is the same as the history of \(a\) in \(s'\). Therefore, the operation satisfies part (b) of the lemma.

2) The operation changes \(a\) arbitrarily, i.e. the operation includes \(a\) in its \(\Delta\)-list but does not refer to \(a'\) in its predicate.

Using Method 1, \((s, s')\) is a transition of the operation if the precondition and postcondition of the operation are true for the assignment of values to the variables in \(s\) and \(s'\) respectively. Using Method 2, \((t, t')\) is a transition of the operation if the precondition and postcondition of the operation are true for the assignment of values to the variables in \(t\) and \(t'\) respectively.

(a) Since \(s\) satisfies the operation’s precondition using Method 1, \(s_{map}(A)(s)\) satisfies the precondition using Method 2 by Lemma 3. Also, since \(s'\) satisfies the postcondition of the operation using Method 1, \(s_{map}(A)(s')\) satisfies the postcondition using Method 2. Therefore, the operation satisfies part (a) of the lemma.

(b) Since \(t\) satisfies the operation’s precondition using Method 2, all states \(s\) in \(s_{map}(A)^+ \{ t \}\) satisfy the precondition using Method 1 by Lemma 3. Also, since \(t'\) satisfies the postcondition of the operation using Method 2, all \(s'\) in \(s_{map}(A)^+ \{ t' \}\) satisfy the postcondition using Method 1. Therefore, the operation satisfies part (b) of the lemma.

3) The operation is \(a, \text{op}\) and the assignment of values to \(a, \text{op}\)’s parameters are such that the occurrence of the operation corresponds to the event \(e\).
Using Method 1, \((s, s')\) is a transition of the operation when the tuple consisting of the history of \(a\) in \(s\) and the history of \(a\) in \(s'\) is in \(h\text{-trans}(A, e)\). Using Method 2, \((t, t')\) is a transition of the operation when the tuple consisting of the behaviour of \(a\) in \(t\) and the behaviour of \(a\) in \(t'\) is in \(h\text{-trans}(A, e)\).

(a) Let \(h\) denote the history of \(a\) in \(s\) and \(h'\) the history of \(a\) in \(s'\). Since \((h, h')\) is in \(h\text{-trans}(A, e)\), \(\text{behav}(A)(h), \text{behav}(A)(h')\) is in \(h\text{-trans}(A, e)\) by Lemma 4.

Therefore, \((\text{sel-map}(A)(s), \text{sel-map}(A)(s'))\) satisfies the operation using Method 2.

(b) Let \(h\) denote the behaviour of \(a\) in \(t\) and \(h'\) the behaviour of \(a\) in \(t'\). Since \((h, h')\) is in \(h\text{-trans}(A, e)\), for all histories \(h'\) such that \(\text{behav}(A)(h') = b'\), there exists a history \(h\) such that \(\text{behav}(A)(h) = b\) and \((h, h')\) is in \(h\text{-trans}(A, e)\) by Lemma 5. Therefore, for all states \(s'\) in \(\text{sel-map}(A)^*\{\{t\}\}\), there exists a state \(s\) in \(\text{sel-map}(A)^*\{\{t\}\}\) such that \((s, s')\) satisfies the operation using Method 1.

All other operations in \(C[A]\) can be constructed from the operations above and, using the above results, can be shown to satisfy the lemma.  

**Lemma 7.** Let \(H_1\) denote the set of histories of \(C[A]\) derived using Method 1.

Given any finite history \(h_1\) which is in \(H_1\), for any state \(s\) such that \(\text{sel-map}(A)(s) = \text{sel-map}(A)(\#h_1, \text{states}(\#h_1, \text{states}))\), there exists a history \(h_1'\) also in \(H_1\) such that \(\text{sel-map}(A)(h_1') = \text{sel-map}(A)(h_1)\) and the final state of \(h_1'\) is \(s\).

\[
\forall h_1 : H_1 \bullet
\forall s : \text{sel-map}(A)^*\{\#s\} \Rightarrow
\exists h_1' : H_1 \bullet
\text{sel-map}(A)(h_1') = \text{sel-map}(A)(h_1) \land
h_1'. \text{states} = \#h_1'. \text{states} = s
\]

**Proof.** The proof is by induction over the length of \(h_1.\text{events}\).

(i) If \#\(h_1.\text{events}\) = 0 then \(h_1.\text{states}(\#h_1.\text{states}) = h_1.\text{states}(1)\) must satisfy the predicate of the initial state schema of \(C[A]\) using Method 1. Therefore, all states \(s\) such that \(\text{sel-map}(A)(s) = \text{sel-map}(A)(\#h_1, \text{states}(1))\) must also satisfy the predicate of the initial state schema of \(C[A]\) by Lemma 3. Hence, there must exist an \(h_1'\) in \(H_1\) such that \(\text{sel-map}(A)(h_1') = \text{sel-map}(A)(h_1)\) and the final state of \(h_1'\) is \(s\).

(ii) Assume the lemma is true for all \(h_1\) in \(H_1\) such that \#\(h_1.\text{events}\) = \(n\) for some \(n \geq 0\).

If \#\(h_1.\text{events}\) = \(n + 1\) then \((h_1.\text{states}(n + 1), h_1.\text{states}(n + 2))\) must be a state transition of the event \(h_1.\text{events}(n + 1)\). Hence, for all states \(s\) such that \(\text{sel-map}(A)(s') = \text{sel-map}(A)((h_1.\text{states}(n + 2)), \text{by Lemma 6, there exists a state } s\) such that \(\text{sel-map}(A)(s) = \text{sel-map}(A)((h_1.\text{states}(n + 1))\) and \((s, s')\) is a transition of \(h_1.\text{events}(n + 1)\).

Since all pre-histories of \(h_1\) are in \(H_1\), there exists a history \(p_{h_1}\) in \(H_1\) such that \(p_{h_1} \in \text{prehist}(h_1)\) and \#\(p_{h_1}.\text{events}\) = \(n\). Therefore, there exists a history \(p_{h_1}'\) in \(H_1\) such that \(\text{sel-map}(A)(p_{h_1}') = \text{sel-map}(A)(p_{h_1})\) and \(p_{h_1}'. \text{states}(n + 1) = s\) by the assumption. Therefore, there must exist an \(h_1'\) (which extends \(p_{h_1}'\)) in \(H_1\) such that \(\text{sel-map}(A)(h_1') = \text{sel-map}(A)(h_1)\) and the final state of \(h_1'\) is \(s\). Hence, the lemma is true for all \(h_1\) in \(H_1\) such that \#\(h_1.\text{events}\) = \(n + 1\).  

\[\text{Received November 1993} \]

\[\text{Accepted in revised form October 1994 by C. B. Jones}\]