A JUNCTION BETWEEN STATE BASED AND BEHAVIOURAL SPECIFICATION

H. Bowman
J. Derrick*

Abstract: Two of the dominant paradigms for formally describing and analysing OO distributed systems are *state based specification*, e.g. Object-Z, and *behavioural specification*, e.g. process algebra. The style of specification embodied by the two paradigms is highly contrasting. With state based techniques the data state is explicitly defined while the temporal ordering of operations is left implicit, in contrast in behavioural techniques, no explicit data state definition is given while the temporal ordering of action offers is focused on. However, in order to support sophisticated software engineering principles, e.g. multi-paradigm specification, viewpoints modelling and subtyping, there is now considerable interest in developing strategies for relating state based and behavioural specification paradigms.

This paper serves two purposes - firstly, it reviews the existing body of work on relating these two specification paradigms and secondly, it presents some new results on the topic. In particular, we present a testing characterisation of downward simulation, which is the standard state based refinement relation and is also closely related to subtyping. Central to this characterisation is giving a behavioural interpretation to the meaning of state based operations outside their pre-conditions.

*Computing Laboratory, University of Kent at Canterbury, Canterbury, Kent, CT2 7NF, UK;
Email: {H.Bowman,J.Derrick}@ukc.ac.uk
(The first author is currently on leave at VERIMAG, Centre Equation, 2 rue Vignate, 38610 GIERES, France and CNR-Istituto CNUCE, Via S. Maria 36, 56126 - Pisa - Italy with the support of the European Commission's TMR programme.)

P. Ciancarini et al. (eds.), *Formal Methods for Open Object-Based Distributed Systems*

INTRODUCTION

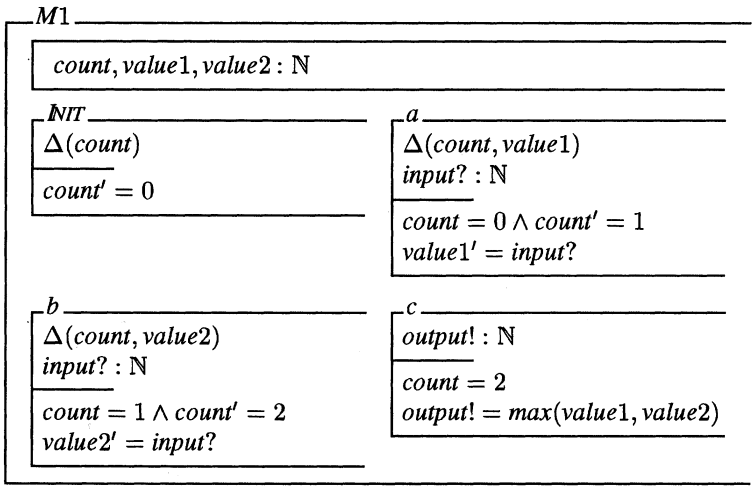
Two of the dominant paradigms for formally describing and analysing computer systems and in particular, OO distributed systems, are *state based specification* and *behavioural specification.* Both have many realisations, e.g.

- *state based techniques* - Z [51], Object-Z [20], ZEST [15], B [1], VDM [34], VDM++ [37] and Liskov and Wing notation [40].
- *behavioural specification* - process algebra, such as, LOTOS [6], CCS [42], CSP [31] and Pi-calculus [43]; and extended state machine techniques, such as, Estelle [32] and SDL [13].

The style of specification embodied by the two is without doubt highly contrasting:-

- With *State Based Techniques* the data state is described explicitly, e.g. via Z state schemas. Then operations are defined over this state space, with the pre-condition of the operation defining the data states at which an operation can be applied and the post-condition defining the data results of applying an operation. Thus, the temporal order of operations is not explicitly defined.
- In contrast, with *Behavioural Techniques* no explicit definition of the data state is given, rather the specification comprises an expression of the temporal order in which events can occur. (We use the term event to embrace a spectrum of primitive concepts, in particular, *actions* from process algebra and *transitions* from state machines.)

We can illustrate the difference between the two techniques by considering the following simple example of an Object-Z class:



The variables *count*, *value*1 and *value*2 declared in the (unnamed) state schema are local to the class. The initial state schema *INIT* defines the initial values of the variables

in the state schema. The class specified above has three operations: a, b and c. Each operation has a Δ-list which contains those state variables which may change when the operation is applied to an object of that class. An operation does not change the state variables that are not listed in its Δ-list. Hence the operation a implicitly contains a predicate $value2 = value2'$. The operations in this class allow two integers to be inserted (using a and b), and the operation c will output the maximum of those values.

Compare this Object-Z specification with the following LOTOS specification:

process $M1[a, b, c]$: **noexit** :=
$a?input1 : nat;\ b?input2 : nat;\ c!max(input1, input2);\ stop$
endproc

which performs the actions a, b and c in sequence; inputting values on the first two and then outputting the maximum of the two values on the third. The process then evolves to deadlock. Although stylistically very different, after some consideration, it is clear that these two descriptions do exhibit "the same" behaviour.

However, developing general purpose strategies for relating state based and behavioural specifications a difficult, but important topic. In particular, the following reasons motivate our interest in this issue:-

- *Subtyping.* Subtyping plays a pivotal role in obtaining incremental system development, with its relationship to different inheritance mechanisms being crucial. It is defined as follows - a type A is a subtype of a type B iff objects of type A may be used in any situation where an object of type B was expected, without the environment being able to tell the difference. Now both state based and behavioural specification interpretations of subtyping[1] have been given. Specifically, Liskov and Wing have developed a state based description notation, over which they define subtyping (we refer to this approach as *state based subtyping*), while testing preorders in the behavioural domain ensure substitutability in an analagous way to subtyping. This leaves the question of how these two realisations relate to one another. In fact, at first sight they seem incompatible. This is because state based subtyping (in line with OO interpretations of subtyping) allows *functionality extension*, while the most well behaved testing preorders, reduction [11], failures, trace preorder, failure traces [56] etc, do not. By way of clarification, the functionality of a specification is extended if operations are allowed to be applied in more states. Mechanisms embraced by this term include interface enlargement and trace extension [9]. In order to investigate this seeming anomoly, a junction between the state based and behavioural worlds needs to be found.

- *Multi-paradigm Specification.* There is considerable current interest in combining multiple description paradigms into a single specification approach. In some respects this offers an alternative to the strategy of devising single all embracing

[1] Here we particularly mean "behavioural" subtyping as opposed to signature based interpretations of the concept. However, we speak only of subtyping in the main body of the text to avoid the terminology clash of the two uses of behavioural.

formal specification languages, such as Raise [27] or E-LOTOS. Its advantage is that languages appropriate to different styles of specification can be used in conjunction. In this respect, the integration of state based and behavioural techniques is an obvious one, since the expressive powers of the two paradigms are highly complementary: data aspects can be described in state based notations, while concurrent interaction can be specified in behavioural notations. This observation has prompted a lot of recent research, e.g. integration of CSP and Object-Z [49, 23, 25, 50] and CSP and Z [24].

- *Partial Specification Approaches.* We can also point to the considerable interest in partial specification. These are development strategies that allow the complete system specification to comprise a number of partial specifications, each giving a different perspective on the target system. Partial specification arises in a number of different development strategies, e.g.

  (ODP) Viewpoints [39], (Finkelstein) Viewpoints [22], Aspects [36], UML diagrams [7], Views [33], constraints [5].

  Although, the possibility that all the partial specifications are described in the same formalism is not precluded, different partial specifications will typically be described in different notations. This allows the expressive capabilities of different formalisms to be matched to the specification requirements of different perspectives. Thus, we once again obtain a multi-paradigm specification formalism[2] and two of the formalisms used could be state based and behavioural notations, e.g. in ODP viewpoints Z or Object-Z could be use for the information viewpoint and LOTOS for the computational viewpoint.

There is now a very large body of work on relating the two paradigms, e.g. [35, 2, 3, 54, 49, 50, 23, 25, 24]. In respect of this work this paper serves two purposes. Firstly, it summarises this existing body of work and secondly, it presents some new results. Our new results are a testing characterisation of downward simulation, which is the standard state based refinement and also corresponds to state based subtyping. Central to this characterisation is giving an interpretation to the behaviour of state based operations outside their pre-condition.

In order to maintain the level of generality of the paper we work with relational models of the state based world and labelled transition system interpretations of the behavioural world. However, our exposition is fully general and for example, direct extrapolations to the Liskov and Wing approach could easily be made.

Basic Notation. We quickly review some basic Labelled Transition System (LTS) notation which we will use in the remainder of the paper.

Labelled Transition Systems. Let LTS denote the set of all labelled transition systems over a set of events A. Then an arbitrary, $l_p \in \text{LTS}$ has the form,

$$< S_p, p, \longrightarrow_p >$$

[2]However, a characteristic aspect of partial specification is that the different specifications are likely to have "overlapping" intent - each imposing a different constraint on the same aspect of the target system. Such overlapping constraints do not necessarily arise in the multi-paradigm setting.

where,

- S_p is the set of states and $S_p \subseteq \Sigma$ - the set of all possible states;
- $p \in S_p$ is the start state;
- $\longrightarrow_p \subseteq S_p \times A \times S_p$, is the transition relation and $(p_1, a, p_2) \in \longrightarrow_p$ is denoted $p_1 \xrightarrow{a} p_2$.

Traces. Let σ range over $A^* \cup \{\epsilon\}$, the set of event traces, where ϵ is the empty trace. Then we define the generalised event relation ($\Longrightarrow$) as:

$$\forall p, p' \,.\, (p \stackrel{\epsilon}{\Longrightarrow} p \ \wedge \ (p \stackrel{a\sigma}{\Longrightarrow} p' \text{ iff } \exists q \,.\, p \xrightarrow{a} q \ \wedge \ q \stackrel{\sigma}{\Longrightarrow} p'))$$

Notice that this formulation is simpler than normal since we do not have internal events.

Stop. In the sequel, we will need to refer to deadlocked behaviour, thus we introduce the labelled transition system, *stop*, where, $stop = < \{d\}, d, \varnothing >$ for some $d \in \Sigma$.

Out. We also define the set of events that are initially offered by a process, the *out* set:

$$\forall p \in \Sigma \,.\, out(p) = \{ a \in A \mid p \xrightarrow{a} \}$$

Paper Plan. We begin (in the next section) by considering how state based notations can be given behavioural semantics, e.g. as labelled transition systems or failures. Then refinement and subtyping preorders in the two paradigms are related. Then we give a bisimulation characterisation of downward simulation, which is one of the most important state based refinement relations and is also very closely related to subtyping. This is followed by our testing characterisation of downward simulation and finally we presented some concluding remarks.

INTERPRETING STATE BASED TECHNIQUES WITH BEHAVIOURAL SEMANTICS

As discussed in the introduction there have been a number of proposals which seek to combine, or at least use together, state-based techniques with behavioural specification. In order to do so it is necessary to reconcile the differing approaches to the semantics of these techniques, and in this section we discuss the possible ways in which this might be done.

Processes and Objects. In order to be able to do this it is necessary to find a semantic model which can represent constructs defined in both state-based and behavioural techniques. One problem with this, however, is the lack of a construct in most state-based specification languages identifiable with processes in a behavioural technique such as a process algebra. This is not true of object-oriented state-based specification languages such as Object-Z. Central to OO is the view of a system as a collection of distinct, interacting objects whose state and operations are encapsulated in *classes*. Hence, there is a strong relationship between classes in OO systems and processes in

concurrent systems: interactions between instances of each define the system behaviour [21, 48, 55, 58].

This identification shifts the focus away from relations to transitions when we are considering questions of semantics, and lies at the heart of the reconciliation between state-based techniques and behavioural specification. To illustrate what we mean by this let us first consider the standard approach to defining the semantics of a state-based specification language.

State Based Semantics. The underlying model of a state based system is a relational model, where the components of an abstract data type (ADT) are relations. An ADT is a quadruple $\mathcal{A} = (Astate, ai, \{aop_i\}_{i \in I}, af)$ which acts on a global state space G such that,

- $Astate$ is the space of values;
- $ai \in G \leftrightarrow Astate$ is an initialisation;
- $af \in Astate \leftrightarrow G$ is a finalisation;
- aop_i are operations in $Astate \leftrightarrow Astate$.

At this stage we consider all relations to be total. A program P is a sequence of operations upon a data type beginning with an initialisation and ending with a finalisation, e.g.

$$P(\mathcal{A}) = ai \fatsemi aop_1 \fatsemi aop_2 \fatsemi af$$

Thus we consider a program to be a relation over some global state G, with the operations acting on some local state space $Astate$. In most formalisms (e.g. Z, Object-Z, VDM, B etc) the finalizations do not appear explicitly as they are just projections onto the space G from $Astate$, and we needn't consider them further here.

In this relational framework the relations were assumed to be total relations. However, in a specification not all operations are total, and the meaning of an operation ρ specified as a partial relation is:

- ρ behaves as specified when used within its precondition (domain);
- outside its precondition, anything may happen.

In order to deal with this in the semantic framework we totalise the partial relations (informally, this is called *adding clouds*), i.e. we add a distinguished element $\perp$, denoting undefinedness, to the state space $Astate$ and we denote such an augmented version of X by $X^{\perp}$. Thus if ρ is a partial relation between X and Y, we add the following sets of pairs to ρ

$$\{x : X^{\perp}, y : Y^{\perp} \mid x \notin \operatorname{dom} \rho \bullet x \mapsto y\}$$

and call this new (total) relation $\dot{\rho}$. Figure 1 shows a partial relation (depicted with solid arrows) and its underlying totalisation. A program is then a sequence of totalised relations, e.g.

$$P(\dot{\mathcal{A}}) = \dot{ai} \fatsemi \dot{aop}_1 \fatsemi \dot{aop}_2 \fatsemi \dot{af}$$

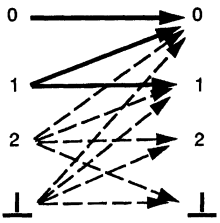


Figure 1 A partial relation and its totalisation (or adding a cloud)

This interpretation works well for the specification of sequential systems, however, in order to properly describe distributed and communicating systems it is also necessary to be able to express that an operation *cannot* occur. To describe this other totalisations need to be used, the most common being the interpretation that nothing can happen outside the precondition [53, 52]. This is known as the *blocking model* or firing condition interpretation, and the specification language Object-Z adopts this interpretation. Some languages, such as B, have constructs which enable both interpretations to be specified.

The relevance of this is two fold. Firstly, the choice of relational interpretation (blocking or non-blocking) will effect how a state-based technique is combined with a behavioural language. Secondly, the choice of totalisation effects the refinement rules and consequently our notion of refinement and subtyping.

The view of a state based system as a relation between input and output is also a very sequential notion, and it is not easy to see its relationship with the idea of an evolving process. However, as discussed above there is a strong relationship between OO and concurrent systems, and we can exploit this when giving a behavioural semantics to an OO state based technique by tying the notion of a class to that of a process.

Operations and Events. In order to relate classes and processes, we need a relationship between operations and events. Since both represent observable, atomic actions, the most obvious approach is to identify them. This can be called the single event approach.

However, while identifying operations and events is the obvious choice, there are in fact other options. For example, Benjamin [2] suggests identifying operation parameters and events in order to integrate Z and CSP.

This is an example of the multi-event approach when defining the granularity of operations, where there is not a one-to-one correspondence between operations and events. This is discussed in some detail in [24] where it is noted that the double (and multi) event approaches take a more implementation oriented view than the single event approach. The double-event approach, which introduces separate events for input and output, is used in [46], and the multi-event approach in [41, 54].

Double and multi-event approaches are complicated, and in particular have no theory of refinement. For that reason we shall confine ourselves to the single event approach for the remainder of this paper. However, it should be noted that the single event approach is clearly an abstraction of the reality of OO systems, where a single

method/operation invocation comprises a number of interactions/actions. Thus, we view the single event approach as a useful simplifying abstraction which is applicable at high levels of abstraction and early stages of system development.

Having discussed state based semantics we are now in a position to review how we might adopt a particular behavioural interpretation, and we begin with labelled transition systems (LTS).

LTS semantics

The simplest behavioural semantics we can give to a state based technique is a labelled transition system. Consider for the moment a state based technique with a blocking model of operations, examples of which include Object-Z and ZEST. We can view such a specification as defining an LTS where the nodes in the LTS are the possible states of the system, and the transitions correspond to operations. For example, in Object-Z we can define an LTS (known as the *structural model*) which consists of (see [48]):

states the set of states in the state transition system. This will be the possible states of an object of an Object-Z class.

initial the set of possible initial states of the system. This will be the set of states which satisfy the predicate of the initial state schema of an Object-Z class.

trans a function from the set of possible operations to the associated set of state transitions. For each operation in an Object-Z class, this includes those pairs of states from *states* which satisfy the precondition and resulting postcondition of the operation associated with the operation.

For example, in the structural semantics for the Object-Z class $M1$ presented in the introduction, the state space consists of sets assigning values to the three state variables $count, value1, value2$. The initial states are those components of *states* which have $count \mapsto 0$, and *trans* contains transitions representing the operations a, b and c, which we depict in figure 2 (there is a transition for each input and output, here we just depict one per operation).

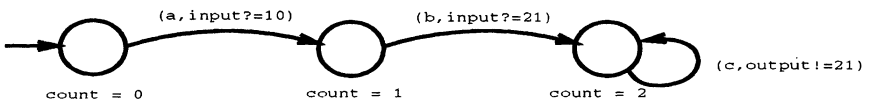


Figure 2 The LTS semantics of the class $M1$

The blocking model of Object-Z is clearly visible here. If we were to adopt a non-blocking model, then additional transitions would need to be added to represent the fact that any possible behaviour was feasible outside the precondition. For example, we would have to add transitions for the b operation as we partly show in figure 3

(the black dot represents the undefined bottom element), and similarly for the other operations a and c. Similar derivations of labelled transition systems from other OO

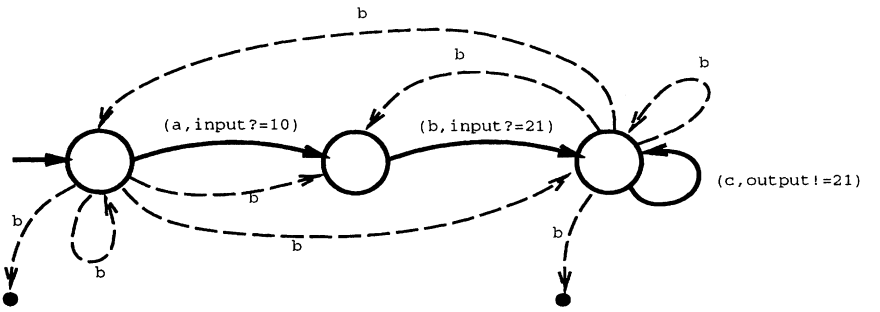


Figure 3 Adding transitions in a non-blocking model

state based techniques also exist [16, 47]. As in [48], the basic idea used in [16] and [47] is that labels in the transition system are operation schema names together with any input/output values. A transition is added whenever an operation is applicable at a node, which represents a particular binding of state variables.

In addition to pure state based techniques, some combinations of state based languages with behavioural notations also use labelled transition systems to define the semantics of the hybrid language, and this of course involves using an LTS for the semantics of the state based component. Examples of this approach include [26, 54], which both offer combinations of Z and CCS. For example, in [54] Z specifications are embedded into a labelled transition system by first interpreting all operation schemas in the specification as transitions between old and new states, and then the set of all assignments is used as the states in the LTS and the set of all names of operation schemas is used as the set of labels. There are some minor differences here compared to the work of [48], however the essential idea behind all these schemes is the same.

Failures Semantics

An alternative behavioural characterisation is to use a failures based semantics as opposed to an LTS. For example, the standard semantics of CSP is the failures-divergences semantics developed in [12]. A process l_p is modelled by the triple (A, F, D) where A is its alphabet, F is its *failures* and D is its *divergences*. The failures of l_p are pairs (σ, X) where σ is a trace of l_p and X is a set of events the process *may* refuse to perform after undergoing σ. That is, after performing σ the process can reach a state where a deadlock will result if the environment only offers actions in X. More formally, (σ, X) is a failure of l_p iff,

$$\exists p' . p \stackrel{\sigma}{\Longrightarrow} p' \ \wedge \ X = A - out(p')$$

The divergences of a process are the traces after which the process *may* undergo an infinite sequence of internal events, i.e. livelock. Divergences also result from unguarded recursion. The semantics is well-formed if the failures and divergences satisfy a number of axioms [12].

Such a semantics has been used for the Object-Z language when it has been combined with CSP [49, 23, 25, 50]. For example, in [49] classes are given a failures-divergences semantics, and this allows classes defined in the Object-Z part of the specification to be used directly in the CSP part and hence for these two languages to be combined.

To define the semantics Smith models a class C by a process. The alphabet is taken to be the set of operations of the class, and the traces are sequences of events corresponding to sequences of operations. The failures are derived from the histories[3] of a class as follows: (σ, X) is a failure of the process if

- there exists a finite history of C whose first state satisfies the initialization,
- the sequence of operations of the history corresponds to the sequence of events in σ, and
- for each event in X, there does not exist a history which extends the original history by an operation corresponding to the event.

Divergence is not possible since Object-Z does not allow hiding of operations nor recursive definitions of operations, therefore the divergences of a class are empty. With these derivations Smith shows that the semantics are well-formed.

A similar approach is taken in [23] which defines a combination of Object-Z and CSP called CSP-OZ. The principle difference between the work in [23] and that of [49] is the handling of outputs from operations. In [49] when operations are mapped to events the distinction between input and output is removed, i.e. the basename of a parameter name is used. For example, given the operation *Join*:

Join

$\Delta(items)$
$item? : T$

$\#items < max$
$items' = \langle item? \rangle \frown items$

the event corresponding to joining a value x is $Join.\{(item, x)\}$, where the decoration of ? in *item?* has been removed.

On the other hand in [23] a distinction is made in the refusals between input and output parameters. The reason for making this distinction is that Fischer allows non-determinism in the outputs to be reduced under refinement in his CSP semantics, which is consistent with the standard approach to refining outputs in state-based systems. Whereas the semantics in [49] allows no further restriction on the outputs to be made.

[3] The history model extends the structural model to enable object evolution to be modelled.

Another interesting aspect of [23] is that Fischer incorporates both the blocking and non-blocking model. He does this by using an extra schema to represent the guard of an operation. He uses the keyword `enable` for the guard and `effect` for the schema describing the state change of the operation. The operation is blocked outside of `enable`, but it is divergent if it is not blocked but applied outside the precondition of `effect`.

Other semantic models

The above are not the only semantic models possible for both state-based and behavioural languages.

For example, the work we have described so far has considered a relational framework for state-based techniques and languages. Although this is the most common characterisation, it is not the only one. An alternative model is to use weakest precondition formulae over action systems instead of relations. Such an approach forms the semantic model for the refinement calculus [44]. An action system consists of a state, an initialisation, and a number of labelled actions on the state. Each action is a guarded command. In [57], Woodcock and Morgan show how to calculate the failures divergence semantics for action systems, thereby giving an alternative link between state based systems and behavioural semantics. This work represents an independent strand in its own right and we will not consider it in depth in this paper.

Also worth mentioning here is the range of behavioural semantics considered by Smith in [48], which builds a series of semantic models leading to a fully abstract semantics. A semantics is fully abstract when any two components with identical semantic denotations are behaviourally equivalent. This definition means that to prove full abstraction another semantics is needed from which a notion of behavioural equivalence can be derived. Smith uses a trace model as the basis for behavioural equivalence. This work is interesting because a fully abstract semantics captures meaning independent of any syntactic representation, and thus there is the potential to define subtyping in terms of behavioural compatibility. This is discussed briefly in [48].

Smith shows that neither the history model nor the readiness model are fully abstract. The readiness model (cf [45]) includes the set of event that an object is ready to perform in addition to the trace.

His fully abstract model extends the readiness model by not only including the ready set after a trace, but also includes a sequence of ready sets at each stage of its evolution. This model is known as the complete-readiness model. Then two classes are behaviourally equivalent whenever they have the same complete-readiness model. This complete-readiness model is stronger than the failures model, however, it is weaker than bisimulation.

RELATING REFINEMENT AND SUBTYPING RELATIONS

Interpreting state based methods in behavioural semantics is an obvious first step when relating state based and behavioural techniques. However, semantic interpretations of specifications should always be viewed modulo relevant preorder and equivalence

relations. Thus, to really identify a junction between state based and behavioural techniques, we need to determine how particular preorders and equivalences in the two paradigms relate.

We consider this issue here; first we discuss state based and then behavioural preorders, and then we show how they relate. We concentrate on preorders, since these are most relevant to the topics (as highlighted in the introduction) that have motivated our investigations of the state based/ behavioural junction. In addition, most equivalences can be obtained by intersecting a particular preorder with its inverse.

State Based Preorders

Simulations. In this subsection we consider notions of refinement and subtyping that arise in the relational framework. We first consider refinement of state-based specifications. To consider refinement we assume that the abstract and concrete data types have the same global state space G and that the indexing sets for the operations coincide (i.e. every abstract operation has a concrete counterpart and vice versa). Refinement is defined as being reduction of non-determinism:

Definition 1 *A data type C refines a data type A if, for every program P, $P(C) \subseteq P(A)$.*

This definition of refinement involves quantification over all programs, and in order to verify such refinements, `simulations`[4] are used which consider values produced at each step of a program's execution. `Simulations` are thus the means to make the verification of a refinement feasible. In order to consider values produced at each step we need a relation r between the two state spaces *Astate* and *Cstate*. Such a *retrieve* relation gives rise to two types of step by step comparisons: `downwards simulation` and `upwards simulation`.

The definition of `downward` and `upward simulations` can be given for partial relations as follows[5]:

Definition 2 *A* `downward simulation` *is a relation r from Astate to Cstate s.t.*

$$ci \subseteq ai \fatsemi r$$
$$(dom\, aop \lhd r \fatsemi cop) \subseteq aop \fatsemi r$$
$$ran((dom\, aop) \lhd r) \subseteq dom\, cop$$

An `upward simulation` *is a relation l from Cstate to Astate s.t.*

$$ci \fatsemi l \subseteq ai$$
$$\overline{dom(l \rhd (dom\, aop))} \lhd cop \fatsemi l \subseteq l \fatsemi aop$$
$$\overline{dom\, cop} \subseteq dom(l \rhd (dom\, aop))$$

[4] We have terminology problem since relations called simulations arise in both the state based and behavioural worlds. In order to avoid confusion, when we refer to a state based simulation we write it in type writer font and when we refer to a behavioural simulation we will write it in sans serif.

[5] We ignore finalizations; $\lhd$ is domain restriction, $\rhd$ is range subtraction, $\lhd$ is domain subtraction, and $\overline{X}$ is the complement of X, see [51].

These `simulations` are derived with a non-blocking model of partial operations, and hence allow the weakening of preconditions under refinement. With a non-blocking model this is just reduction of non-determinism, by widening a precondition we are constraining the operation to have a particular behaviour in place of the (non-blocking) anything may happen.

In a specification language such as Z the rules for `downward simulations` become (there are similar `upward simulation` rules):

1. $\forall Cstate' \bullet Cinit \Rightarrow \exists Astate' \bullet Ainit \wedge R$
2. $\forall Astate;\ Cstate \bullet \mathrm{pre}\, AOp \wedge R \Longrightarrow \mathrm{pre}\, COp$
3. $\forall Astate;\ Cstate;\ Cstate' \bullet \mathrm{pre}\, AOp \wedge R \wedge COp \Longrightarrow \exists Astate' \bullet R' \wedge AOp$

where R is the retrieve relation and $Ainit$ is the abstract initialisation, and every abstract operation AOp has a concrete counterpart COp. We write $C \leq A$ to denote that the two state based specifications A and C are related according to these rules[6].

`Simulation` rules for a blocking model can easily be derived in a similar fashion. Now we can't widen the precondition because the precondition now defines the total limit of behaviour. Thus the `simulation` rules for the blocking model are

1. $\forall Cstate' \bullet Cinit \Rightarrow \exists Astate' \bullet Ainit \wedge R$
2. $\forall Astate;\ Cstate \bullet \mathrm{pre}\, AOp \wedge R \Longleftrightarrow \mathrm{pre}\, COp$
3. $\forall Astate;\ Cstate;\ Cstate' \bullet R \wedge COp \Longrightarrow \exists Astate' \bullet R' \wedge AOp$

These characterisations of refinement as `simulations` consider all operations to be observable. In [18] `simulation` rules are developed for non-blocking models which contain internal operations. These are known as weak `simulation` rules. Blocking model versions of `weak simulations` (which also treat divergence) are given in [17], and `weak simulation` rules for the combined Object-Z and CSP languages of Smith are considered in [25, 50].

Subtyping. In addition to refinement, the idea of subtyping has become an important concept in OO languages. Like refinement, subtyping is based upon the idea of compatibility or substitutability and there have been a variety of definitions, e.g. [40, 14] etc. As a basis for a comparison we will look at the definition due to Liskov and Wing [40]. Definitions of subtyping consider a mutable (i.e. changeable) object under the control of one or more users.

In the context of a single user of a mutable object, the definition of subtyping is close to that of a `downward simulation`. An abstraction function is used to relate the state spaces of the sub and supertype in an identical way to a retrieve relation. With this abstraction function there are two sets of rules that must hold: signature rules and method rules.

[6]Notice the order of the relation - C is a refinement of A. In much of the literature, refinement is written in the opposite direction. We adopt this direction to maintain consistency of presentation throughout the paper.

Method rules are just the non-blocking `downward simulation` rules for the objects methods. That is, modulo the abstraction function, the preconditions of a subtype can be widened, and the postcondition can reduce any non-determinism.

The signature rules extend the notion of subtyping to the signature of a method, allowing contravariance of arguments and covariance of results.

This is a departure from the standard notions of refinement by `simulations` which allow no change to the types of the inputs or outputs of operations under refinement. There are, however, some recent generalisations of refinement in which refinements of the inputs and outputs can be undertaken [4, 52].

In the context of shared users the definition of subtyping needs further constraints, and [40] propose the use of history properties. These properties aim to rule out subtypes where surprising behaviour results from allowing multiple users to change one object. These history properties have no obvious relational refinement counterpart.

Behavioural Preorders

There is a very large spectrum of behavioural preorders, which can be divided into two basic classes: *sequence based testing* and (bi)simulation relations. Both types of relations play a role in the sequel.

Sequence Based Testing. A rich and extensively investigated aspect of concurrency theory is testing [56]. This considers how the behaviour of processes can be observed by their environment. Such observation of systems naturally yields a preorder between processes. Two processes l_p and l_q might be related by such a preorder if when given l_p the environment is not able to distinguish its behaviour from that of l_q.

Probably the most important testing preorder is failures [12]; it realises the following testing property:

$$l_{p_1} \sqsubseteq_F l_{p_2} \quad \text{iff} \quad \forall G \subseteq A, \forall l_t \in \text{LTS} \; . \; p_1 \mid [G] \mid t \stackrel{\sigma}{\Longrightarrow} \sim stop \text{ implies } p_2 \mid [G] \mid t \stackrel{\sigma}{\Longrightarrow} \sim stop$$

where $l_{p_1} \sqsubseteq_F l_{p_2}$ denotes that l_{p_1} is a failures refinement of l_{p_2}; $\mid [G] \mid$ is a parallel composition operator which requires its two components to synchronise on actions in G [6]; $\sim$ is bisimulation equivalence [42]; and relation composition is denoted by juxtaposition[7]. Informally, the condition states that l_{p_1} refines l_{p_2} if and only if, for all possible testers, if p_1 can perform a trace σ and then deadlock, then under the control of the same tester p_2 can perform σ and then deadlock. Thus, even more informally, l_{p_1} does not add any new deadlocks to those that can arise from l_{p_2}. This is a "no suprises" property analagous to that embodied in subtyping.

Testing relations can be characterised by comparing the set of *observations* that can be made of a process. The observations that characterise $\sqsubseteq_F$ are failures [12][8], as introduced in section 3. In particular,

---

[7] i.e. $p \mid [G] \mid t \stackrel{\sigma}{\Longrightarrow} \sim stop$ means $\exists q \; . \; p \mid [G] \mid t \stackrel{\sigma}{\Longrightarrow} q \wedge q \sim stop$

[8] In fact, since we do not have internal actions the differences between failures, reduction [11] and must testing [56] disappear.

$l_{p_1} \sqsubseteq_F l_{p_2}$ if and only if every failure of l_{p_1} is also a failure of l_{p_2}

The importance of $\sqsubseteq_F$ arises because $l_{p_1} \sqsubseteq_F l_{p_2}$ can also be viewed as expressing that l_{p_1} cannot be distinguished from l_{p_2} by any tester where the tester is exactly a third process, e.g. l_t. Thus, it characterises the testing power of the behavioural specification notation itself!

However, we can consider testing in a more general context by viewing processes as closed systems with some form of interface to the outside world. Then the process is observed through this interface. By varying the nature of the interface, one can conceptually vary the "closedness" of the box. Thus, some classes of interface offer a very limited capacity to interact with the process, while others allow highly invasive interaction with the process.

In this way, different notions of testing can be obtained, each supporting a different level of invasiveness and importantly, each can be characterised by a different style of observation, yielding a different preorder. For example, the obervations in *trace preorder* are traces, the observations in *failure traces* are traces with failures information throughout, and the observations in *readiness preorder* consider the actions that may be accepted rather than those that may be refused. Further, this spectrum of testing preorders can be placed in a hierachy of strength. For example, [56] considers the related hierachy of equivalences that results from these different notions of testing.

We use the term *sequence based testing* to embrace all forms of testing which yield linear sequences of observations (perhaps entwined with some refusal or ready information). All the preorders that we have so far considered sit within this category.

(Bi)Simulation Based Relations. An alternative way to relate processes is to match transitions between the processes in the familiar inductive style of (bi)simulation relations. This yields a further spectrum of preorders, e.g. simulation, 2/3-bisimulation [38], weak and strong bisimulation [56] and the strength of these relations can be compared to the simple testing relations, yielding an enlarged hierachy of preorders.

Testing categorizations of these simulation relations can also be given. However, since these relations are more discriminating with regard to the branching structure of LTSs, sequence based testing is not sufficient, rather observations have to be constructed as trees. This can be viewed as giving the environment the capability that at any time during the run of a process an arbitrary (but finite) number of copies of the process in its current state can be taken and all observed independently. This copying yields the branches in the observation tree. We call such testing *tree based testing*.

Relating State Based and Behavioural Preorders

In this subsection we consider the relationship between the various state-based and behavioural preorders. There are a number of results in this area and we finish this section with a list of some of the remaining open issues.

Basic Results. We first consider a non-blocking model of operations and the relations that they represent. In this model the `simulation` rules allow both the strengthening of an operations postcondition and the weakening of its precondition. Because of the non-blocking viewpoint both these notions are in fact reduction of non-determinism. Therefore it can be shown that refinement in a state-based technique corresponds to

failures-divergences refinement [28, 29] (in addition to not adding failures, failures-divergences requires that the divergences are also not increased). The relational framework we have described above is used, however, similar results can be obtained in other frameworks. For example, in [57], Woodcock and Morgan use weakest precondition formulae over action systems and show that in this setting failures-divergences refinement corresponds to state-based refinement by showing that the `simulation` methods are sound and jointly complete.

Similar results are obtained in a blocking model of operations, this time the results are due to Josephs in [35]. He casts his results in a transitional setting and shows that `upward` and `downward simulation` are sound and jointly complete for failures-divergences refinement. With a blocking model he can dispense with divergence (operations don't diverge outside their preconditions), however he now has to explicitly deal with refusals. The issue here is that in a non-blocking model there are no refusals since operations are total (by totalisation) so all traces are possible and refinement simply reduces non-determinism and divergence. With a blocking model outside an operations precondition we get a refusal but no divergence. However, although the calculations of failures and divergences are different in blocking and non-blocking models, refinement in each model corresponds to failures-divergences and the `simulation` methods are always sound and jointly complete. These results are summarised in table 1. The final row is the issue we consider in the sequel.

Table 1 Relating Preorders

	state based refinement	*(equivalent) behavioural refinement*
blocking model	downward and upward simulation	failures-divergences
non-blocking model	downward and upward simulation	failures-divergences
non-blocking model	downward simulation	????

Weak Simulation Relations. An extension to the work of Joseph has been made in [17] which shows that weak `simulation` rules again correspond to failures-divergences refinement. Because we are now dealing with internal operations [17] provides a treatment of divergence as well.

Subtyping. We have confined discussion so far to refinement relations. Let us now consider notions of subtyping. One immediate question to answer is what is the relationship between subtyping and refinement. Without the use of the history properties we saw above that subtyping was synonymous with `downward simulations` (modulo the signature rules). Therefore the natural question to ask is what is the corresponding behavioural preorder, which is exactly the unknown result in table 1.

There are also a number of subsidiary questions that arise from this discussion. One obvious one is whether `upward simulations` have a role in a state-based notion of subtyping? Another direction is to look at the use of history properties in subtyping, and in particular is there a parallel notion in the context of refinement? In addition, can we find a behavioural characterization of subtyping with history properties, i.e. a testing preorder that characterizes subtyping for shared mutable objects? The subtyping rules we looked at so far have taken a form similar to `downward simulations`, however, alternative notions arise in the context of the fully abstract semantics defined in [48]. Although this offers a potential definition of subtyping, it is characterized at a behavioural level and it is an open issue whether there is a relational characterization similar to that of refinement. We can summarize these questions as follows:

- Can we find a behavioural characterization of subtyping?
- Do `upward simulations` have a role in a state-based notion of subtyping?
- Can we find a parallel notion to history properties in refinement?
- Is there a relational characterization for the complete-readiness model of subtyping?

In the next section we consider the first of these questions.

A BISIMULATION CHARACTERISATION OF $\leq$

In this section we review some work presented in [19], which gives a bisimulation characterisation of $\leq$. Its formulation is modulo the mapping of state based specifications into labelled transition systems presented in the second section. In particular, it works over the structural LTS model of state based specifications, i.e. the one without clouds added. Thus, two state based specifications are related by `downward simulation` iff their corresponding structural transition systems are related by the bisimulation characterisation.

The basic idea of the characterisation is to reformulate the standard pre and post-condition rules of state based refinement in terms of transitions in the labelled transition system. The resulting relation is called a 1/3-bisimulation[9].

Definition 3 *A* 1/3-bisimulation *is a binary relation* $R \subseteq \Sigma \times \Sigma$ *such that,*

1. $p\,R\,q \wedge q \xrightarrow{a} \wedge\ p \xrightarrow{a} p'$ *implies* $\exists q' \,.\, q \xrightarrow{a} q' \wedge p'\,R\,q'$
2. $p\,R\,q \wedge q \xrightarrow{a}$ *implies* $p \xrightarrow{a}$

In standard fashion, if there exists a 1/3-bisimulation *R between p and q then we say that p is* 1/3-bisimilar *to q and write* $p \leq_{1/3} q$[10]. *In addition, we can lift this definition from states to labelled transition systems. Thus, if* $p \leq_{1/3} q$ *we have* $l_p \leq_{1/3} l_q$.

[9] We call this a bisimulation because it has evolved from considering the 2/3-bisimulation relation. However, the bi here is not wholely appropriate as the relation R only functions in one direction. But, we continue with the name to maintain historical consistency.

[10] Compared to the definition in [19] we have reversed the direction of the relation, for consistency of presentation.

The intuition of 1/3-bisimulation is as follows:

- The first condition ensures that for the behaviour "defined"[11] by q, p cannot add non-determinism.
- The second condition ensures that in any two states that are related, every action that the abstract system can perform can also be performed at the concrete state.

What the relation enforces in terms of the traces of p and q is more subtle. In fact, it does not ensure either trace extension or trace reduction. The rules allow traces to be added as long as the "defined" behaviour of q is not compromised by this addition, i.e. non-determinism cannot be added. They also allow traces to be reduced if this arises from reduction of q's non-determinism.

The correspondence between 1/3-bisimulation and `downward simulation` can be seen quite clearly. The first condition plays the role of the post-condition rule of `downward simulation` (i.e. not allowing post-conditions to be enlarged), while the second condition plays the role of the pre-condition rule (i.e. it prevents pre-conditions from being shrunk). In addition, the linking of transition system states inherent in simulation and bisimulation characterisations, the R in definition 3, corresponds to the retrieve relation/abstraction function in the state based world.

The following result formalizes these correspondences.

Theorem 1 *$S_1 \leq S_2$ iff $l^{S_1} \leq_{1/3} l^{S_2}$ where l^{S_1} (respectively l^{S_2}) is the structural transition system of the state based specification S_1 (respectively S_2).*

Proof
Follows routinely from the state based to structural transition system mapping and the definitions of 1/3-bisimulation and `downward simulation`.
○

We can also observe where 1/3-bisimulation (and hence `downward simulation`) sits relative to other simulation style relations. [19] gives definitions of all the following relations and justifies the following statement:

simulation $\supset$ 1/3-bisimulation (= $\leq$) $\supset$ 2/3-bisimulation $\supset$ bisimulation

A TESTING CHARACTERISATION OF $\leq$

Testing and State Based Specification

Perhaps the central concept in testing is interaction - what happens when the environment attempts to perform a particular interaction, i.e. attempts to synchronize on a particular action. There are two issues to consider - what happens if the action is currently offered by the process and what happens if the action is not currently offered. The result in the former case is straightforward - the synchronisation is performed and then the process and environment proceed to their next potential interaction. However, the result in the latter case turns out to be less clear cut.

[11] The reason why we place this word in quotes will become clear shortly.

In the behavioural world, requiring interaction on an action that is not currently offered results in deadlock, or in other terms, the process will *refuse* the interaction. However, this outcome does not always carry over to the state based world.

The analogy in the state based world to requesting an interaction that is not currently offered, is applying an operation outside its pre-condition. As discussed in section 1, there are two possible interpretations; blocking and non-blocking. In a non-blocking model the outcome of performing the operation is only defined at states in which the pre-condition of the operation holds. In contrast, performing an operation at a state where its pre-condition does not hold yields *undefined*, i.e. the outcome is completely unpredictable. In an "operational sense" anything could occur and the choice between these alternatives is non-deterministic.

This alternative outcome is also pivotal in explaining why state based specifications allow functionality extension, since an operation whose outcome is undefined in a particular state can always be refined to an operation whose outcome is defined. In this way, the defined functionality of a specification can be extended.

In order to give a testing interpretation to `downward simulation` in a non-blocking model, we have to address this issue of undefined operations. Our approach is to take the structural labelled transition system generated from a state based specification and transform it into one containing extra behaviour, which reflects this alternative handling of undefined. We do this by adding *testing* clouds!!

Testing Cloud Construction

We considered the same issue of adding clouds in [9] and proposed a mapping which took an existing labelled transition system and added clouds. In this paper we employ a different cloud construction. The reasons for this are two fold. Firstly, [9] used internal actions in constructing clouds, since such actions do not arise in standard state based specification notations, we have avoided them in this paper. Secondly, [9] was considering the applicability of the LOTOS relation reduction for behavioural subtyping, thus, clouds there were constructed in terms of the top element of the reduction preorder. We consider a different refinement ordering here and a different top element is applicable.

The intuition behind our testing clouds is as follows:

> If in the original transition system an action cannot be performed at a particular state (i.e. the pre-condition of the corresponding operation is false at this state) then the transformed transition system should be able to perform the action, but the result of performing it will be completely unpredictable, i.e. it can evolve to offering any possible future behaviour.

Now we show how testing clouds can be added to behavioural specifications.

Definition 4 *Firstly, we define* $\mathcal{T}S = \{\mathcal{T}q \mid q \in S\}$ *where* $\mathcal{T}q$ *denotes a new distinct state in* S *and* $\mathcal{T}$ *is injective (i.e.* $\neg\exists p, q(p \neq q) . \mathcal{T}p = \mathcal{T}q$*). Then for* $l_p \in \mathrm{LTS}$ *we define* $l_{\mathcal{T}p}$*, the clouded* l_p*, as follows:-*

$$l_{\mathcal{T}p} = < \mathcal{T}S_p \cup \Sigma , \mathcal{T}p , \longrightarrow_{\mathcal{T}p} >$$

where, $\longrightarrow_{\mathcal{T}p}$ *is the smallest transition relation satisfying the following rules:-*

$$(T1)\ \frac{q \xrightarrow{a}_p q'}{\mathcal{T}q \xrightarrow{a}_{\mathcal{T}p} \mathcal{T}q'} \quad (T2)\ \frac{l_r \in \mathrm{LTS}\ \ q \xrightarrow{a}_p}{\mathcal{T}q \xrightarrow{a}_{\mathcal{T}p} r} \quad (T3)\ \frac{l_r \in \mathrm{LTS}\ \ q \xrightarrow{a}_r q'}{q \xrightarrow{a}_{\mathcal{T}p} q'}$$

This definition adds clouds at the states of l_p. Firstly, the existing behaviour of p is preserved (rule $T1$). Secondly, for any action that is not offered at a state, transitions on that action are added to any possible transition system (rule $T2$). Finally, rule $T3$ interprets each transition system.

Thus, the clouds that we add force their first action to be offered (i.e. this first action cannot be refined out). This is because we want the effect that all actions can be performed at every state and then for any initially undefined action we evolve to any possible future behaviour.

Of course an alternative to these testing clouds would be to work with clouds in the relational style, which were presented in section 1. However, we prefer our testing clouds since they complete the story started in [9] and also, they fit more naturally into a testing setting. However, the next subsection shows that our results carry over to relational clouds.

Testing

Using the notation,

$$l_p \sqsubseteq_{\mathcal{T}F} l_q \quad \text{iff} \quad l_{\mathcal{T}p} \sqsubseteq_F l_{\mathcal{T}q}$$
$$l_p =_{\mathcal{T}F} l_q \quad \text{iff} \quad l_{\mathcal{T}p} =_F l_{\mathcal{T}q} \quad \text{iff} \quad l_{\mathcal{T}p} \sqsubseteq_F \cap\, \sqsubseteq_F^{-1} l_{\mathcal{T}q}$$

an obvious possibility is that $\sqsubseteq_{\mathcal{T}F} = \leq_{1/3}$, i.e. $\leq_{1/3} = \sqsubseteq_F + clouds$[12]. However, this fails to hold[13], which we can see from $l_{\mathcal{T}p}$ and $l_{\mathcal{T}q}$ shown in figure 4. The triangles are the clouds that have been added by $\mathcal{T}$, each is annotated with the label(s) of the action(s) the cloud starts with (i.e. the initial non-deterministic choice). It can be verified that $l_{\mathcal{T}p} \leq_{1/3} l_{\mathcal{T}q}$, but $\neg(l_{\mathcal{T}q} \leq_{1/3} l_{\mathcal{T}p})$, i.e. non-determinism can be moved earlier, but not later. However, $l_{\mathcal{T}p} =_F l_{\mathcal{T}q}$. This is actually a classic example of how bisimulation/simulation relations and sequence based testing relations differ - typically sequence based testing relations cannot differentiate processes that only differ in how early/late a piece of non-determinism occurs. However, since bisimulation/simulation relations look more closely at the branching structure of processes, they can typically make such differentiation. Thus, we need a stronger notion of testing, than failures, that does not allow non-determinism to be moved later.

Ready Simulation Testing. The solution is to consider a tree based testing relation, called ready simulation testing, that does explore the branching structure. The resulting observations of the behaviour of the process are constructed using a simple modal logic, which codes up the observation tree.

[12] In fact, [9] suggested a strong relationship between behavioural subtyping and failures + clouds. However, the full story, which is presented in this paper, is somewhat more complicated than anticipated there.

[13] In fact, as a consequence of our theorem we will be able to see that $\leq_{1/3} \subset \sqsubseteq_{\mathcal{T}F}$. This is because $\preceq_{RS} \subset \sqsubseteq_F$.

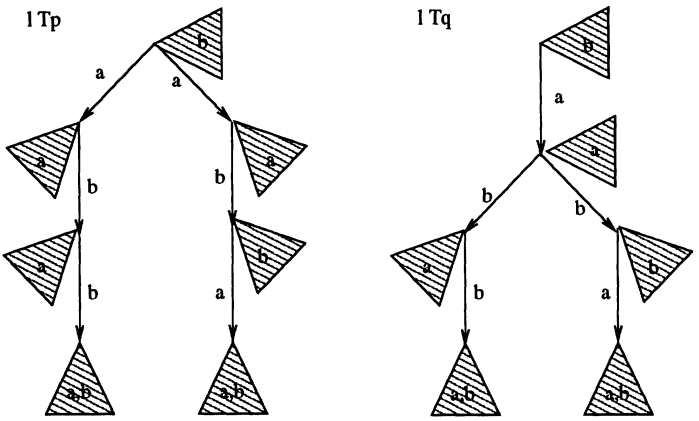


Figure 4 Early and Late Non-determinism

Definition 5 *The logic, denoted* RSL, *is called* ready simulation *logic and for $a \in A$ and $X \subseteq A$ an arbitrary formula ϕ is characterised by the following syntax:*

$$\phi := True \mid \phi \wedge \phi \mid a\phi \mid X$$

For example, the following are some of the observations that both $l_{\mathcal{T}p}$ and $l_{\mathcal{T}q}$ yield:-

$$abaTrue \quad aba\{a,b\} \quad a\{a,b\} \quad abTrue \wedge aTrue \quad ab\{a,b\} \wedge aTrue$$

The generalisation of sequence based testing arises because not only can we express traces - the action sequences, and refusal/acceptance properties - the set of actions X, we can also express branching using $\wedge$.

The capability of a process to yield an observation is expressed using logical satisfaction, $\models \subseteq$ LTS $\times$ RSL. Thus, $l_p \models \phi$ means intuitively that when tested the transition system l_p can exhibit the observable behaviour defined by ϕ. Satisfaction is defined inductively over the structure of formulae, as follows:

$$\begin{aligned} p &\models True \\ p &\models \phi \wedge \psi \quad \text{iff} \quad p \models \phi \; and \; p \models \psi \\ p &\models a\phi \quad \text{iff} \quad \exists p' . p \xrightarrow{a} p' \; and \; p' \models \phi \\ p &\models X \quad \text{iff} \quad out(p) = X \end{aligned}$$

Using $\models$, we define S : LTS $\rightarrow \mathbb{P}$(RSL), which yields the set of formulae that a particular labelled transition system satisfies, i.e., $S(p) = \{ \phi \mid p \models \phi \}$. In other words, $S(p)$ is the set of all observations that p can exhibit and we can define a testing preorder, which we call ready simulation preorder and denote $\preceq_{RS}$, as follows:

$$l_p \preceq_{RS} l_q \text{ iff } S(p) \subseteq S(q)$$

Intuitively, all the observations that can be made of l_p could have been made of l_q.

As an indication that $\preceq_{RS}$ is the appropriate preorder consider $l_{\mathcal{T}p}$ and $l_{\mathcal{T}q}$ shown in figure 4. Although laborious, it can be shown that, $S(\mathcal{T}p) \subset S(\mathcal{T}q)$ and the strictness of this subset inclusion arises since, for example,

$$\phi = a(bbTrue \wedge baTrue) \in \mathcal{T}q \quad \text{but} \quad \phi \notin \mathcal{T}p$$

The following theorem, which is verified in [10] shows that ready simulation preorder is indeed the relation we seek.

Theorem 2 $\forall l_p, l_q \in \text{LTS} \; . \mathcal{T}p \preceq_{RS} \mathcal{T}q \iff p \leq_{1/3} q.$

Alternative (Relational) Cloud Construction

Relational clouds are added slightly differently to testing clouds.

Definition 6 *Firstly, adding relational clouds only adds one new state to the existing states in the transition system, $\perp$. Thus, for $l_p \in \text{LTS}$ we define $l_{\mathcal{R}P}$, the Relational clouded l_p, as follows:-*

$$l_{\mathcal{R}p} = < \mathcal{R}S_p \cup \{\perp\} \, , \, \mathcal{R}p \, , \longrightarrow_{\mathcal{R}p} >$$

where, as with $\mathcal{T}$, $\mathcal{R}q$ denotes a new distinct state and $\mathcal{R}$ is injective and $\longrightarrow_{\mathcal{R}p}$ is the smallest transition relation satisfying the following rules:-

$$(T1) \; \frac{q \xrightarrow{a}_p q'}{\mathcal{R}q \xrightarrow{a}_{\mathcal{R}p} \mathcal{R}q'} \qquad (T2) \; \frac{q \not\xrightarrow{a}_p \quad r \in \mathcal{R}S_p \cup \{\perp\}}{\mathcal{R}q \xrightarrow{a}_{\mathcal{R}p} r}$$

The original transition system is enhanced as follows. For any action that is not offered at a state, transitions on that action are added to all states in the system and additionally to the non-termination state, $\perp$.

[10] shows that with appropriate handling of $\perp$ in the ready simulation preorder a corresponding result to theorem 2 can be verified with relational clouds.

CONCLUSIONS

We have reviewed the field of research on relating state based and behavioural specification formalisms. To this end, we have considered how state based techniques can be interpreted in behavioural semantics and related preorder (refinement) relations in the two paradigms. For the latter of these we have highlighted behavioural interpretations of a spectrum of different state based preorders. In particular, we have added to the existing classification by giving a behavioural interpretation to `downward simulation` in a non-blocking setting. This is an important result because downward simulation is possibly the most frequently used state based refinement relation and is also very closely related to state based subtyping.

Our results have shown that with appropriate cloud constructions (which model a non-blocking interpretation), `downward simulation` can be behaviourally characterised as ready simulation testing. We also reviewed the existing result that

`downward simulation` corresponds to 1/3-bisimulation when clouds are not added, i.e. structural transition systems are taken.

Thus, in this paper, we have reviewed and added to the existing theory at the junction of state based and behavioural notations. However, as highlighted at the end of section 3, a number of outstanding issues remain:-

- Can we find a behavioural characterization of subtyping?
- Do `upward simulations` have a role in state-based subtyping?
- Can we find a parallel notion to history properties in refinement?
- Is there a relational characterization for the complete-readiness model of subtyping?

While the last three of these questions remain completely open, our characterisation of `downward simulation` makes a contribution to answering the first. However, to fully answer this question we would have to know how the history properties found in state based subtyping can be interpreted behaviourally. Our conjecture is that history properties limit the underlying cloudyness of state based specifications. In the sense that they constrain where new functionality can be added (in a sense they enforce actual refusals). Thus, by adding clouds selectively to the "defined" part of state based specifications, we conjecture that it would be possible to handle history properties in the framework we have set up in this paper.

Acknowledgments

The authors have collaborated with a number of people on particular aspects of what is presented in this paper. We would thus like to acknowledgement the contribution of these collaborators, we mention in particular, Eerke Boiten, Charles Briscoe-Smith, Tommaso Bolognesi, Peter Linington, Tim Regan, Graeme Smith, Maarten Steen and Ben Strulo.

References

[1] J. R. Abrial. *The B-Book: Assigning programs to meanings.* CUP, 1996.

[2] M. Benjamin. A message passing system: An example of combining CSP and Z. In J. E. Nicholls, editor, *Z User Workshop*, Workshops in Computing, pages 221–228, Oxford, 1989. Springer-Verlag.

[3] E. Boiten, H. Bowman, J. Derrick, and M. Steen. Viewpoint consistency in Z and LOTOS: A case study. In J. Fitzgerald, C. B. Jones, and P. Lucas, editors, *Formal Methods Europe (FME '97)*, Lecture Notes in Computer Science 1313, pages 644–664, Graz, Austria, September 1997. Springer-Verlag.

[4] E.A. Boiten and J. Derrick. IO-refinement in Z. In *3rd BCS-FACS Northern Formal Methods Workshop*, Electronic Workshops in Computing. Springer Verlag, September 1998.

[5] T. Bolognesi. Expressive flexibility in constraint-oriented specification: Lotos and co-notation. In H. Bowman and J. Derrick, editors, *FMOODS'97, Formal Methods for Open Object-based Distributed Systems.* Chapman and Hall, 1997.

[6] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1988.

[7] G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modelling Language*, 1997. Rational Software Corporation, Santa Clara, CA-95051, USA.

[8] J. P. Bowen, A. Fett, and M. G. Hinchey, editors. *ZUM'98: The Z Formal Specification Notation, 11th International Conference of Z Users, Berlin, Germany, 24–26 September 1998*, volume 1493. Springer-Verlag, 1998.

[9] H. Bowman, C. Briscoe-Smith, J. Derrick, and B. Strulo. On behavioural subtyping in LOTOS. In H. Bowman and J. Derrick, editors, *FMOODS'97, 2nd IFIP Conference on Formal Methods for Open Object Based Distributed Systems.* Chapman and Hall, July 1997.

[10] H. Bowman and J. Derrick. A junction between state based and behavioural specification - full version. Technical Report 15-98, U of Kent at Canterbury, 1998.

[11] E. Brinksma, G. Scollo, and C. Steenbergen. Process specification, their implementation and their tests. In *Protocol Specification, Testing and Verification, VI*, pages 349–360, Montreal, Canada, June 1986. North-Holland.

[12] S.D. Brookes and A.W. Roscoe. An improved failures model for communicating processes. In *Pittsburgh Symposium on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 281–305. Springer-Verlag, 1985.

[13] CCITT Z.100. *Specification and Description Language SDL*, 1988.

[14] E. Cusack. Inheritance in object oriented Z. In P. America, editor, *Proc. ECOOP'91 European Conference on Object-Oriented Programming*, volume 512, pages 167–179. Springer-Verlag, 1991.

[15] E. Cusack and G. H. B. Rafsanjani. ZEST. In S. Stepney, R. Barden, and D. Cooper, editors, *Object Orientation in Z*, Workshops in Computing, pages 113–126. Springer-Verlag, 1992.

[16] E. Cusack and C. Wezeman. Deriving tests for objects specified in Z. In J. P. Bowen and J. E. Nicholls, editors, *Seventh Annual Z User Workshop*, pages 180–195, London, December 1992. Springer-Verlag.

[17] J. Derrick and E.A. Boiten. Refinement of state-based systems with internal operations. Under revision, 1998.

[18] J. Derrick, E.A. Boiten, H. Bowman, and M. Steen. Specifying and refining internal operations in Z. *Formal Aspects of Computing*, 1999. To appear.

[19] J. Derrick, H. Bowman, E. Boiten, and M. Steen. Comparing LOTOS and Z refinement relations. In *FORTE/PSTV'96*, pages 501–516, Kaiserslautern, Germany, October 1996. Chapman & Hall.

[20] R. Duke, G. Rose, and G. Smith. Object-Z: A specification language advocated for the description of standards. *Comp. Standards and Interfaces*, 17:511–533, 1995.

[21] H. Ehrich, J. Goguen, and A. Sernadas. A categorical theory of objects as observed processes. In J.W. Bakker, W.P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages*, volume 489 of *Lecture Notes in Computer Science*, pages 203–228. Springer-Verlag, 1991.

[22] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: a framework for integrating multiple perspectives in system development. *International Journal on Software Engineering and Knowledge Engineering, Special issue on Trends and Research Directions in Software Engineering Environments*, 2(1):31–58, March 1992.

[23] C. Fischer. CSP-OZ - a combination of CSP and Object-Z. In H. Bowman and J. Derrick, editors, *2nd IFIP int conference on Formal Methods for Open Object-based Distributed Systems*, pages 423–438. Chapman & Hall, July 1997.

[24] C. Fischer. How to combine Z with a process algebra. In Bowen et al. [8], pages 5–23.

[25] C. Fischer and G. Smith. Combining CSP and Object-Z: Finite or infinite trace semantics. In T. Higashino and A. Togashi, editors, *FORTE/PSTV'97*, pages 503–518, Osaka, Japan, November 1997. Chapman & Hall.

[26] A. Galloway and W. Stoddart. An operational semantics for ZCCS. In Hinchey and Liu [30], pages 272–282.

[27] The RAISE Language Group. *The RAISE Specification Language*. Prentice Hall, 1992.

[28] He Jifeng, C. A. R. Hoare, and J. W. Sanders. Data refinement refined. In B. Robinet and R. Wilhelm, editors, *Proc. ESOP 86*, volume 213, pages 187–196. Springer-Verlag, 1986.

[29] He Jifeng and C.A.R. Hoare. Prespecification and data refinement. In *Data Refinement in a Categorical Setting*, number PRG-90 in Technical Monograph. Oxford University Computing Laboratory, November 1990.

[30] M. G. Hinchey and Shaoying Liu, editors. *Formal Engineering Methods: Proc. 1st International Conference on Formal Engineering Methods (ICFEM'97)*, Hiroshima, Japan, 12–14 November 1997. IEEE Computer Society Press.

[31] C.A.R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice-Hall, 1985.

[32] ISO 9074. *Estelle, a Formal Description Technique based on an extended state transition model*, June 1987.

[33] D. Jackson. Structuring Z specifications with views. *ACM Transactions on Software Engineering and Methodology*, 4(4), October 1995.

[34] C. B. Jones. *Systematic Software Development using VDM*. Prentice Hall, 1989.

[35] M.B. Josephs. A state-based approach to communicating processes. *Distributed Computing*, 3:9–18, 1988.

[36] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J-M. Loningtier, and J. Irwin. Apsect-oriented programming. Technical Report SPL97-008 P9710042, PARC, 1997. See http://www.parc.xerox.com/spl/projects/aop/reports.html.

[37] K. C. Lano and S. Goldsack. Integrated formal and object-oriented methods: The VDM^{++} approach. In T. Bryant and L. Semmens, editors, *Methods Integration*, Electronic Workshops in Computing. Springer-Verlag, 1996.

[38] K. Larsen and A. Skou. Bisimulation through probabilistic testing. In *Sixteenth Annual ACM Symp on Principles of Programming Languages*. ACM Press, 1989.

[39] P. F. Linington. RM-ODP The Architecture. In K. Raymond and L. Armstrong, editors, *IFIP TC6 International Conference on Open Distributed Processing*, pages 15–33, Brisbane, Australia, February 1995. Chapman and Hall.

[40] B. Liskov and J. M. Wing. Specifications and their use in defining subtypes. In J. P. Bowen and M. G. Hinchey, editors, *ZUM'95: The Z Formal Specification Notation*, volume 967, pages 246–263. Springer-Verlag, 1995.

[41] B. Mahony and J. S. Dong. Adding timed concurrent processes to Object-Z: A case study in TCOZ. In Bowen et al. [8], pages 308–327.

[42] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice-Hall, 1989.

[43] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100:1–77, 1992.

[44] C. C. Morgan and T. Vickers, editors. *On the Refinement Calculus*. Formal Approaches to Computing and Information Technology series (FACIT). Springer-Verlag, 1994.

[45] E.-R. Olderog and C.A.R. Hoare. Specification oriented semantics for communicating process. *Acta Informatica*, 23:9–26, 1986.

[46] A.W. Roscoe, J. C. P. Woodcock, and L.Wulf. Non-interference through determinism. In D. Gollmann, editor, *ESORICS'94*, volume 875 of *LNCS*, pages 33–54. Springer-Verlag, 1994.

[47] S. A. Schumann, D. H. Pitt, and P. J. Byers. Object-oriented process specification. In C. Rattray, editor, *Specification and Verification of Concurrent Systems*, Workshops in Computing, pages 21–70. Springer-Verlag, 1990.

[48] G. Smith. A fully abstract semantics of classes for Object-Z. *Formal Aspects of Computing*, 7(3):289–313, 1995.

[49] G. Smith. A semantic integration of Object-Z and CSP for the specification of concurrent systems. In *Formal Methods Europe (FME '97)*, volume 1313 of *LNCS*, pages 62–81. Springer-Verlag, September 1997.

[50] G. Smith and J. Derrick. Refinement and verification of concurrent systems specified in Object-Z and CSP. In M. Hinchey and Shaoying Liu, editors, *First IEEE International Conference on Formal Engineering Methods (ICFEM '97)*, pages 293–302, Hiroshima, Japan, November 1997. IEEE Computer Society.

[51] J. M. Spivey. *The Z notation: A reference manual*. Prentice Hall, 1989.

[52] S. Stepney, D. Cooper, and J. C. P. Woodcock. More powerful data refinement in Z. In Bowen et al. [8], pages 284–307.

[53] B. Strulo. How firing conditions help inheritance. In J. P. Bowen and M. G. Hinchey, editors, *Ninth Annual Z User Workshop*, LNCS 967, pages 264–275, Limerick, September 1995. Springer-Verlag.

[54] K. Taguchi and K. Araki. The state-based CCS semantics for concurrent Z specification. In Hinchey and Liu [30], pages 283–292.

[55] F.W. Vaandrager. Process algebra semantics for POOL. Technical Report CS-R8629, CWI, Amsterdam, the Netherlands, 1991.

[56] R.J. van Glabbeek. The linear time - branching time spectrum. In *CONCUR 90*, LNCS 458, pages 278–297, Amsterdam, 1990. Springer-Verlag.

[57] J.C.P. Woodcock and C.C. Morgan. Refinement of state-based concurrent systems. In D. Bjorner, C.A.R. Hoare, and H. Langmaack, editors, *VDM'90:VDM and Z!*, volume 428 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[58] A. Yonezawa and M. Tokoro, editors. *Object-Oriented Concurrent Programming*. MIT Press, 1987.