Formal Aspects of Computing (1998) 3: 1–36

Specifying and Refining internal operations in Z

John Derrick, Eerke Boiten, Howard Bowman and Maarten Steen*

Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK.
(Phone: + 44 1227 764000, Email: J.Derrick@ukc.ac.uk.)

Keywords: Z; Refinement; Distributed Systems; Internal Operations; Process Algebras; Concurrency.

Abstract. An important aspect in the specification of distributed systems is the role of the internal (or unobservable) operation. Such operations are not part of the interface to the environment (i.e. the user cannot invoke them), however, they are essential to our understanding and correct modelling of the system. In this paper we are interested in the use of the formal specification notation Z for the description of distributed systems. Various conventions have been employed to model internal operations when specifying such systems in Z. If internal operations are distinguished in the specification notation, then refinement needs to deal with internal operations in appropriate ways.

Using an example of a telecommunications protocol we show that standard Z refinement is inappropriate for refining a system when internal operations are specified explicitly. We present a generalization of Z refinement, called weak refinement, which treats internal operations differently from observable operations when refining a system. We discuss the role of internal operations in a Z specification, and in particular whether an equivalent specification not containing internal operations can be found. The nature of divergence through livelock is also discussed.

Correspondence and offprint requests to: John Derrick, Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK.

* This work was partially funded by British Telecom Research Labs., and the EPSRC under grant number GR/K13035.

1. Introduction

The Z specification language [Spi89] has gained a certain amount of acceptance in the software community as an industrial strength formal method. Z is a state-based language based upon set theory and first order logic. The most common style of specification in Z is the so called "state plus operations" style, where a collection of operations describe changes to the state space. The state space and operations are described as *schemas*, and the schema calculus has proved to be an enduring structuring mechanism for specifying complex systems.

A growing literature and a number of industrial case studies have demonstrated the usability of the language, and attention is being turned to new domains of applicability - one such example being the use of Z for the specification of concurrent and distributed systems [Cus91, Rud91, MZ94, Lam94, Str95]. However, concurrent and distributed systems place a number of requirements on notations used to specify such systems, and, in particular, one aspect that is important is the role of the internal (or unobservable) operation. Internal operations are not part of the interface to the environment (i.e. the user cannot invoke them), however, they are essential to our understanding and correct modelling of the system. Such operations (or actions) arise naturally in distributed systems, either as a result of modelling concurrency or the non-determinism that is inherent in a model of such a system. For example, internal operations can be used to model communication (e.g. as in the language CCS [Mil89]), non-determinism arises as a by-product of this interpretation. Internal operations are also central to obtaining abstract specification through hiding, a particularly important example of this is to enable communication to be internalised - a central facet in the design of distributed systems.

The majority of formal notations which have been designed with concurrent systems in mind have a notion of internal action, event or operation as part of the language or its semantics. Examples include CCS [Mil89], CSP [Hoa85] and LOTOS [BB88]. In particular, internal events have an important role in the theory of process algebras, and a special symbol is reserved for the occurrence of such an internal event (e.g. i in LOTOS or τ in CCS).

In addition to the description, i.e., specification, of a system an important benefit that formal methods offer is the ability to develop a system's specification according to some theory of refinement in that language. Examples include the use of refinement in Z [Spi89, WD96], VDM [Jon89] or bisimulation in a process algebra [Mil89]. However, if internal events are distinguished in a particular specification notation, then the theory of refinement in that language should deal with such internal events in an appropriate way. One way is to treat an internal event no differently from observable events, the strong bisimulation relation in a process algebra is an example of an equivalence relation adopting such a convention. However, it is well recognised that strong bisimulation is inappropriate as a refinement relation because it discriminates too many specifications that might reasonably be seen as equivalent. Therefore internal events in refinement and equivalence relations typically have a different role than the observable events of the system. Examples of relations in which the observable is differentiated from the internal are weak bisimulation [Mil89], testing equivalence [Bri88], reduction and extension [BSS86], failures refinement [Hoa85] and Hennessy's testing pre-orders [Hen88]. Central to these relations is the understanding that internal events are unobservable, and that refinement relations must refine the observable behaviour of a specification differently from the internal aspects of its behaviour.

Now that Z is being used for the specification of concurrent or distributed systems, a number of authors have recognised the need to explicitly specify internal operations separately from the observable interface, and a number of conventions have been adopted for their description. In each case the internal operation is specified as normal and either has a distinguished name or informal commentary telling us that it is not part of the interface to the environment (we will see examples of both approaches below). This approach immediately raises two questions. Firstly, is it possible to dispense with such internal operations by adding their behaviour to the observable interface in some fashion? Secondly, if internal operations are to appear explicitly in a Z specification, we need to consider the possibility of refining these specifications. How should we treat the refinement of internal operations in Z? This paper seeks to address these issues. In particular, we shall show that the standard Z refinement rules are inappropriate for the refinement of internal operations. We make a proposal called *weak refinement* which seeks to offer a correct generalisation of refinement when specifications contain internal operations. This has a similar relation to ordinary Z refinement as weak bisimulation does to strong bisimulation in a process algebra. In particular, we define weak refinement by considering the stand point of an external observer of the system, who manipulates operations in the user interface.

Such an external observer will require that a retrieve relation is still defined between the state spaces of the abstract and concrete specifications and that each abstract observable operation AOp is recast as a concrete observable operation COp. The weak refinement relation is defined to ensure that the observable behaviour of the concrete specification is a refinement of the observable behaviour of the abstract specification.

We will also consider to what extent internal operations are necessary and whether we can dispense with them. For specifications that do not contain livelock (i.e., infinite sequences of internal events) we will argue that we can dispense with the explicit use of internal operations in the specification. For specifications containing divergence in the form of livelock whether we can dispense with their explicit specification will turn out to depend on the interpretation of divergence used.

Throughout the paper we assume the state plus operations style of Z specification, and our discussion takes place within that context.

The structure of the paper is as follows. In Section 2 we review the use of internal operations in Z specifications. Section 3 presents an example of a specification and refinement involving internal operations, the example illustrates that standard Z refinement is inappropriate in the presence of internal operations. Section 4 formulates the generalization that we call weak refinement, which is motivated by the treatment of internal events in process algebras. Section 5 revisits the protocol example to show that weak refinement has the required properties of a refinement where internal operations have been specified. Section 6 considers whether we can dispense with internal operations and the role of divergence in answering that question. Section 7 discusses some properties of weak refinement, related work is then reviewed in Section 8, and we conclude in Section 9.

2. Internal Operations

In the traditional approach to the specification of sequential systems in Z, the operations specified represent the interface to the environment. That is, a state change occurs in the system if and only if the environment invokes one of the operations. Each operation therefore represents a potential *observable* event of the system under construction, and this is usually an acceptable model. However, when modelling concurrent and distributed systems it is convenient to model *internal* events. These internal events represent operations over which the environment has no control (hence the name internal), but are still necessary to specify in a full description of the system. Since they are not part of the environmental or user interface they can be invoked by the system whenever their pre-conditions hold. They can arise either due to the natural non-determinism of a distributed system [Hoa85], or due to communication within the system [Mil89] or due to some aspect of the system being hidden at this level of abstraction [BB88]. The necessity for the specification of internal events in process algebras is well recognised [Mil89], and a number of researchers have found it convenient or necessary to specify internal operations in Z when specifying distributed systems [CW92, WJ94, Raf94, Str95, WD96, DBBS96a].

For example, Strulo [Str95] considers the use of Z in network management and describes the need for both observable and internal operations in this application area. A particular example is described of a network manager's view of a router within a network. There, alarm notifications are a typical example of internal events which are specified as usual but with informal commentary describing which operations are observable and which are internal. A similar approach and application area is described in [WJ94, Raf94].

Cusack and Wezeman, in [CW92], adopt a number of conventions for the use of Z for the specification of OSI network management standards. In particular, they make the distinction between internal and observable operations according to whether an operation has input/output: *operations which use $\Delta State$ but have neither input or output variables are internal (unobservable) actions, corresponding to the internal event in LOTOS. All other operations can be thought of as interactions with the environment, or external operations [CW92].* Their work is placed in an object-oriented setting and they consider notions of subtyping based upon conformance instead of refinement.

In [DBBS96a] a distinguishing name (i) is used to denote which operations are internal. The motivation there was to provide a direct mapping between events in LOTOS and operations in Z in order to support the use of multiple viewpoints in the Open Distributed Processing reference model [ITU95].

Woodcock and Davies [WD96] also use informal commentary to describe which operations are internal and which are observable. They also comment on whether these internal operations add to the expressive power of the language, saying: *It should be clear that we could dispense with such operations, but only by adding the required degree of non-determinism to the remainder of the specification.* We will give a constructive proof of this statement in Section 6.

Evans in [Eva97] considers the use of Z for the specification of parallel systems, and in particular discusses issues of liveness and fairness in dynamic specifications. Internal operations are specified as in [WD96], and he also considers the refinement relations needed for Z specifications of concurrent systems. Similar work has appeared in other state-based formalisms. For example, Butler [But97] considers the specification and refinement of internal actions in the B method

[Abr96]. There, internal actions are specified explicitly in an abstract machine. Additional work in this area also includes the work of Lano, e.g. [Lan97].

In each case the internal operation is specified as normal and either has a distinguished name or informal commentary telling us that it is not part of the user interface. We will see examples of both below. Used in this way, Z is clearly sufficient as a notation for the specification of internal operations or events, and as can be seen from the examples referenced above, internal events are needed when Z is used to specify parts of a distributed system which contain large amounts of state information. Typical of this application area are managed objects or the information viewpoint of the Open Distributed Processing reference model, where the specifications contain a lot of state but there is also a need to model internal operations such as alarms.

This section has reviewed the use of internal operations in Z specifications, the next section considers an example of their specification and refinement.

3. Refinement

A Z specification describes the state space together with a collection of operations. The Z refinement relation [Spi89, WD96], defined between two Z specifications, allows both the state space and the individual operations to be refined in a uniform manner[†].

Operation refinement is the process of recasting each abstract operation AOp into a concrete operation COp, such that, informally, the following holds. The pre-condition of COp may be weaker than the pre-condition of AOp, and COp may have a stronger post-condition than AOp. That is, COp must be applicable whenever AOp is, and if AOp is applicable, then every state which COp might produce must be one of those which AOp might produce. Data refinement extends operation refinement by allowing the state space of the concrete operations to be different from the state space of the abstract operations.

Consider an abstract specification with state space $Astate$, operation AOp, and initialisation $Ainit$, and a refined specification with state space $Cstate$, operation COp, and initialisation $Cinit$. Refinement is defined in terms of an abstraction schema or retrieve relation, usually called Ret, $Retrieve$ or Abs, which relates the abstract and concrete states. It has the same signature as $Astate \wedge Cstate$, and its property holds if the concrete state is one of those which represent the abstract state [Spi89]. The retrieve relation does not need to be total nor functional. The concrete specification is a refinement of the abstract specification if the following conditions hold:

Initialisation $\forall\, Cstate' \bullet Cinit \vdash \exists\, Astate' \bullet Ainit \wedge Ret'$

Applicability $\forall\, Astate;\ Cstate \bullet \mathrm{pre}\, AOp \wedge Ret \vdash \mathrm{pre}\, COp$

Correctness $\forall\, Astate;\ Cstate;\ Cstate' \bullet \mathrm{pre}\, AOp \wedge Ret \wedge COp \vdash \exists\, Astate' \bullet Ret' \wedge AOp$

An illustration of refinement will be given in the following subsection.

There is a growing body of experience and literature concerning refinement in the traditional context of sequential systems specified in Z, e.g. [WD96]. However,

[†] We consider only refinements defined by forward simulations in this paper. Similar results could be obtained for backwards simulations if needed.

these refinement rules assume all operations are observable. How does refinement behave if some of the operations are internal or unobservable?

As an illustration of refinement involving internal operations we consider the specification and refinement of a telecoms protocol (the Signalling System No. 7 standard) adapted from [WD96, HMR89]. The first specification defines the external view of the protocol, subsequently we develop a sectional view which specifies the route that messages take through the protocol. [HMR89] discusses the formalisation of the informal specification in more depth, our purpose here is to use the formalisation given in [WD96] as an illustrative example.

3.1. Specification 1: the external view

Let M be the set of messages that the protocol handles. The state of the system is represented by the state schema *Ext*, and comprises two sequences which represent messages that have arrived in the protocol (in), and those that have been forwarded (out).

$$
\begin{array}{|l}
\textit{Ext} \\
\hline
in, out : \operatorname{seq} M \\
\hline
\exists\, s : \operatorname{seq} M \bullet in = s \frown out \\
\hline
\end{array}
$$

Incoming messages are added to the left of in, and the messages contained in in but not in out represent those currently inside the protocol. The state invariant specifies that the protocol must not corrupt or re-order. Initially, no messages have been sent, and this is specified by the following initialisation schema:

$$\textit{ExtInit} \mathrel{\widehat{=}} [\, \textit{Ext}' \mid in' = \langle\,\rangle \,]$$

The specification at this level is completed by the description of two operations which model the transmission (*Transmit*) and reception (*Receive*) of messages into and out of the protocol. In the specification of the *Receive* operation, either no message is available (e.g. all messages are en route in the protocol) or the next one is output, at this level of abstraction this choice is made non-deterministically. The specifications are straightforward‡.

$$
\begin{array}{|l}
\textit{Transmit} \\
\hline
\Delta \textit{Ext} \\
m? : M \\
\hline
in' = \langle m? \rangle \frown in \\
out' = out \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{Receive} \\
\hline
\Delta \textit{Ext} \\
\hline
in' = in \\
\# out' = \# out + 1 \lor out' = out \\
\hline
\end{array}
$$

‡ The *Receive* operation could, if desired, actually output the transmitted value, however this is immaterial to our concerns here.

3.2. Specification 2: the sectional view

The second specification describes the sectional view which specifies the route the messages take through the protocol in terms of a number of *sections*. Each section in the protocol may receive and send messages, and those which have been received, but not yet sent on, are in the section. The messages pass through the sections in order. Let N be the number of sections. In the state schema, $ins\ i$ represents the messages currently inside section i, $rec\ i$ the messages that have been received by section i, and $sent\ i$ the messages that have been sent onwards from section i. The state and initialisation schemas are then given by

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Section}} \\
rec, ins, sent : \mathrm{seq}(\mathrm{seq}\, M) \\
\hline
N = \#rec = \#ins = \#sent \\
rec = ins \frown\frown sent \\
front\ sent = tail\ rec \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SectionInit}} \\
Section' \\
\hline
\forall\, i : 1..N \bullet \\
\quad rec'\ i = ins'\ i = sent'\ i = \langle\,\rangle \\
\hline
\end{array}
$$

where $\frown\frown$ denotes pairwise concatenation of the two sequences (so for every i we have $rec\ i = ins\ i \frown sent\ i$). The predicate $front\ sent = tail\ rec$ ensures that messages that are sent from one section are those that have been received by the next. This specification also has operations to transmit and receive messages, and they are specified as follows:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{STransmit}} \\
\Delta Section \\
m? : M \\
\hline
head\ rec' = \langle m? \rangle \frown (head\ rec) \\
tail\ rec' = tail\ rec \\
sent' = sent \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SReceive}_0} \\
\Delta Section \\
\hline
rec' = rec \\
front\ ins' = front\ ins \\
last\ ins' = front(last\ ins) \\
front\ sent' = front\ sent \\
last\ sent' = \langle last(last\ ins) \rangle \frown (last\ sent) \\
\hline
\end{array}
$$

$SReceive \mathrel{\widehat=} SReceive_0 \lor \Xi Section$

Here, the new message received is added to the first section in the route by the operation *STransmit*.

The operation *SReceive* will deliver a message from the last section in the route. In the external view presented above, messages arrive non-deterministically because we did not model the interior of the protocol. In the sectional view this non-determinism is represented by the progress of the messages through the sections. Therefore in this more detailed design, we need to specify how the messages make progress through the sections. We do so by defining an operation *Daemon* which non-deterministically selects a section to make progress. The oldest message is then transferred to the following section, and nothing else changes. The *important* part of this operation is given by:

$$
\begin{array}{|l}
\hline
\textit{Daemon}_0 \\
\Delta \textit{Section} \\
\hline
\exists\, i : 1..N-1 \mid \\
\quad \textit{ins}\ i \neq \langle\,\rangle \bullet \\
\quad \textit{ins}'\,i = \textit{front}(\textit{ins}\ i) \\
\quad \textit{ins}'(i+1) = \langle \textit{last}(\textit{ins}\ i)\rangle \frown \textit{ins}(i+1) \\
\quad \forall\, j : 1..N \mid j \neq i \land j \neq i+1 \bullet \textit{ins}'j = \textit{ins}\ j \\
\hline
\end{array}
$$

The informal commentary accompanying the specification tells us that *Daemon* is an internal operation, and so can be invoked by the system whenever its precondition holds. As noted in [WD96]: *This operation is not part of the user interface. The user cannot invoke Daemon, but it is essential to our understanding of the system and to its correctness.*

The sectional view is in some way a refinement of the external view, where the retrieve relation is given by:

$$
\begin{array}{|l}
\hline
\textit{Retrieve} \\
\textit{Ext} \\
\textit{Section} \\
\hline
\textit{head rec} = \textit{in} \\
\textit{last sent} = \textit{out} \\
\hline
\end{array}
$$

We note that the retrieve relation used here is a total function, i.e., $\forall\, \textit{Section} \bullet \exists_1\, \textit{Ext} \bullet \textit{Retrieve}$.

Under this refinement *STransmit* and *SReceive* correspond to *Transmit* and *Receive* respectively, and the internal operation *Daemon* corresponds to the external operation $\Xi \textit{Ext}$, i.e. the identity operation on *Ext*. The refinement is proved correct by showing that (where we have omitted the appropriate quantification over the states):

$$
\begin{array}{l}
\textit{SectionInit} \land \textit{Retrieve}' \Rightarrow \textit{ExtInit} \\
\text{pre } \textit{Transmit} \land \textit{Retrieve} \Rightarrow \text{pre } \textit{STransmit} \\
\text{pre } \textit{Transmit} \land \textit{Retrieve} \land \textit{STransmit} \land \textit{Retrieve}' \Rightarrow \textit{Transmit} \\
\text{pre } \textit{Receive} \land \textit{Retrieve} \Rightarrow \text{pre } \textit{SReceive} \\
\text{pre } \textit{Receive} \land \textit{Retrieve} \land \textit{SReceive} \land \textit{Retrieve}' \Rightarrow \textit{Receive} \\
\text{pre } \Xi \textit{Ext} \land \textit{Retrieve} \Rightarrow \text{pre } \textit{Daemon} \\
\text{pre } \Xi \textit{Ext} \land \textit{Retrieve} \land \textit{Daemon} \land \textit{Retrieve}' \Rightarrow \Xi \textit{Ext}
\end{array}
$$

The refinement is discussed in [WD96]. This completes the first refinement of the external view.

Let us summarise the situation so far. We can specify a system that contains non-determinism in some of the operations in its user interface (e.g. *Receive*), but which does not contain any internal operations. We can then refine this specification to one that contains internal operations that correctly models (in the sense of a refinement existing between the specifications) the abstract specification. We have used the standard Z refinement relations, which have been perfectly adequate at this level.

3.3. Specification 3: refining internal operations

However, let us look at the refinement of the internal operation *Daemon* again. As it stands $Daemon_0$ represents the functionality that for non-empty sections ($ins\ i \neq \langle\,\rangle$) we transfer a message along the sections. But in order that the complete operation *Daemon* refines ΞExt, $Daemon_0$ must be extended to ensure that

$$\mathrm{pre}\,\Xi Ext \land Retrieve \Rightarrow \mathrm{pre}\,Daemon$$

i.e. that *Daemon* is always applicable.

This means that the internal operation *Daemon* can always be invoked by the system, and therefore we have introduced livelock into the specification. This would not be acceptable in an implementation.

The alternative to this would be to leave *Daemon* as $Daemon_0$, i.e., just specify the intended behaviour. However, now it is not a refinement since

$$\mathrm{pre}\,\Xi Ext \land Retrieve \Rightarrow \mathrm{pre}\,Daemon$$

fails. We will return to this point later.

Suppose for the moment that we are given the sectional view specification containing an internal operation $Daemon \mathrel{\widehat=} Daemon_0$, we can now refine this further. In particular we can refine the *Daemon* operation. This operation is partial (as it does not specify what happens if $ins\ i = \langle\,\rangle$ for every i), and using the standard Z refinement rules we can weaken its pre-condition, and refine it to the following:

$$
\begin{array}{|l}
\hline
NDaemon \\
\Delta Section \\
\hline
(\forall i : 1..N-1;\ \exists m : M \bullet ins\ i = \langle\,\rangle \land ins'1 = \langle m \rangle) \lor \\
(\exists i : 1..N-1 \mid \\
\quad ins\ i \neq \langle\,\rangle \bullet \\
\quad ins'i = front(ins\ i) \\
\quad ins'(i+1) = \langle last(ins\ i) \rangle \frown ins(i+1) \\
\quad \forall j : 1..N \mid j \neq i \land j \neq i+1 \bullet ins'j = ins\ j) \\
\hline
\end{array}
$$

This operation includes the same functionality as before, except that in addition the system can invoke it non-deterministically (since it is an internal operation) initially to insert an arbitrary message into the first section. Thus initially there are two possible behaviours of the system: as before the user could invoke *Transmit* to insert a message into the protocol, or now the system could non-deterministically invoke *NDaemon* which corrupts the input stream of the protocol before the user has inserted any messages ($ins'1 = \langle m \rangle$).

The specification which contains the sectional view operations together with this new *NDaemon* in place of *Daemon* is a refinement of the sectional view. Yet clearly implementations which introduce arbitrary amounts of noise into a stream of protocol messages are unacceptable. But in these situations, using standard Z refinement this has been allowed to happen, what has gone wrong?

We have used standard Z refinement here, and at issue is the refinement of internal operations. Internal operations have behaviour which isn't subject to the normal interpretation of operations that are in the user interface, therefore it is not surprising that the standard refinement rules bring about unexpected and undesirable consequences.

Furthermore, the standard refinement rules allow the possibility of livelock or divergence to be *added* when we refine an internal operation. For example, the *Daemon* internal operation in the sectional view could be replaced by a divergent version, *DDaemon*, specified by:

$$
\begin{array}{|l}
\textit{DDaemon} \\
\hline
\Delta \textit{Section} \\
\hline
\textit{ins}' = \textit{ins} \\
\hline
\end{array}
$$

The specification containing this operation as an internal operation is a refinement of the external view. However, the system now contains divergence in that *DDaemon* can be invoked non-deterministically an arbitrary number of times, causing a livelock.

The introduction of livelock is not due to the introduction of an internal operation *Daemon* refining the identity on *Ext*, ΞExt. To see this it is sufficient to note that a divergent version of *NDaemon* given by

$$
\begin{array}{|l}
\textit{DNDaemon} \\
\hline
\Delta \textit{Section} \\
\hline
(\forall i : 1..N-1 \bullet \textit{ins}\ i = \langle\,\rangle \wedge \textit{ins}' = \textit{ins}) \vee \\
(\exists i : 1..N-1 \mid \\
\quad \textit{ins}\ i \neq \langle\,\rangle \bullet \\
\quad \textit{ins}'\, i = \textit{front}(\textit{ins}\ i) \\
\quad \textit{ins}'(i+1) = \langle \textit{last}(\textit{ins}\ i) \rangle \frown \textit{ins}(i+1) \\
\quad \forall j : 1..N \mid j \neq i \wedge j \neq i+1 \bullet \textit{ins}'\, j = \textit{ins}\ j) \\
\hline
\end{array}
$$

is a refinement of *Daemon*, and introduces similar potential livelock at the initial system state.

The weak refinement rules presented below will contain two conditions which are necessary and sufficient to prevent divergence being introduced upon refinement. An alternative approach to these rules which explicitly prevent livelock being introduced is to adopt a *non-catastrophic* interpretation of divergence, this approach is discussed in Section 6.1 below.

3.4. The firing condition interpretation

The firing condition interpretation is a potential solution to the problems encountered when refining internal operations described by Strulo in [Str95]. It has the merit of simplicity, but, as we shall see, perhaps constrains refinement too far. Strulo calls internal operations *active*, and operations in the user interface *passive*. The firing condition interpretation is the idea that the pre-condition of an operation specifies when the operation can happen instead of saying that an operation is undefined, but possible, outside its pre-condition. That is, the pre-condition represents the guard of an operation.

To define refinement, Strulo identifies three regions for an operation (unconstrained, empty and interesting). The three regions of an operation represent:

The unconstrained region: states where the operation is divergent because no constraints are made on the after state;

The empty region: states outside the usual pre-condition but which aren't divergent, and the operation is considered to be impossible in this region; and

The interesting region: the remaining states where some but not all after states are allowed.

The applicability and correctness refinement rules are then re-interpreted for internal operations as:

$$\vdash COp \Rightarrow AOp$$
$$\vdash (\exists\, State' \bullet AOp) \land (\exists\, State' \bullet \neg\, AOp) \Rightarrow (\exists\, State' \bullet COp) \land (\exists\, State' \bullet \neg\, COp)$$

In terms of these interpretations and the regions of definition of an operation, the first condition prevents an operation becoming possible (unconstrained or interesting) where it was impossible (empty), and the second condition ensures that the concrete operation doesn't become impossible (empty) where it was defined and possible (interesting).

For a full discussion the reader should consult [Str95]. It is worth remarking that no data refinement is considered here and that these rules constitute conditions for operation refinement only.

We can apply these ideas to the above example, and in doing so we find that with the firing condition interpretation, *NDaemon* is not a refinement of *Daemon*. This is because it is not true that

$$\vdash NDaemon \Rightarrow Daemon$$

Thus this interpretation successfully stops the pre-condition of an internal operation from being weakened. However, in order to achieve this the rules place a barrier between observable and unobservable operation refinements. In particular, for hybrid specifications (ones involving both internal and observable operations), the refinement rules used depend on the type of operation - standard refinement for observable operations, and the firing condition interpretation for internal operations.

However, the division is not always as simple as that, on occasion we may wish to *introduce* internal operations during a refinement, or we may wish to *remove* internal operations in a refinement. The refinement of the external view to the sectional view is an example of the introduction of internal operations, and we will give an example of their removal shortly.

The consequence of this is that, unfortunately, under the firing condition interpretation we find that the sectional view is not a refinement of the external view of the protocol, because now *Daemon* does not correspond to ΞExt under the firing condition interpretation refinement rules (since we are adding an explicit internal operation when there was no one previously). To overcome this, can we restrict the use of the firing condition interpretation refinement rules to when the abstract operation is internal? The following example illustrates that we cannot.

Consider an abstract specification with an operation AOp in the user interface, and an internal operation IOp. The concrete specification consists of a single operation COp. Both have state space *State* consisting of a $mode : \{0, 1\}$. Initially *mode* is set to 0. The only operations in the specifications are given by:

$$\begin{array}{|l}
\multicolumn{1}{l}{AOp} \\ \hline
\Delta State \\ \hline
mode = 0 \land mode' = 1 \\ \hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{IOp} \\ \hline
\Delta State \\
error! : yes \mid no \\ \hline
mode = 1 \land mode' = 0 \\
error! = yes \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{COp} \\ \hline
\Delta State \\
error! : yes \mid no \\ \hline
mode = mode' = 0 \land error! = yes \\ \hline
\end{array}$$

With these specifications their observable behaviour is identical to an external observer. Therefore it is natural to view the concrete specification as a refinement of the abstract. In the abstract, after invoking AOp an error message will occur (triggered by the internal operation IOp happening, which it eventually always will[§]). Likewise in the concrete specification, after invoking COp an error message will occur. This type of removal of internal events lies at the heart of all treatments of internal operations in process algebras. However, under the firing condition interpretation, the concrete operation is not a refinement of the abstract, because no operation that was possible can become impossible - even if the internal behaviour has moved elsewhere [¶].

Summarising the discussion so far, we have found that the standard notion of refinement in Z is too liberal in the presence of internal operations. Problems have arisen because of the interpretation of internal operations which have allowed undesirable behaviour to be introduced into a refinement, including the possibility of divergence through livelock. By considering the pre-condition of an operation to represent its guard, an alternative approach to refinement is developed in [Str95]. However, this involves a different interpretation of operations, and the refinement of internal behaviour can be too strict as the example above shows. In the next section we will seek an alternative generalization of refinement motivated by the treatment of internal events in process algebras.

4. Weak Refinement

To define weak refinement we will consider the standpoint of an *external* observer who is concerned with the observable operations only. Such an external observer will require that a retrieve relation is still defined between the state spaces of the abstract and concrete specifications and that each observable operation AOp is recast as a concrete operation COp. The refinement relation will ensure that the observable behaviour of the concrete specification is a refinement of the observable behaviour of the abstract specification.

Three of the weak refinement rules have the same form as standard refinement:

§ We are assuming an implicit weak fairness condition here, that if an internal operation is continuously offered it eventually will be taken. This is the standard assumption to make [Led91], and we do not discuss it further in this paper.

¶ The issue of internal operations having output is discussed in Section 5.2.

Initialisation $\forall\, Cstate' \bullet Cinit_w \vdash \exists\, Astate' \bullet Ainit_w \wedge Ret'$

Applicability $\forall\, Astate;\ Cstate \bullet \mathrm{pre}_w\, AOp \wedge Ret \vdash \mathrm{pre}_w\, COp$

Correctness $\forall\, Astate;\ Cstate;\ Cstate' \bullet \mathrm{pre}_w\, AOp \wedge Ret \wedge COp_w \vdash \exists\, Astate' \bullet Ret' \wedge AOp_w$

except that the subscript w denotes a weak counterpart which we will define below and involves sequences of internal operations.

In addition, we introduce two conditions that prevent the introduction of divergence upon refinement, they are:

D1 $Ret \vdash E \in WF$

D2 $\forall\, i \bullet Ret \wedge i \vdash E' < E$

where the quantification in D2 is over all internal operations in the concrete specification, and (WF,<) is a well-founded set and E an expression in the state variables‖.

To motivate our ideas the next subsection reviews the treatment of internal events in process algebras, and we use these ideas in our formulation of weak refinement which will follow.

4.1. Internal events in Process Algebras

Refinement in a process algebra is defined in terms of the transitions a behaviour or process can undergo. We write $P \xrightarrow{a} P'$ if a process (or behaviour) P can perform the action a and then evolve to the process P'. Refinements and equivalences are defined in terms of a systems transitions. Typically, for each relation, two versions are possible - a strong relation which treats all actions identically whether observable or not, and a weak version that makes allowances for internal events and is only concerned with observable transitions.

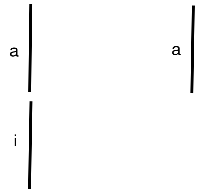
To make allowances for internal actions, consideration is given to what is meant by an observable transition. An observable transition is taken to be any observable action preceded or succeeded by any (finite) number of internal events. Observable transitions are written $P \stackrel{a}{\Longrightarrow} P'$, which means that process P can evolve to process P' by undergoing an unspecified (but finite) number of internal events, followed by the action a, followed by an unspecified number of internal events.

Given a (strong) relation defined in terms of allowable transitions its weak or observable counterpart would replace a transition $P \xrightarrow{a} P'$ by the observable transition: $P \stackrel{a}{\Longrightarrow} P'$.

For example, strong bisimulation relates two behaviours P and Q as equivalent whenever a transition $P_1 \xrightarrow{a} P_2$ in P is matched exactly by a transition $Q_1 \xrightarrow{a} Q_2$ in Q (for a complete definition and full details see, for example, [Mil89]). Weak bisimulation (or observational equivalence), [Mil89], weakens the requirement in strong bisimulation in the sense that two behaviours P and Q are weakly equivalent whenever a transition $P_1 \xrightarrow{a} P_2$ in P is matched by a similar observable transition $Q_1 \stackrel{a}{\Longrightarrow} Q_2$ in Q. An extremely simple example (cf Section

‖ This is essentially the technique of using a variant function to prove termination.

3.4) is the following two behaviours (represented by transition diagrams) which are weak bisimular but not strongly bisimular:



4.2. Formulating weak refinement

Throughout this section we denote the state spaces of the abstract and concrete specifications by *Astate* and *Cstate* respectively. Let *Ret* be the retrieve relation defined between the specifications. *AOp* and *COp* stand for operations on the abstract and concrete state spaces where *COp* implements *AOp*. The initial states are given by schemas *Cinit* and *Ainit*.

Our formulation of weak refinement will be motivated by the approach taken in process algebras. Application of an operation in Z corresponds to a transition in a process algebra, and in weak refinement in place of the application of an operation *Op* we allow a finite number of internal operations before and after the occurrence of the operation. This corresponds to the change from $P \xrightarrow{a} P'$ to $P \xRightarrow{a} P'$ in a process algebra when moving from a strong to observable scenario.

Here we take advantage of the Z schema calculus, and note that $\xRightarrow{Op}$ can be denoted by saying that there exist internal operations $i_1, \ldots, i_k, j_1, \ldots, j_l$ (for some $k, l \geq 0$) such that we can apply the composition $i_1 \fatsemi \ldots \fatsemi i_k \fatsemi Op \fatsemi j_1 \fatsemi \ldots \fatsemi j_l$. In order to avoid such quantifications over sequences of internal operations, we encode "all possible internal evolution" for a specification as a *single* operation I (such that we can write $I \fatsemi Op \fatsemi I$) as follows.

Let *Internals* be the set of all internal operations in the specification; this set can be typed as $\mathbb{P}\, StateOp$ for some $StateOp$. Let $IntSeq == \mathrm{seq}\, Internals$, representing all *finite* sequences of internal operations. The *effect* of such a sequence is obtained using the operator $\circ : IntSeq \rightarrow StateOp$ defined, using distributed schema composition, by

$$\overset{\circ}{\langle\,\rangle} = \Xi State$$
$$\overset{\circ}{ops} = \fatsemi / ops \qquad \text{for } ops \neq \langle\,\rangle$$

"Every possible finite internal evolution" is now described by the schema disjunction of the effects of all possible finite sequences of internal operations, i.e.

$$I = \exists\, x : IntSeq \bullet \overset{\circ}{x}$$

or in other words, two states are related by I iff there exists a series of internal operations x such that the combined effect $\overset{\circ}{x}$ of these operations relates the states.

We distinguish between internal operations in the concrete and abstract specifications by using the subscripts C and A on I. For operations Op abbreviate $\mathrm{pre}(I \fatsemi Op)$ by $\mathrm{pre}_w\, Op$, and $I \fatsemi Op \fatsemi I$ by Op_w if desired. (Note that $\mathrm{pre}\, Op_w = \mathrm{pre}_w\, Op$ since I is total.)

We can now re-formulate each of the three conditions for refinement for a system containing internal operations. We begin with the initialization condition.

Initialization

Without internal operations the relationship required upon initialization is that each possible initial state of the concrete specification must represent a possible initial state of the abstract specification. In the presence of internal operations after an initialization the system might evolve internally to another state. Therefore, "each possible initial state of the concrete specification" now includes all possible evolutions of the initial state under internal operations. Likewise "a possible initial state of the abstract specification" can now include a potential evolution of the initial state due to invocation of internal operations in the system.

To formalise this we require that:

$$\forall\, Cstate' \bullet Cinit \fatsemi I_C \vdash \exists\, Astate' \bullet (Ainit \fatsemi I_A) \land Ret$$

The (hidden) quantification (over all possible evolutions) of the internal operations in $Cinit \fatsemi I_C$ is important. What we wish to ensure is that *every* initial concrete path (including all possible internal operations) can be matched by *some* initial abstract path (possibly involving internal operations). We abbreviate the condition to

$$\forall\, Cstate' \bullet Cinit_w \vdash \exists\, Astate' \bullet Ainit_w \land Ret'$$

Applicability

Applicability must ensure that if an abstract and concrete state are related by the retrieve relation, then the concrete operation should terminate whenever the abstract operation terminated, where termination is usually expressed in terms of satisfaction of the pre-condition of an operation. In the presence of internal operations we must allow for potential invocation of internal operations, and hence we require that: if an abstract and concrete state are related by the retrieve relation, then whenever the abstract operation terminates possibly after any internal evolution then the concrete operation terminates after some internal evolution. This is described by saying there exists internal operations $i_1, \ldots, i_k$ such that $\mathrm{pre}(i_1 \fatsemi \ldots \fatsemi i_k \fatsemi AOp)$ holds.

Applicability can then be expressed as

$$\forall\, Astate;\ Cstate \bullet \mathrm{pre}(I_A \fatsemi AOp) \land Ret \vdash \mathrm{pre}(I_C \fatsemi COp)$$

Using the abbreviation $\mathrm{pre}_w\, AOp$, where we note that we have replaced $\mathrm{pre}\, AOp$ by the condition that AOp is applicable after a number of internal operations, applicability in weak refinement reduces to

$$\forall\, Astate;\ Cstate \bullet \mathrm{pre}_w\, AOp \land Ret \vdash \mathrm{pre}_w\, COp$$

Correctness

For correctness, we require the weak analogy to the following: if an abstract state and a concrete state are related by Ret, and both the abstract and concrete operations are guaranteed to terminate, then every possible state after the concrete operation must be related by Ret' to a possible state after the abstract operation [Spi89]. For the weak version $\mathrm{pre}\, AOp$ is replaced by $\mathrm{pre}_w\, AOp$ and we ask that, every possible state after the concrete operation must be related by Ret' to a possible state after the abstract operation, except that now 'after' means an arbitrary number of internal operations may occur before and after the abstract operation. The condition thus becomes, in full,

$$\forall\, Astate;\ Cstate;\ Cstate' \bullet \mathrm{pre}(I_A \mathbin{;} AOp) \land Ret \land (I_C \mathbin{;} COp \mathbin{;} I_C) \vdash \\ \exists\, Astate' \bullet Ret' \land (I_A \mathbin{;} AOp \mathbin{;} I_A)$$

which we abbreviate to

$$\forall\, Astate;\ Cstate;\ Cstate' \bullet \mathrm{pre}_w\, AOp \land Ret \land COp_w \vdash \exists\, Astate' \bullet Ret' \land AOp_w$$

Again the quantification over every possible finite internal evolution in COp_w is important. We need to ensure that every path involving COp and possible internal operations can be matched by some path involving AOp and (possibly) internal operations. Hence the quantification in COp_w is over all finite sequences of internal operations before and after COp.

Rules for Internal operations

We will also apply the correctness rule to internal operations. For internal operations we do not want applicability to prevent an internal operation becoming impossible where it was previously possible, indeed we want to refine out such internal operations if appropriate. Therefore for an internal operation i (defined on a state space $State$) we define its weak pre-condition (not its pre-condition) by

$$\mathrm{pre}_w\, i = \mathrm{pre}\, \Xi State = State$$

Although this definition of the weak pre-condition for internal operations looks strange, it does not allow us to arbitrarily weaken the pre-condition of an internal operation under weak refinement. The circumstances when we can are governed by what observable operations are present in the abstract specification, and the correctness rules for *observable* operations prevent the arbitrary weakening of pre-conditions of internal operations.

Applicability for internal operations will reduce to checking that the concrete state is implied by the abstract state (modulo the retrieve relation).

The final piece in the jigsaw is the meaning of correctness for internal operations. Recall that we define the weak version of an operation Op by

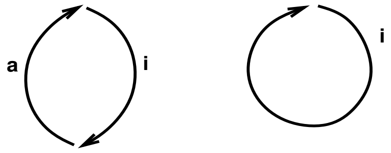
$$Op_w = \begin{cases} I \mathbin{;} Op \mathbin{;} I & \text{for an observable } Op, \\ I & \text{for an internal operation } Op \end{cases}$$

This ensures that we can match up an occurrence of an internal operation in the abstract specification by zero or more internal actions (using I) in the concrete specification.

To prevent divergence being introduced upon refinement we introduce two divergence refinement rules. The criteria these rules embody are based upon those in [But97]. We use a well-founded set WF with a partial order $<$, and a variant which is an expression in the state variables. The variant, E, should always be an element of the set WF, and it should be decreased by each internal operation in the concrete operation. These two conditions can be formulated as:

D1 $Ret \vdash E \in WF$
D2 $\forall i \bullet Ret \land i \vdash E' < E$

where the quantification in D2 is over all internal operations in the concrete specification. Note that although internal operations decrease the variant, there are no constraints on observable operations, which are allowed to increase the variant. This means that an internal operation can be invoked an infinite *number* of times, but not in an infinite sequence. So for example in the following figure with appropriately chosen variant the behaviour on the left satisfies D1 and D2, whereas the behaviour on the right cannot possibly do so.



Summarising the conditions we find that weak refinement requires that

- $\forall Cstate' \bullet Cinit_w \vdash \exists Astate' \bullet Ainit_w \land Ret'$
- $\forall Astate;\ Cstate \bullet \mathrm{pre}_w\ AOp \land Ret \vdash \mathrm{pre}_w\ COp$
- $\forall Astate;\ Cstate;\ Cstate' \bullet \mathrm{pre}_w\ AOp \land Ret \land COp_w \vdash \exists Astate' \bullet Ret' \land AOp_w$

where $\mathrm{pre}_w(Op) = \mathrm{pre}(I \mathbin{⨾} Op)$ and

$$Op_w = \begin{cases} I \mathbin{⨾} Op \mathbin{⨾} I & \text{for an observable } Op, \\ I & \text{for an internal operation } Op \end{cases}$$

with correctness (but not applicability) being applied to the internal operations.

In addition, if WF is a well-founded set and E an expression in the state variables, the following rules prevent the introduction of divergence:

D1 $Ret \vdash E \in WF$
D2 $\forall i \bullet Ret \land i \vdash E' < E$

where the quantification in D2 is over all internal operations in the concrete specification.

In the next section we show how these rules are applied in practice, and we shall see that although the full generality introduces complexity, in practice the overheads are not large.

5. Examples

In this section we illustrate the theory that was developed above to the examples presented at the start of the paper. In the protocol example, the intuitive behaviour we wish to capture is that the sectional view is a refinement of the external view, but that the third specification is not a refinement of the sectional view. We show that this is indeed the case with weak refinement. We then consider internal operations which output to the environment and compare the Z specification of such internal events to the approach taken in process algebras.

5.1. The Signalling Protocol

First we show that the sectional view of the protocol is a weak refinement of the external view. We first prove the initialization is correct, noting that the retrieve relation is total and functional, so that we can use the usual simplification, and we show that:

$$\forall\, Ext';\ Section' \bullet SectionInit_w \land Retrieve \vdash ExtInit_w$$

This reduces to $\forall\, Ext';\ Section' \bullet SectionInit \land Retrieve \vdash ExtInit$, since there are no internal operations in the external specification, and no internal operation is applicable after *SectionInit* in the sectional view. This can be verified as in the verification of the standard refinement in Section 3.2.

To verify applicability, we need to show that

$$\mathrm{pre}_w\ Transmit \land Retrieve \vdash \mathrm{pre}_w\ STransmit$$
$$\mathrm{pre}_w\ Receive \land Retrieve \vdash \mathrm{pre}_w\ SReceive$$

In the case of *Transmit*, this weak applicability requirement reduces to

$$\mathrm{pre}\ Transmit \land Retrieve \vdash \mathrm{pre}(I_S \fatsemi STransmit)$$

since $\mathrm{pre}_w\ Transmit = \mathrm{pre}\ Transmit$. We find this to be true by considering the empty sequence of internal operations in the sectional view. A similar argument holds for the weak applicability requirement for *Receive*. Notice that weak refinement does not require that *Daemon* is always applicable since we only verify correctness of internal operations. Therefore *Daemon* is not forced to be a total operation, and the problem of livelock is solved.

Similarly, to verify correctness, we need to show that

$$\mathrm{pre}\ Transmit \land Retrieve \land STransmit_w \land Retrieve' \vdash Transmit$$
$$\mathrm{pre}\ Receive \land Retrieve \land SReceive_w \land Retrieve' \vdash Receive$$
$$\mathrm{pre}\ \Xi Ext \land Retrieve \land Daemon_w \land Retrieve' \vdash \Xi Ext$$

For the first, we need to check that occurrences of the *Daemon* operation before and after *STransmit* in the concrete specification still leave us in a state that is consistent with that produced by *Transmit* in the abstract. From the refinement demonstrated in Section 3.2 we found that $\mathrm{pre}\ \Xi Ext \land Retrieve \land Daemon \land Retrieve' \Rightarrow \Xi Ext$, it therefore follows that $Retrieve \land Daemon \land Retrieve' \Rightarrow \Xi Ext$, and hence that

$$\begin{aligned}&\mathrm{pre}\ Transmit \land Retrieve \land STransmit_w \land Retrieve' \Rightarrow\\ &\qquad \mathrm{pre}\ Transmit \land Retrieve \land \Xi Ext \fatsemi STransmit \fatsemi \Xi Ext \land Retrieve'\\ &\qquad \vdash Transmit\end{aligned}$$

The second case is similar. For the third this reduces to showing that

$$\forall\, k \bullet Ext \land Retrieve \land Daemon^k \land Retrieve' \vdash \Xi Ext$$

where $Daemon^k$ denotes k sequential compositions of *Daemon*. We can make the deduction

$$Ext \land Retrieve \land Daemon^k \land Retrieve' \Rightarrow Ext \land \Xi Ext \Rightarrow \Xi Ext$$

Finally to show that the sectional view does not introduce divergence in the form of potential livelock of its internal operations we will prove that the divergence criteria are satisfied. To do so we consider the well founded set to be the lexiographical ordering on $\mathbb{N}^N$ (where N is the number of sections in the protocol). The variant will be the expression $\langle \# ins 1, \ldots, \# ins N\rangle$, i.e. a sequence consisting of the number of messages inside each section in the route.

Clearly we have $E \in WF$. Furthermore we have

$$Ret \land Daemon \Rightarrow E' < E$$

since

$$Daemon \Rightarrow \exists\, i : 1..N-1 \bullet (ins'\ i = front(ins\ i) \land \forall\, j < i \bullet ins'\ j = ins\ j)$$

so that $\forall\, j < i \bullet \# ins'\ j = \# ins\ j$ and $\# ins'\ i = (\# ins\ i) - 1$. This ensures that if *Daemon* is applicable then it can only be invoked a finite number of times before it is disabled and an observable operation must be invoked.

Therefore we have shown that the sectional view is indeed a weak refinement of the external view and that no livelock has been introduced upon refinement. Moreover, the additional verification requirements imposed by the generality of weak refinement are not large in this example, being confined to the consideration of one internal operation - *Daemon*.

We shall now show that the third specification is *not* a weak refinement of the sectional view. That is, we are not at liberty to weaken the pre-condition of an internal operation arbitrarily. Consider the initialization rule that (for total functional *Retrieve*):

$$\forall\, Astate;\ Cstate \bullet Cinit_w \land Retrieve \vdash Ainit_w$$

Now in the sectional view it is not possible to apply *Daemon* initially. However, it is possible to apply *NDaemon* initially (where it arbitrarily inserts a new element into the protocol). Thus for the third specification to be a weak refinement of the sectional view we require that

$$SectionInit \fatsemi NDaemon \vdash SectionInit$$

This is clearly not true, since

$$SectionInit \fatsemi NDaemon \Rightarrow ins'\ 1 \neq \langle\rangle$$

that is, *ins* is no longer empty.

In addition to the initialization requirement failing in this example, the requirement that

$$\mathrm{pre}_w\ STransmit \land Retrieve \land STransmit_w \land Retrieve' \vdash STransmit_w$$

is also violated for similar reasons as the initial condition fails.

5.2. Internal operations with output

In the second example, presented in Section 3.4, in order to show that the concrete specification is a weak refinement of the abstract specification, we would need to prove that for some retrieve relation *Ret*:

$$\forall\, State \bullet \mathrm{pre}_w\ AOp \land Ret \vdash \mathrm{pre}_w\ COp$$
$$\forall\, State \bullet \mathrm{pre}_w\ AOp \land Ret \land COp_w \vdash \exists\, State' \bullet Ret' \land AOp_w$$

The retrieve relation we will use will link the states for which $mode = 0$, since the state $mode = 1$ was used purely as an intermediate state for the purposes of specifying the temporal ordering of the operations. Hence the retrieve relation will be specified by

$$\begin{array}{|l}\hline \textit{Ret} \\ \textit{State} \\ \hline \textit{mode} = 0 \\ \hline \end{array}$$

With this retrieve relation we will in fact show that the concrete operation COp implements both abstract operations AOp and IOp. Since the concrete specification does not have any internal operations we just need to show that:

$$\begin{array}{l} \mathrm{pre}_w\ AOp \wedge Ret \vdash \mathrm{pre}\ COp \\ \mathrm{pre}_w\ AOp \wedge Ret \wedge COp \wedge Ret' \vdash AOp_w \\ \mathrm{pre}_w\ IOp \wedge Ret \vdash \mathrm{pre}\ COp \\ \mathrm{pre}_w\ IOp \wedge Ret \wedge COp \wedge Ret' \vdash IOp_w \end{array}$$

We can calculate the pre-conditions needed. Note that in the case of $\mathrm{pre}_w\ AOp$ this includes states from which the system can perform an internal operation and then invoke AOp, which then terminates successfully.

$$\begin{array}{|l}\hline \mathrm{pre}_w\ AOp \\ \textit{State} \\ \hline \textit{mode} = 0 \vee \textit{mode} = 1 \\ \hline \end{array} \qquad \begin{array}{|l}\hline \mathrm{pre}\ COp \\ \textit{State} \\ \hline \textit{mode} = 0 \\ \hline \end{array}$$

The applicability and correctness for the refinement of AOp as COp are then easily verified. Consideration of the internal operation amounts to showing that (because of the way the pre-condition of an internal operation is defined)

$$\begin{array}{l} Ret \vdash \mathrm{pre}\ COp \\ Ret \wedge COp \wedge Ret' \vdash \exists\, k \bullet IOp^k \end{array}$$

and the latter holds for $k = 0$.

Therefore the concrete specification is indeed a weak refinement of the abstract (because there are no internal operations in the concrete system we do not need to check for divergence). This illustrates an interesting aspect of specifying internal operations in Z - they can output data (in fact some interpretations of unobservableness in Z outlaw this possibility e.g. [CR92], but generally this is the case [Str95, WJ94]). This is in contrast to a process algebra where typically internal actions can have no data attributes.

Consider, for example, full LOTOS [BB88], where the internal action is written i. Internal actions in LOTOS can arise as a result of direct specification or as a result of hiding observable actions. In the first case, it is syntactically illegal to associate a data attribute with an internal action, e.g. the behaviour

$$i!7;\ B$$

is not well-formed. Here action prefix is represented by **;** and a value declaration on an action is given by a **!**, and B represents the subsequent behaviour. In the second case, upon hiding an observable action with data, the data is hidden as well as the action. So, for example, in the behaviour

$$\mathrm{hide}\ g\ \mathrm{in}\ (g!5;\ \textit{stop})$$

the transition i can be performed, but no data is associated with the occurrence of the internal action i. That is the only transition this behaviour can perform is the following.

$$\text{hide } g \text{ in } (g!5;\ \mathit{stop}) \xrightarrow{i} \text{hide } g \text{ in } \mathit{stop}$$

However, it is desirable to be able to specify an internal event which does have data associated with it. Indeed [Str95] contains an example of such an operation - an alarm notification in a managed object. This is a typical example of the kind of application where it is necessary to be able to specify an atomic internal operation which has output associated with it. Used in this style Z offers a different model to LOTOS in terms of internal events it can specify.

Whether or not such an internal event is unobservable is debatable, and perhaps such events mark the difference between active systems as opposed to reactive systems - the latter often modelled using a process algebra. In an active system events can be under the control of the system but not the environment (e.g. an alarm operation), such events are internal but can have observable effects (such as an alarm notification). This differs from the notion of internal in a process algebra, which equates internal with no observable transition or effect, including output. In such an interpretation the operation *IOp* defined above would not be internal as we can observe its occurrence via its output, and the term *active* used in [Str95] could be used instead. However, the theory of weak refinement developed here is equally applicable to such a class of events.

6. Removing internal operations

In this section we will consider to what extent it is true that we can dispense with internal operations, both in terms of their specification and in terms of refinements of specifications containing them. To do so we begin with a discussion of labelled transition systems (LTS) which provide a suitable model to discuss the role of internal operations. We will use labelled transition systems to answer the question

For any specification containing internal operations, is there an equivalent specification without internal operations?

and to do so we will need to consider a suitable definition of equivalence. We will argue that testing equivalence provides a suitable yardstick by which to compare specifications. We will then show that for any specification containing internal operations, we can find a testing equivalent specification not containing any internal operations.

Having answered the original question in the affirmative, we can then prove that weak refinement is correct in the sense that: if specification S_2 is a weak refinement of specification S_1, then there exists equivalent specifications to S_1 and S_2, T_1, T_2 respectively, not containing internal operations such that T_2 is a standard Z refinement of specification T_1. The consequences of this are that we can dispense with internal operations if we choose, but if we use them then their weak refinement is still correct.

So far this discussion will have taken place in the context of divergence free specifications. We will conclude this section with a discussion on the removal, and interpretation, of divergence due to livelock.

A labelled transition system [BSS86] is a 4-tuple $LTS = \langle S, L, \longrightarrow, s_0 \rangle$, where S is a set of states, L a set of labels, $\longrightarrow \in S \times L \times S$ being a transition relation and $s_0 \in S$ the initial state of the system. As usual we write $s_1 \xrightarrow{a} s_2$ whenever $(s_1, a, s_2) \in \longrightarrow$. We will need the following (standard) definitions:

$P \xrightarrow{a_1 \dots a_n} P'$ means that there exist $P_1, \dots, P_{n-1} \in S$ such that $P \xrightarrow{a_1} P_1 \xrightarrow{a_2} P_2 \dots P_{n-1} \xrightarrow{a_n} P'$.

$P \overset{\sigma}{\Longrightarrow} P'$ if $\sigma = a_1 \dots a_n$ means $\exists\, k_0, \dots, k_n \in \mathbb{N}$ such that $P \xrightarrow{i^{k_0} a_1 i^{k_1} a_2 \dots a_n i^{k_n}} P'$

$P \overset{\sigma}{\Longrightarrow}$ means $\exists\, P'$ such that $P \overset{\sigma}{\Longrightarrow} P'$

$P \overset{\sigma}{\not\Longrightarrow}$ means that $\neg(P \overset{\sigma}{\Longrightarrow})$

P *after* $\sigma = \{P' \mid P \overset{\sigma}{\Longrightarrow} P'\}$ is the set of all states reachable from P after σ.

$Ref(P, \sigma) = \{X \mid \exists\, P' \in P \textit{ after } \sigma \bullet P' \overset{a}{\not\Longrightarrow}, \forall\, a \in X\}$ is the refusal set of P after the trace σ.

$Tr(P) = \{\sigma \mid P \overset{\sigma}{\Longrightarrow}\}$ is the trace set of P.

We also call P stable if P has no initial internal transition. In this discussion we can limit ourselves to stable systems since any Z specification can be considered stable due to the presence of the (observable) initialisation schema. We can now define reduction and testing equivalence for labelled transition systems in a standard fashion [BSS86] (this is the formulation used in the LOTOS community, there are alternative, but equivalent, formulations in CSP).

Definition 1.
Let $P_1 = \langle S_1, L_1, \longrightarrow_1, s_0 \rangle$ and $P_2 = \langle S_2, L_2, \longrightarrow_2, t_0 \rangle$ be labelled transition systems. Then P_1 **red** P_2 iff (i) $Tr(P_1) \subseteq Tr(P_2)$, and (ii) $\forall\, \sigma \in Tr(P_1), Ref(P_1, \sigma) \subseteq Ref(P_2, \sigma)$.

Reduction induces an equivalence called testing equivalence defined as follows: P_1 **te** P_2 iff (i) $Tr(P_1) = Tr(P_2)$, and (ii) $\forall\, \sigma \in Tr(P_1), Ref(P_1, \sigma) = Ref(P_2, \sigma)$.

It has been argued that testing equivalence is a natural and correct notion of equivalence between systems [BB88]. Weak bisimulation is known to respect all the distinctions which could *reasonably* be made by an external observer. However, it is often considered too fine and makes distinctions which couldn't *really* be made by an observer [Led91, BSS86]. Testing equivalence on the other hand makes precisely those distinctions which can be observed by testing the systems under consideration. If we consider labelled transition systems to represent the behaviour of a system or specification, we can use testing equivalence as a suitable notion of equivalence, two systems are equivalent if their LTSs are testing equivalent.

The context we are interested in here is how to answer the following question: given a Z specification with internal operations explicitly specified, can we dispense with such operations by adding their non-determinism to the observable operations present? If we can answer yes to this question (as is claimed in [WD96]), then we know that internal operations do not increase the expressive power of the language. We can then even verify that weak refinement is correct by showing that weak refinement of a specification with internal operations implies the normal Z refinement if the internal operations are absorbed into the observable ones.

We first consider divergence free specifications, i.e. specifications without divergence due to livelock of internal operations. To show that for divergence free Z specifications we can dispense with internal operations we will derive a transformation which will remove internal operations to create a Z specification which is testing equivalent to the original. We will first describe the transformation in terms of labelled transition systems and prove that testing equivalence is preserved, we will then give the transformation for Z specifications. This makes the implicit assumption that we can represent Z specifications as labelled transition systems in the obvious manner, the standard way to do this is given in [Smi95, CW92] for example.

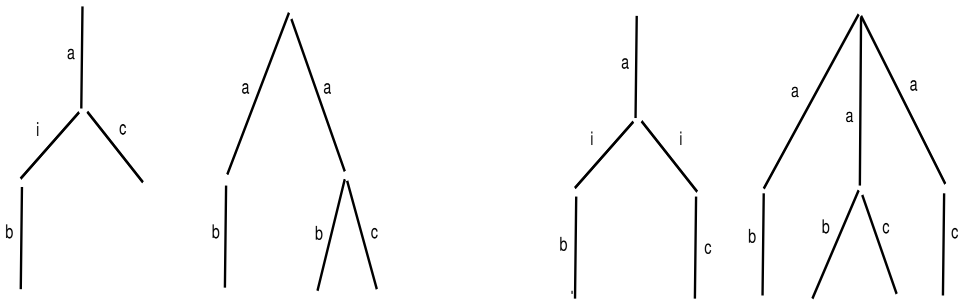
We use testing equivalence as our benchmark for equivalence of specifications as opposed to the equivalence induced by weak refinement because we wish to validate weak refinement against the removal of internal operations. If we had only shown that the transformed specification was weak refinement equivalent to the original, we could not then show that the weak refinement relation was correct. By using testing equivalence we can validate weak refinement.

Given a LTS $P_1 = \langle S, L, \longrightarrow_1, s_0 \rangle$ we derive another labelled transition system $P_2 = \langle S, L, \longrightarrow_2, s_0 \rangle$ which does not contain any internal transitions. The transformation is defined by the following rules:

$$s_1 \xrightarrow{a}_2 s_2 \text{ iff } s_1 \xRightarrow{a}_1 s_2$$

for all observable actions $a \in L$. Note that we are interested in stable labelled transition systems (ones with no initial internal action), as all Z specifications have an initialisation schema which is considered observable.

As an example, we find the above definition produces the following transformations, where in each example the original behaviour is given on the left with the transformed behaviour on the right. Note that the purpose is to generate an equivalent LTS, but it will not necessary be the minimal such system.



Example 1

Example 2

Notice that $P \xRightarrow{\sigma}_1 P'$ if and only $P \xRightarrow{\sigma}_2 P'$. This implies that the traces of the two systems are the same, i.e. $Tr(P_1) = Tr(P_2)$, and furthermore the refusals are identical, that is for all traces σ, $Ref(P_1, \sigma) = Ref(P_2, \sigma)$. Therefore we have proved:

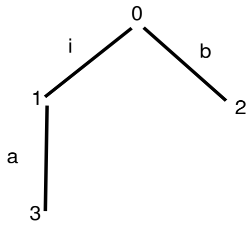
Theorem 1. Every labelled transition system has a transformation to a testing equivalent labelled transition system which contains no internal transitions.

In the context of a Z specification the transformation to remove internal operations consists of redefining each observable operation AOp by an operation AOp_S. That is AOp_S is defined as follows

$$AOp_S \mathrel{\widehat=} I_A \fatsemi AOp \fatsemi I_A$$

Note that this definition is equivalent to taking the disjunction of all combinations of internal operations before and after AOp, i.e. $AOp_S \mathrel{\widehat=} AOp \lor i \fatsemi AOp \lor AOp \fatsemi i \lor i \fatsemi AOp \fatsemi i \lor i \fatsemi i \fatsemi AOp \lor \ldots$. Observe that $\mathrm{pre}\, AOp_S = \mathrm{pre}_w\, AOp$. The transformed Z specification will have an identical number of observable operations, but with the internal operations simply removed. Note that we consider the initialisation schema $\textit{Init}$ as an observable operation, and thus this too absorbs internal operations under the transformation if applicable (i.e. $\textit{Init}_S \mathrel{\widehat=} \textit{Init} \lor \textit{Init} \fatsemi i \lor \textit{Init} \fatsemi i \fatsemi i \lor \ldots$).

For example, consider the behaviour described by the following transition diagram, where a and b are observable events, and i represents an internal operation:



As a Z specification we give this diagram its obvious interpretation as the specification:

$$\begin{array}{|l}\hline \textit{State} \\ \textit{state} : \{0,1,2,3\} \\ \hline\end{array} \qquad \begin{array}{|l}\hline \textit{Init} \\ \Delta \textit{State} \\ \hline \textit{state}' = 0 \\ \hline\end{array}$$

$$\begin{array}{|l}\hline a \\ \Delta \textit{State} \\ \hline \textit{state} = 1 \land \textit{state}' = 3 \\ \hline\end{array} \qquad \begin{array}{|l}\hline b \\ \Delta \textit{State} \\ \hline \textit{state} = 0 \land \textit{state}' = 2 \\ \hline\end{array} \qquad \begin{array}{|l}\hline i \\ \Delta \textit{State} \\ \hline \textit{state} = 0 \land \textit{state}' = 1 \\ \hline\end{array}$$

Then the equivalent specification without internal operations is given by:

$$\begin{array}{|l}\hline \textit{State} \\ \textit{state} : \{0,1,2,3\} \\ \hline\end{array} \qquad \begin{array}{|l}\hline \textit{Init} \\ \Delta \textit{State} \\ \hline \textit{state}' = 0 \lor \textit{state}' = 1 \\ \hline\end{array}$$

$$\begin{array}{|l}\hline a \\ \Delta \textit{State} \\ \hline (\textit{state} = 1 \land \textit{state}' = 3) \lor \\ (\textit{state} = 0 \land \textit{state}' = 3) \\ \hline\end{array} \qquad \begin{array}{|l}\hline b \\ \Delta \textit{State} \\ \hline \textit{state} = 0 \land \textit{state}' = 2 \\ \hline\end{array}$$

With this transformation in place we know we can, if necessary, dispense with internal operations in divergence free specifications. We are now in a position to prove that weak refinement is correct with respect to standard Z refinement, which we do now.

Theorem 2. Let S_1 and S_2 be Z specifications possibly containing internal operations. Let S_2 be a weak refinement of S_1. Then there exists equivalent specifications to S_1 and S_2, denoted T_1, T_2 respectively, not containing internal operations such that T_2 is a standard Z refinement of the specification T_1.

Proof

We assume that there is one internal operation called i in the specifications. The proof generalises easily to an arbitrary number of internal operations.
Because S_2 is a weak refinement of S_1 we know that if the operation COp in S_2 refines the operation AOp in S_1, then the following hold:

- $\forall\, Cstate' \bullet Cinit_w \vdash \exists\, Astate' \bullet Ainit_w \wedge Ret'$
- $\forall\, Astate;\ Cstate \bullet \mathrm{pre}_w\, AOp \wedge Ret \vdash \mathrm{pre}_w\, COp$
- $\forall\, Astate;\ Cstate;\ Cstate' \bullet \mathrm{pre}_w\, AOp \wedge Ret \wedge COp_w \vdash \exists\, Astate' \bullet Ret' \wedge AOp_w$

From the above we know there exist equivalent specifications without internal operations. For each operation Op, let Op_S denote the transformed operation given by the scheme above. We will prove the transformed specifications are refinements, i.e., we will show that

- $\forall\, Cstate' \bullet Cinit_S \vdash \exists\, Astate' \bullet Ainit_S \wedge Ret'$
- $\forall\, Astate;\ Cstate \bullet \mathrm{pre}\, AOp_S \wedge Ret \vdash \mathrm{pre}\, COp_S$
- $\forall\, Astate;\ Cstate;\ Cstate' \bullet \mathrm{pre}\, AOp_S \wedge Ret \wedge COp_S \vdash \exists\, Astate' \bullet Ret' \wedge AOp_S$

Initialization

We can make the following deduction

$$\begin{aligned} Cinit_S &\mathrel{\widehat=} Cinit \fatsemi I_C \\ &\Rightarrow (Ainit \fatsemi I_A) \wedge Ret' \\ &\Rightarrow Ainit_S \wedge Ret' \end{aligned}$$

Applicability

We can make a similar deduction as follows:

$$\begin{aligned} \mathrm{pre}\, AOp_S \wedge Ret &= \mathrm{pre}(I_A \fatsemi AOp \fatsemi I_A) \wedge Ret \\ &= \mathrm{pre}(I_A \fatsemi AOp) \wedge Ret \\ &\Rightarrow \mathrm{pre}(I_C \fatsemi COp) \\ &= \mathrm{pre}\, COp_S \end{aligned}$$

Correctness

Finally, in a similar manner:

$$\begin{aligned} \mathrm{pre}\, AOp_S \wedge Ret \wedge COp_S &\Rightarrow \mathrm{pre}_w\, AOp \wedge Ret \wedge (I_C \fatsemi COp \fatsemi I_C) \\ &\Rightarrow Ret' \wedge AOp_w \\ &\Rightarrow Ret' \wedge AOp_S \end{aligned}$$

This concludes the proof that weak refinement is correct.

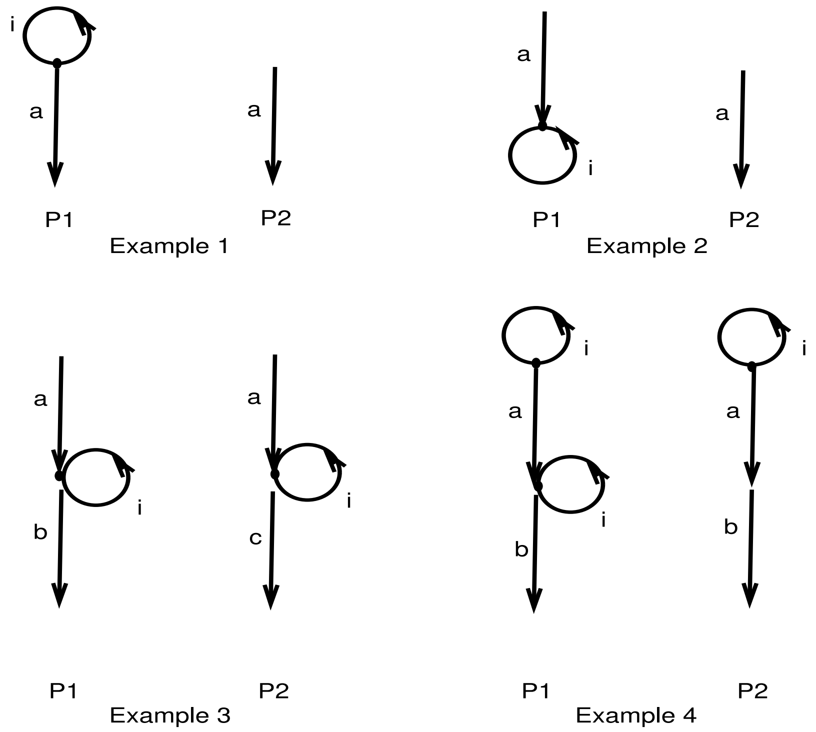
The next subsection considers to what extent these results can carry over to specifications that contain divergence in the form of livelock.

6.1. Divergence

Section 3.3 showed that the standard Z refinement rules could allow *divergence* to be introduced into a Z specification upon refinement. By divergence here we mean a state where an infinite number of internal operations can be invoked, thus causing the system to potentially livelock where it keeps on performing internal and non-visible computations. How best should we treat this type of divergence in Z? One possibility is to use the two refinement rules D1 and D2, which guarantee that if the abstract specification is divergence free, then so will the refinement. However, we would also like to consider whether a divergent specification could be considered equivalent to a specification without internal operations, i.e., whether we really can dispense with internal operations in all circumstances. To answer this we need to consider differing interpretations of divergence.

In a labelled transition system or process algebra there are two standard interpretations of divergence: a catastrophic or non-catastrophic view. The former is based upon the idea that a process diverges after the trace σ if any of its subtraces diverge [BHA84, dNH84] (i.e. $\exists\,\sigma' \leq \sigma$ such that the process diverges after σ'). The alternative non-catastrophic view says that a system P diverges after σ iff there is a state reachable from P by σ such that in that state it is possible to engage in an infinite sequence of internal events [Led91]. These differing interpretations are then reflected in how different equivalences treat divergence.

For example, testing equivalence adopts the non-catastrophic view of divergence, so that it ignores divergence or treats it in a fair manner [Led91]. On the other hand the equivalence induced by *must testing* [Hen88] (denoted $\equiv_{must}$) adopts the catastrophic view of divergence. This equivalence coincides with the failures equivalence of CSP [Hoa85], and therefore CSP is said to take a catastrophic view of divergence, whereas LOTOS with its testing equivalence is said to possess a non-catastrophic view of divergence. For example, consider the following pairs of systems:



We find that in examples 1 and 2, P_1 **te** P_2 ($P1$ and $P2$ have the same traces and refusals) but P_1 and P_2 are not must-equivalent (because in both cases P_1 diverges whereas P_2 does not). However, in example 3, P_1 and P_2 are not testing equivalent (they have different traces), yet they are must-equivalent (the traces only differ after a point of divergence). Finally, example 4 exhibits two systems which are both testing and must-equivalent (they have the same traces and refusals and both diverge initially).

Adopting a non-catastrophic view of divergence allows one to remove internal operations from a Z specification using the same procedure as defined in the previous section. The transformation defined above will remove internal operations from a divergent specification and replace it with a testing equivalent specification containing no internal operations within it.

If one wanted to adopt a catastrophic view of divergence it is more problematic as to whether one can find an equivalent specification without internal operations in it. This depends on whether livelock divergence is considered to be a potentially different kind of divergence than that of a Z operation invoked outside its precondition. Under a catastrophic view, in order to find an equivalent specification without internal operations contained within it, we have to equate the two types of divergence. For example, in example 3 above, to find a specification which is equivalent to the behaviour P_1, we would have to diverge at every trace after state s_1, therefore the best approximation to this would be the specification:

State

$state : \{s_0, s_1\}$

Init

$\Delta State$

$state' = s_0$

a

$\Delta State$

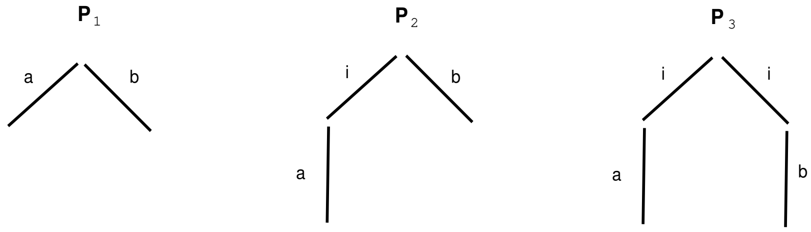
$state = s_0 \wedge state' = s_1$

This specification can perform the operation a initially. However, subsequently it is in state s_1, which is outside the precondition of the operation a. Therefore any subsequent invocation of a will be divergent. The subtle intuitive difference between this specification and P_1 is that in the former it is the invocation of an operation which causes the system to diverge, whereas in P_1 the livelock is invoked by the system itself. So in terms of removal of internal operations it would seem therefore more natural to adopt a non-catastrophic view of divergence in the context of Z specifications.

7. Discussion

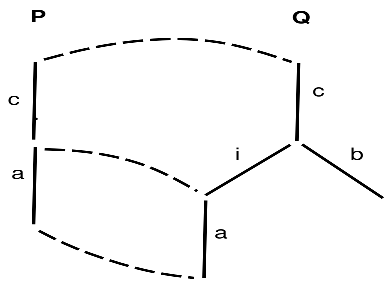
An important aspect of refinement, in both the sequential and concurrent worlds, is the ability to strengthen an implementation by reducing the non-determinism in the abstract specification. Indeed this is a property of standard Z refinement in the absence of internal operations. Adding internal operations in a specification has introduced an additional form of non-determinism into the language. We shall see that weak-refinement allows us to reduce this type of non-determinism by removing internal operations.

Consider the behaviours described by the following transition diagrams, where a and b are observable events, and i represents an internal operation (we have omitted the transition formed by the initialisation schema):



These specifications are not equivalent in any sense, for example in a process algebraic setting none of them are weak bisimulation equivalent. However, we would like a refinement to remove the non-determinism which is present in terms of the internal events, and for P_1 to refine P_2 which in turn refines P_3. Indeed, seen as labelled transition systems or processes they are related in the sense that, for example, P_1 **red** P_2 **red** P_3, where **red** is the reduction relation defined above. Weak refinement, which we denote $\sqsubseteq_w$, also exhibits this property, that is $P_3 \sqsubseteq_w P_2 \sqsubseteq_w P_1$, but $P_1 \not\sqsubseteq_w P_2 \not\sqsubseteq_w P_3$. In terms of Z specifications we are giving these diagrams their obvious interpretation as described in Section 6.

A slightly more complex example is given by the two behaviours defined by the following, where again the event i is internal and all others are observable.

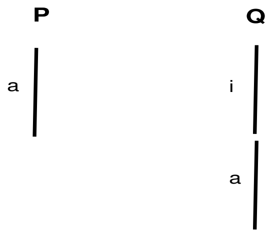


Interpreted as Z specifications we find that P is a weak refinement of Q. This example is interesting because by resolving the non-determinism, the implementation never offers the operation b. The retrieve relation which shows this is a weak refinement is given by the dotted lines in the above diagram. Because pre $b \wedge Ret$ has a predicate which is false, b can be implemented by any operation in the concrete specification (e.g. $\Xi State$ will do).

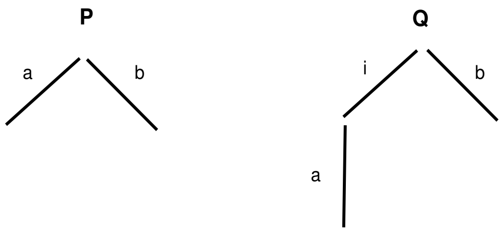
Notice that, as one would hope, Q is not a weak refinement of P, because we have to quantify over *all* paths of internal operations in the concrete specification in the correctness criteria for weak refinement. The corresponding relations between labelled transition systems also hold, i.e. P **red** Q but $\neg\, Q$ **red** P.

One desirable property that standard Z refinement possesses is that it is a congruence. That is, if specification S is refined by S', then in any context $C[.]$, $C[S']$ refines $C[S]$. A consequence of this is that operations can be refined individually and the whole specification is then a refinement of the original.

However, weak refinement is not a congruence, due to the presence of internal operations. To see this consider the two specifications given by the following behaviours:



Then under weak refinement these are equivalent, i.e. $P \sqsubseteq_w Q$ and $Q \sqsubseteq_w P$. However, if we add just one further operation to each specification which is applicable at the initial state, i.e. we specify the behaviour



then, as we observed earlier, Q is not a weak refinement of P. So congruence is lost with weak refinement. Incidentally, this counter-example is the same example that shows weak bisimulation is not a congruence in a process algebra, so the result here is not surprising and the ability to find observational relations which *are* congruences can be non-trivial.

Although weak refinement is not a congruence, it does possess useful properties when used in unification and consistency checking. Unification is a method used to combine partial specifications, and the unification of two specifications is their least common refinement [BBDS97] (least in the sense that any other common refinement is a refinement of the unification). Partial specifications arise in many contexts [FS96], and one such context is their use as *viewpoints* in distributed systems, and in particular their use within the Open Distributed Processing (ODP) standardization initiative [ITU95]. ODP is a joint standardisation activity of the ISO and ITU. A reference model has been defined which describes an architecture for building open distributed systems. Central to this architecture is a viewpoints model. This enables distributed systems to be described from as a number of different partial specifications, each representing a different perspective.

ODP is typical of applications where it is useful to use Z for the specification of distributed systems, i.e., one where we might wish to use a language that will support data refinement of specifications which involve the complex representation of state and use explicit internal operations in the description.

The use of a number of viewpoints to represent multiple aspects of one system under construction means that we need to be able to check the viewpoints for consistency. One consistency checking method is to construct their unification and to check it for contradictions. [BDBS96] describes how this may be achieved if the viewpoints are specified in Z. The unification of two Z viewpoints is constructed in two phases. In the first phase ("state unification"), a unified state space for the two viewpoints has to be constructed. The viewpoint operations are then adapted to operate on this unified state. At this stage we have to check that a condition called *state consistency* is satisfied. In the second phase, called *operation unification*, each pair of adapted operations from the viewpoints which are partial descriptions of the same operation have to be combined into a single operation on the unified state. This also involves a consistency condition (*operation consistency*) which ensures that the unified operation is a refinement of the viewpoint operations.

What is the correct unification strategy if the viewpoints contain internal operations? In the context of ODP this is almost certain to happen, since the viewpoints occur at different levels of abstraction, and operations in one viewpoint may be hidden in another. Do we have to transform the viewpoints to ones not containing internal operations before we apply unification? Fortunately we do not, since it can be shown that the least common weak refinement is equivalent to the least common refinement of the viewpoints without internal operations. That is, if we use the transformation defined earlier that produced testing equivalent specifications without internal operations, and take the least common standard refinement for the unification, this unification will be (testing) equivalent to the least common weak refinement of the original viewpoints. The consequence of this is that we can unify using weak refinement and we do not have to remove internal operations first - a transformation that can be very complex.

The use of viewpoints, or partial specifications, in a number of application areas has led to proposals (see for example [Ben89, MD98, Fis97, Smi97]) to combine state-based methods with process algebras in order that the strengths of a particular method can be applied in an appropriate way. It would be interesting to compare refinement in these methodologies with the ideas of weak refinement discussed in this paper.

8. Related Work

In this section we discuss related approaches to the issue of refinement of state-based specifications containing internal operations. A preliminary version of this paper appears in [DBBS97]. Other work in this area includes [Str95], [But97] and [Eva97]. The work of Strulo, [Str95], was discussed in Section 3.4, and we consider here the proposals of Butler [But97] and Evans [Eva97].

In [But97], Butler considers the design of distributed systems using the B abstract machine notation [Abr96]. His approach is based on the action system formalism, and he considers refinement of abstract machines which contain internal actions. Although placed in a different formalism, the refinement rules in [But97] can be seen to be a restricted version of the weak refinement rules presented here. Butler first considers refinement of an abstract system M to a concrete system N where neither contains any internal actions. Refinement in this context is defined by the following rules:

1. $M.init \sqsubseteq N.init$
2. $M.a \sqsubseteq N.a$ for each (observable) action a
3. $AI \wedge gd(M.a) \Rightarrow gd(N.a)$ for each (observable) action a

where AI is the retrieve relation, $\sqsubseteq$ denotes action refinement in B, $M.a$ represents the action a in system M, and $gd(M.a)$ is the guard of the action a in system M. Informally the first two conditions ensure that each action of N is refinement of its counterpart in M. The third condition ensures that N may only refuse an action when M may refuse it.

Butler then introduces internal actions in an abstract machine as follows, [But97]. "Internal actions are not visible to the environment of a machine. Any number of executions of an internal action may occur in between each execution of a visible action. If the action system reaches a state where internal actions can be executed infinitely, then the action system diverges. Internal actions do not have input or output parameters, and are specified explicitly in a machine." To extend refinement to a concrete system that may contain internal actions, we let $\beta(N)$ denote the set of internal actions in a system N. The extended refinement rules are then given by:

1. $M.init \sqsubseteq N.init$
2. $M.a \sqsubseteq N.a$ for each (observable) action a
3. $skip \sqsubseteq N.h$ for each internal action $h \in \beta(N)$
4. $AI \Rightarrow E \in WF$
5. $AI \wedge E = e \Rightarrow [N.h](E < e)$
6. $AI \wedge gd(M.a) \Rightarrow gd(N.a) \vee (\exists h \in \beta(N) \bullet gd(N.h))$ for each (observable) action a

The divergence conditions (4 and 5) are identical to the ones we have used in our formulation of weak refinement (as is the notation), and we do not discuss them further.

The principal restriction made by Butler (and difference to our work) is to consider only internal actions in the concrete system and for none to occur in the system under refinement. He therefore does not have a mechanism to refine systems containing internal actions. Such a restriction simplifies the refinement rules for internal actions considerably. For example, we find that rule 3: $skip \sqsubseteq N.h$, can be deduced from the weak refinement applicability rule applied to internal operations, since in the Z setting *skip* corresponds to $\Xi State$.

Furthermore, because of the third condition, together with the divergence conditions (4 and 5), the final condition $(AI \wedge gd(M.a) \Rightarrow gd(N.a) \vee (\exists h \in \beta(N) \bullet gd(N.h)))$ represents the same requirements as the weak refinement applicability rule applied to observable operations. This is because conditions 4 and 5 prevent infinite execution of internal actions, and $skip \sqsubseteq N.h$ ensures that execution of an internal action won't effect the abstract state, so that $gd(N.a) \vee (\exists h \in \beta(N) \bullet gd(N.h))$ implies that potentially a finite number of internal actions can occur and then $N.a$ will be enabled. This represents the same criteria as applicability in weak refinement.

However, the initialisation condition (1) and the correctness condition (2) here are more restrictive than their weak refinement counterparts. For example, the B machine initialisation condition does not allow any internal evolution of the concrete system unlike initialisation in weak refinement. Correctness is similarly restrictive.

Evans, in [Eva97], makes a proposal for the refinement of Z specifications in the presence of internal operations. Evans is principally concerned with the specification of safety and liveness properties, and discusses refinement in that context. Even without considering internal operations he uses a reformulation of standard Z refinement which he claims will ensure that safety and liveness properties are preserved under refinement. This reformulation replaces the normal correctness criteria with the following:

$$NextState_C \wedge \Delta Abs \vdash NextState_A$$

where Abs is the retrieve relation and $NextState_C$ ($NextState_A$) is the disjunction of all the operations in the concrete (abstract) specification. For example, in the external view of the protocol discussed above, $NextState_A$ would be $Receive \vee Transmit$.

Evans then considers refinement in the presence of internal operations, and his definition uses four conditions, the standard initialisation condition together with

1. $NextState_C \wedge \Delta Abs \vdash NextState_A$
2. $NextState_C \wedge \neg\, COp \wedge \Delta Abs \vdash \text{pre}\, AOp \Rightarrow \text{pre}\, AOp'$
3. $\text{pre}\, AOp \wedge Abs \leadsto \text{pre}\, COp$

where $\leadsto$ is a formulization of the leads-to property, see [Eva97] for details. $NextState_C$ now includes all the internal operations, for example, in the sectional view of the protocol, $NextState_C$ will be $SReceive \vee STransmit \vee Daemon$.

Unfortunately it is unclear whether internal operations are allowed in the abstract specification or just the concrete. It is also not clear as to whether the final refinement rule of Evans should apply to just the observable operations (as one would expect). Assuming that we apply the final refinement rule to just the observable operations, then this rule can be seen to be a weak applicability rule, assuming the concrete specification doesn't contain divergence (divergence is not discussed in [Eva97]).

The motivation Evans gives for the second condition is its use as a liveness condition to ensure that whenever an abstract operation is enabled, it will remain enabled at least until the corresponding concrete operation occurs. Because his conditions are motivated by liveness and safety issues, his first two conditions are orthogonal to the weak refinement conditions whose motivation was different. Again, like in the work of Butler but unlike our weak refinement, the initialisation condition of Evans does not allow any unobservable evolution of the initial states of the system.

9. Conclusions

The motivation for the work described in this paper arose out of our interest in the use of Z for the specification of distributed systems, and in particular its use within the Open Distributed Processing standardization initiative. A reference model for the standard has been defined which describes an architecture for building open distributed systems. Central to this architecture is a viewpoints model. This enables distributed systems to be described from a number of different perspectives. There are five viewpoints: enterprise, information, computational, engineering and technology. Z and LOTOS are strong candidates for use in some of the ODP viewpoints, for example Z in the information viewpoint and LOTOS in the computational and engineering viewpoints. The use of different viewpoints specified in different languages means we have to have mechanisms to check for the *consistency* of specifications. One aspect of our work has been the development of means to check for the consistency of two Z specifications, and a means to translate LOTOS specifications into Z [DBBS96a].

Requirements and specifications of an ODP system can be made from any of the viewpoints, and these viewpoint specifications will typically be made at different levels of abstraction. It is important therefore that techniques, including refinement, are developed to cope with such partial specifications occuring at differing levels of abstraction. The presence of internal operations in a specification is just one of the consequences of such an approach to large scale software engineering.

In addition, development of viewpoints written in different languages will be undertaken using different refinement relations, and this also motivates the need to develop a notion of weak-refinement in Z which is related to refinements in LOTOS. A full discussion of the relationships between the differing refinement relations is given in [DBBS96b] (which incidentally assumes the firing condition interpretation discussed above).

In this paper we used an example of a telecommunications protocol to show that standard Z refinement is inappropriate for refining a system when internal operations are specified explicitly. We then formulated a generalization of Z refinement, called weak refinement, which treats internal operations differently from observable operations when refining a system. We also discussed the role of internal operations in a Z specification, and in particular whether an equivalent specification not containing internal operations can always be found. If a specification is divergence free we showed that we could find a testing equivalent specification that did not contain internal operations. In the presence of potential livelock we discussed the effect of differing interpretations of divergence have on finding such an equivalent specification.

References

[Abr96] J. R. Abrial. *The B-Book: Assigning programs to meanings.* CUP, 1996.

[BB88] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1988.

[BBDS97] 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)*, LNCS 1313, pages 644–664, Graz, Austria, September 1997. Springer-Verlag.

[BDBS96] E. Boiten, J. Derrick, H. Bowman, and M. Steen. Consistency and refinement for partial specification in Z. In M.-C. Gaudel and J. Woodcock, editors, *FME'96: Industrial Benefit of Formal Methods, Third International Symposium of Formal Methods Europe*, volume 1051 of *Lecture Notes in Computer Science*, pages 287–306. Springer-Verlag, March 1996.

[Ben89] 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.

[BHA84] S. D. Brookes, C.A.R. Hoare, and A.W.Roscoe. A theory of Communicating Sequential Processes. *Journ. ACM*, 31(3):560–599, July 1984.

[Bri88] E. Brinksma. A theory for the derivation of tests. In S. Aggarwal and K. Sabnani, editors, *Protocol Specification, Testing and Verification, VIII*, pages 63–74, Atlantic City, USA, June 1988. North-Holland.

[BSS86] E. Brinksma, G. Scollo, and C. Steenbergen. Process specification, their implementation and their tests. In B. Sarikaya and G. v. Bochmann, editors, *Protocol Specification, Testing and Verification, VI*, pages 349–360, Montreal, Canada, June 1986. North-Holland.

[But97] M. Butler. An approach to the design of distributed systems with B AMN. In J. P. Bowen, M. G. Hinchey, and D. Till, editors, *ZUM'97: The Z formal specification notation*, LNCS 1212, pages 223–241, Reading, April 1997. Springer-Verlag.

[CR92] 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.

[Cus91] E. Cusack. Object oriented modelling in Z for Open Distributed Systems. In J. de Meer, V. Heymer, and R. Roth, editors, *IFIP TC6 International Workshop on Open Distributed Processing*, pages 167–178, Berlin, Germany, September 1991. North-Holland.

[CW92] 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.

[DBBS96a] J. Derrick, E.A. Boiten, H. Bowman, and M. Steen. Supporting ODP - translating LOTOS to Z. In E. Najm and J.-B. Stefani, editors, *First IFIP International workshop on Formal Methods for Open Object-based Distributed Systems*, pages 399–406, Paris, March 1996. Chapman & Hall.

[DBBS96b] 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.

[DBBS97] J. Derrick, E.A. Boiten, H. Bowman, and M. Steen. Weak refinement in Z. In J. P. Bowen, M. G. Hinchey, and D. Till, editors, *ZUM'97: The Z formal specification notation*, LNCS 1212, pages 369–388, Reading, April 1997. Springer-Verlag.

[dNH84] R. de Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34(3):83–133, 1984.

[Eva97] A.S. Evans. A case study in specifying, verifying and refining and parallel system in Z. In *FMPPTA'97*, Geneva, April 1997.

[Fis97] C. Fischer. CSP-OZ - a combination of CSP and Object-Z. In H. Bowman and J. Derrick, editors, *Second IFIP International conference on Formal Methods for Open Object-based Distributed Systems*, pages 423–438. Chapman & Hall, July 1997.

[FS96] A. Finkelstein and G. Spanoudakis, editors. *SIGSOFT '96 International Workshop on Multiple Perspectives in Software Development (Viewpoints '96).* 1996.

[Hen88] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[HMR89] I. Hayes, M. Mowbray, and G.A. Rose. Signalling System No. 7 - The network layer. In E. Brinksma, G. Scollo, and C.A. Vissers, editors, *Protocol Specification Testing and Verification IX*, pages 3–14. North-Holland, 1989.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[ITU95] ITU Recommendation X.901-904 — ISO/IEC 10746 1-4. *Open Distributed Processing - Reference Model - Parts 1-4*, July 1995.

[Jon89] C. B. Jones. *Systematic Software Development using VDM.* Prentice Hall, 1989.

[Lam94] L. Lamport. TLZ. In J.P. Bowen and J.A. Hall, editors, *ZUM'94, Z User Workshop*, pages 267–268, Cambridge, United Kingdom, June 1994.

[Lan97] K. Lano. Specifying reactive systems in B AMN. In J. P. Bowen, M. G. Hinchey, and D. Till, editors, *ZUM'97: The Z formal specification notation*, LNCS 1212, pages 242–274, Reading, April 1997. Springer-Verlag.

[Led91] G. Leduc. *On the Role of Implementation Relations in the Design of Distributed Systems using LOTOS.* PhD thesis, University of Liège, Liège, Belgium, June 1991.

[MD98] B. Mahony and J-SM. Dong. Blending Object-Z and Timed CSP: An introduction to TCOZ. In *Int. Conf. on Software Engineering.* IEEE Computer Press, 1998.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[MZ94] P. Mataga and P. Zave. Formal specification of telephone features. In J. P. Bowen and J. A. Hall, editors, *Eighth Annual Z User Workshop*, pages 29–50, Cambridge, July 1994. Springer-Verlag.

[Raf94] G. H. B. Rafsanjani. ZEST - Z Extended with Structuring: A users's guide. Technical report, BT, June 1994.

[Rud91] S. Rudkin. Modelling information objects in Z. In J. de Meer, V. Heymer, and R. Roth, editors, *IFIP TC6 International Workshop on Open Distributed Processing*, pages 267–280, Berlin, Germany, September 1991. North-Holland.

[Smi95] G. Smith. A fully abstract semantics of classes for Object-Z. *Formal Aspects of Computing*, 7(3):289–313, 1995.

[Smi97] G. Smith. A semantic integration of Object-Z and CSP for the specification of concurrent systems. In J. Fitzgerald, C. B. Jones, and P. Lucas, editors, *Formal Methods Europe (FME '97)*, LNCS 1313, pages 62–81, Graz, Austria, September 1997. Springer-Verlag.

[Spi89] J. M. Spivey. *The Z notation: A reference manual.* Prentice Hall, 1989.

[Str95] 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.

[WD96] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof.* Prentice Hall, 1996.

[WJ94] C. Wezeman and A. J. Judge. Z for managed objects. In J. P. Bowen and J. A. Hall, editors, *Eighth Annual Z User Workshop*, pages 108–119, Cambridge, July 1994. Springer-Verlag.