Working Paper Series
ISSN 1177-777X

COMPARISON OF DATA AND PROCESS REFINEMENT

Steve Reeves and David Streader

Working Paper: 05/03
May 2003

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Comparison of data and process refinement

Steve Reeves and David Streader

Department of Computer Science, University of Waikato, Hamilton, New Zealand
{dstr,stever}@cs.waikato.ac.nz

Abstract. When is it reasonable, or possible, to refine a one place buffer into a two place buffer? In order to answer this question we characterise refinement based on substitution in restricted contexts. We see that data refinement (specifically in Z) and process refinement give differing answers the original question, and we compare the precise circumstances which give rise to this difference by translating programs and processes into labelled transition systems, so providing a common basis upon which to make the comparison. We also look at the closely related area of subtyping of objects. Along the way we see how all these sorts of computational construct are related as far as refinement is concerned, discover and characterise some (as far as we can tell) new sorts of refinement and, finally, point up some research avenues for the future.

Keywords: data refinement, process refinement, labelled transition systems, Z, subtyping, Theoretical paper.

1 Introduction

With advent of both the International Standard on Open Distributed Processing [1] and aspect-oriented programming [2], there has been increasing interest in using different formalisms to specify different views or aspects of the same system. Recent work defining operational semantics, using labelled transition systems (lts), for a variety of formalisms is a step in the direction of integrating differing formalisms [3–5]. For example the execution of an operation of an abstract data type ADT or method of an object is modelled by a transition labelled with the operation or method name. Similarly a program (*i.e.* a sequence of uses of operations of an ADT) can be modelled using transitions labelled $\overline{\mathsf{name}}$ that call operations/methods labelled name.

Although different kinds of things (data types, processes or objects) can be quite naturally given an operational semantics in the form of lts, a potential problem is that the meaning of a specification, *i.e.* what the lts can be refined to, depends upon the kind of thing it represents, as we shall see.

We also note that different kinds of things can be placed in differing contexts. For example, in [4] ADTs can only be placed in contexts (programs) that are traces (*i.e.* sequences) of (calls to) operations, whereas processes can be be placed in contexts modelled by branching transition systems [6].

Combining these observations we construct two general definitions of refinement of lts that are parameterized on the contexts in which the lts can be placed. One definition is in the style of process refinement. The other refinement is more

in a state-based style. This latter originates from the idea that A can be refined into C when no user could "*observe the difference*". We show when the two general definitions are equivalent.

We will, for example, give a one place buffer the same lts semantics whether viewed as a data type or a process. Where the views differ is that as an ADT the one place buffer can be refined into a two place buffer whereas viewed as (*i.e.* placed in the contexts of) a process it cannot. This is understandable given that, by restricting the contexts in which some thing can be placed, what can be observed is restricted and consequently we are less likely to "*observe the difference*".

These definitions of refinement can be applied to different kinds of things. That is to say, the general definition of refinement is made more concrete by fixing the contexts in which the things are to be placed. Doing this we find that our notion of refinement of processes, placed in all process contexts, is equivalent to failure refinement [6] and our notion of refinement of ADTs, placed in all ADT contexts, is equivalent to singleton failure refinement [4]. These are the results we would expect.

In our state-based style of refinement, $\mathsf{A} \sqsubseteq \mathsf{C}$, we restrict the contexts we place C in to those *where* A *was expected*, this restriction of contexts depending on A being very common in behavioural subtyping. We apply this in the general setting and then, when we select a universe of contexts for ADTs and then again for processes, we have definitions of:

ADT refinement related but not equivalent to LOTOS's *ext* [7],

process refinement equivalent to a definition of object-oriented *behavioural subtyping* given in [8].

We define the notation for labelled transition systems in Section 2 and review some definitions of refinement of processes from the literature in Section 2.1. We introduce our items of interest in Section 3 and then we define two formalizations, both parameterized by sets of contexts, of what we mean, in general, by refinement in Section 4.

In Section 5 we use Z to define data types and subsequently define the operational semantics for them based on the *guarded* interpretation (which models most closely what is usual in processes, as opposed to the more 'usual' Z interpretation of chaos outside of preconditions).

In Section 6 we define the contexts in which we can place ADTs and use this and the definition of Section 4 to define ADT refinement. Similarly in Section 7 we define the contexts in which we can place processes and use this and the definition of Section 4 to define process refinement.

We briefly relate our results to subtyping of objects in Section 8 and discuss how to take this work further, and in Section 9 we give pointers to prior work.

In our conclusions Section 10 we summarise our categorisation, our discoveries and set-up agenda for future research.

2 labelled transition systems

In this section we define the notation we will use. It is a combination of notation from ACP [9] and Z [10]. We assume a universe of observable action names Act, from which we build $\overline{Act} \stackrel{\text{def}}{=} \{\overline{a} \mid a \in Act\}$, and then $Act^\tau \stackrel{\text{def}}{=} Act \cup \{\tau\}$

Definition 1 *labelled transition systems*
$\mathsf{A} \stackrel{\text{def}}{=} (Nodes_\mathsf{A}, Tran_\mathsf{A}, s_\mathsf{A})$ *where* $s_\mathsf{A} \in Nodes_\mathsf{A}$ *and* $Tran_\mathsf{A} \stackrel{\text{def}}{=} \{n \xrightarrow{a} m \mid n, m \in Nodes_\mathsf{A} \wedge a \in Act^\tau\}$.

We lift "$-$" to sets of transitions and to labeled transition systems in the obvious way. Any single labeled transition systems will either have transitions labeled form $Act \cup \{\tau\}$ or transitions labeled from $\overline{Act} \cup \{\tau\}$ (used as contexts).

Let $a \in Act$ and $\rho \in Act^*$. We write $\rho\,|_n$ for the n^{th} element of ρ and $\rho\,\lceil_n$ for the first n elements of ρ. We write $\rho \upharpoonright X$ for the sequence ρ with all elements not in set X removed, so $prefix(\rho) \stackrel{\text{def}}{=} \{\rho\,\lceil_n \mid n <\mid \rho\mid\}$.

Where A is obvious from context, we write: $n \xrightarrow{a} m$ for $(n, a, m) \in Tran_\mathsf{A}$, $n \xrightarrow{a}$ for $\exists_m \cdot (n, a, m) \in Tran_\mathsf{A}$ and $n \xrightarrow{\rho} m$ for $\exists_{m_1 \ldots m_i} \cdot (m_1, \rho\,|_1, m_2), \ldots (m_i, \rho\,|_i, m_{i+1}) \in Tran_\mathsf{A} \wedge n = m_1 \wedge m = m_{i+1} \wedge \mid \rho \mid = i$.

$\alpha(\mathsf{A}) \stackrel{\text{def}}{=} \{a \mid n \xrightarrow{a} m \in Tran_\mathsf{A}\}$, $\pi(s) \stackrel{\text{def}}{=} \{a \mid s \xrightarrow{a}\}$

The traces of A are $Tr(\mathsf{A}) \stackrel{\text{def}}{=} \{\rho \mid s_\mathsf{A} \xrightarrow{\rho}\}$ and the complete traces of A are $Tr^c(\mathsf{A}) \stackrel{\text{def}}{=} \{\rho \mid s_\mathsf{A} \xrightarrow{\rho} n \wedge (\pi(n) = \varnothing \vee \mid \rho \mid = \infty)\}$.

$(\mathsf{A})\delta_X \stackrel{\text{def}}{=} (Nodes_\mathsf{A}, Tran_{(\mathsf{A})\delta_X}, s_\mathsf{A})$ where $Tran_{(\mathsf{A})\delta_X} \stackrel{\text{def}}{=} \{n \xrightarrow{a} m \mid n \xrightarrow{a} m \in Tran_\mathsf{A} \wedge a \notin X\}$.

$(\mathsf{A})\tau_X \stackrel{\text{def}}{=} (Nodes_\mathsf{A}, Tran_{(\mathsf{A})\tau_X}, s_\mathsf{A})$ where $Tran_{(\mathsf{A})\tau_X} \stackrel{\text{def}}{=} \{n \xrightarrow{a} m \mid n \xrightarrow{a} m \in Tran_\mathsf{A} \wedge a \notin X\} \cup \{n \xrightarrow{\tau} m \mid n \xrightarrow{a} m \in Tran_\mathsf{A} \wedge a \in X\}$.

The synchronisation function γ_X which maps $(a, b) \mapsto c$ adds c, representing the synchronisation of a and b, where a and b could be performed concurrently.

We treat the synchronization of x and $\overline{x}$ as giving the observable $\overline{x}$. In detail (which the reader may skip), in order to do this, and allow the deletion of unsynchronised $\overline{x}$ actions, we first map them to $\overline{x}^o$ (so γ_X contains $(x, \overline{x}) \mapsto \overline{x}^o$) then delete $\overline{x}$ via $\delta_{\overline{X}}$ and then rename $\overline{x}^o$ to $\overline{x}$ via Ren_X. All this is brought together in the following definition (which we say more about in Section 3):

$- \|_X - \stackrel{\text{def}}{=} ((- \|_{\gamma_X} -)\delta_{\overline{X}})Ren_X$

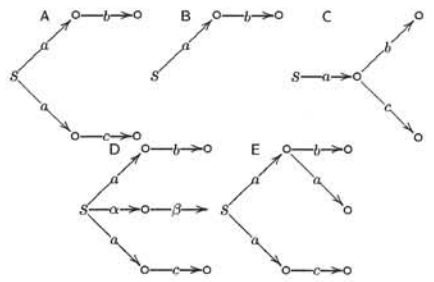
Finally we have refusal sets: $Ref(\rho, \mathsf{C}) \stackrel{\text{def}}{=} \{X \mid \exists \rho.s_\mathsf{C} \xrightarrow{\rho} s \wedge X \subseteq Act - \pi(s)\}$ and singleton refusal sets: $Ref_s(\rho, \mathsf{C}) \stackrel{\text{def}}{=} \{\{a\} \mid \exists \rho.s_\mathsf{C} \xrightarrow{\rho} s \wedge a \in Act - \pi(s)\}$.

2.1 Some known refinement relations for action-based systems

Hennessy's "may and must" testing refinement [11] $\sqsubseteq_{test}$ is the most constrained form of refinement we consider. A relaxation is LOTOS's extensional refinement $\sqsubseteq_{ext}$ [7] which allows feature addition in the form of both alphabet extension α^+ and the addition of new traces Tr^+ over the original alphabet. We

define $\sqsubseteq_{pro}$, that only introduces alphabet extension α^+, from which we have $\sqsubseteq_{test} \Rightarrow \sqsubseteq_{pro} \Rightarrow \sqsubseteq_{ext}$.

Each of these can be relaxed by adding the ability to prune nondeterministic traces, see $\mathsf{A} \sqsubseteq \mathsf{B}$ Figure 1. Adding this ability to $\sqsubseteq_{test}$ forms $\sqsubseteq_{Ftest}$ (failure refinement [6], which is known to be equivalent to must testing refinement [11]) and $\sqsubseteq_{Fpro}$, which is shown Lemma 1 to be equivalent to "weak subtyping" [8].



		Prune	α^+	**Tr**$^+$
[11]	test	×	×	×
	pro	×	√	×
[7]	ext	×	√	√
[6]	Ftest	√	×	×
[8]	Fpro	√	√	×
	Fext	√	√	√
	$\mathsf{A} \sqsubseteq$	B	D	E

Fig. 1. $\mathsf{A} \sqsubseteq_{F....} \mathsf{B}$ $\mathsf{A} \sqsubseteq_{test} \mathsf{C}$ $\mathsf{A} \sqsubseteq_{pro} \mathsf{C}, \mathsf{D}$ $\mathsf{A} \sqsubseteq_{ext} \mathsf{C}, \mathsf{D}, \mathsf{E}$

Definition 2 *Process refinements. Let* $New = (\alpha(\mathsf{C}) - \alpha(\mathsf{A}))$*:*

$(\mathsf{A} \sqsubseteq_{test} \mathsf{C}) \Leftrightarrow Tr(\mathsf{A}) = Tr(\mathsf{C}) \wedge \forall \rho . Ref(\rho, \mathsf{C}) \subseteq Ref(\rho, \mathsf{A})$.

$(\mathsf{A} \sqsubseteq_{pro} \mathsf{C}) \Leftrightarrow Tr(\mathsf{A}) = Tr(\mathsf{C}\delta_{New}) \wedge \forall \rho \in Tr(\mathsf{A}) . Ref(\rho, \mathsf{C}) \subseteq Ref(\rho, \mathsf{A})$.

$(\mathsf{A} \sqsubseteq_{ext} \mathsf{C}) \Leftrightarrow Tr(\mathsf{A}) \subseteq Tr(\mathsf{C}) \wedge \forall \rho \in Tr(\mathsf{A}) . Ref(\rho, \mathsf{C}) \subseteq Ref(\rho, \mathsf{A})$.

$(\mathsf{A} \sqsubseteq_{Ftest} \mathsf{C}) \Leftrightarrow \forall \rho . Ref(\rho, \mathsf{C}) \subseteq Ref(\rho, \mathsf{A})$.

$(\mathsf{A} \sqsubseteq_{Fpro} \mathsf{C}) \Leftrightarrow Tr(\mathsf{A}) \supseteq Tr(\mathsf{C}\delta_{New}) \wedge \forall \rho \in Tr(\mathsf{A}) . Ref(\rho, \mathsf{C}) \subseteq Ref(\rho, \mathsf{A})$

$(\mathsf{A} \sqsubseteq_{Fext} \mathsf{C}) \Leftrightarrow \forall \rho \in Tr(\mathsf{A}) . Ref(\rho, \mathsf{C}) \subseteq Ref(\rho, \mathsf{A})$.

All the above are based on refusals $Ref(\rho, X)$, from [4, 12] we see that ADTs are more appropriately based on singleton refusals $Ref_s(\rho, X)$. Consequently by replacing $Ref(\rho, X)$ with $Ref_s(\rho, X)$ in the above we have a whole new set of refinement relations $\sqsubseteq^s_X$ tailored for ADTs.

Although we have not found references to $\sqsubseteq_{Fext}$ or $\sqsubseteq_{pro}$ they can be seen as little more than completing the square in Figure 1. More interestingly, we find (as we shall see) that our definition of ADT refinement corresponds to a singleton version of $\sqsubseteq_{Fext}$ *i.e.* $\sqsubseteq^s_{Fext}$.

$\mathsf{A} \sqsubseteq_{Ftest} \mathsf{C}\delta_{New}$ has been used in *weak subtyping* in [8] where they take as a requirement of behavioural subtyping that if $New = \varnothing$ then refinement should be failure refinement. A consequence of this decision, as we shall see, is that a one place buffer cannot be refined into a two place buffer.

Lemma 1 $\mathsf{A} \sqsubseteq_{Fpro} \mathsf{C} \Leftrightarrow \mathsf{A} \sqsubseteq_{Ftest} \mathsf{C}\delta_{New}$

Proof Step 1. $\mathsf{A} \sqsubseteq_{Fpro} \mathsf{C} \Rightarrow \mathsf{A} \sqsubseteq_{Ftest} \mathsf{C}\delta_{New}$ Assume $\mathsf{A} \sqsubseteq_{Fpro} \mathsf{C}$.

$\mathsf{A} \sqsubseteq_{Fpro} \mathsf{C} \stackrel{\text{def}}{=} Tr(\mathsf{A}) \supseteq Tr(\mathsf{C}\delta_{New}) \wedge \forall \rho \in Tr(\mathsf{A}) . Ref(\rho, \mathsf{C}) \subseteq Ref(\rho, \mathsf{A})$.

1. $\forall \rho \in Tr(\mathsf{A}).Ref(\rho, \mathsf{C}) \subseteq Ref(\rho, \mathsf{A}) \Rightarrow \forall \rho \in Tr(\mathsf{A}).Ref(\rho, \mathsf{C}\delta_{New}) \subseteq Ref(\rho, \mathsf{A})$.

As $Tr(\mathsf{A}) \supseteq Tr(\mathsf{C}\delta_{New})$ if $\rho \notin Tr(\mathsf{A})$ then $\rho \notin Tr(\mathsf{C}\delta_{New})$. Hence 2. if $\rho \notin Tr(\mathsf{A})$ then $Ref(\rho, \mathsf{C}\delta_{New}) \subseteq Ref(\rho, \mathsf{A})$.

From 1.and 2. q $\forall \rho.Ref(\rho, \mathsf{C}\delta_{New}) \subseteq Ref(\rho, \mathsf{A}) \stackrel{\text{def}}{=} (\mathsf{A} \sqsubseteq_{Ftest} \mathsf{C}\delta_{New})$.

Step 2. $\mathsf{A} \sqsubseteq_{Ftest} \mathsf{C} \Rightarrow \delta_{New}\mathsf{A} \sqsubseteq_{Fpro} \mathsf{C}$ .Assume $\mathsf{A} \sqsubseteq_{Ftest} \mathsf{C}$.

$(\mathsf{A} \sqsubseteq_{Ftest} \mathsf{C}\delta_{New}) \stackrel{\text{def}}{=} \forall \rho.Ref(\rho, \mathsf{C}\delta_{New}) \subseteq Ref(\rho, \mathsf{A})$.

1. $\forall \rho.Ref(\rho, \mathsf{C}\delta_{New}) \subseteq Ref(\rho, \mathsf{A}) \Rightarrow Tr(\mathsf{A}) \supseteq Tr(\mathsf{C}\delta_{New})$.

2. $\forall \rho.Ref(\rho, \mathsf{C}\delta_{New}) \subseteq Ref(\rho, \mathsf{A}) \Rightarrow \forall \rho \in Tr(\mathsf{A}).Ref(\rho, \mathsf{C}\delta_{New}) \subseteq Ref(\rho, \mathsf{A})$. As $New = (\alpha(\mathsf{C}) - \alpha(\mathsf{A}))$ then $Ref(\rho, \mathsf{C}\delta_{New}) \subseteq Ref(\rho, \mathsf{A}) \Rightarrow Ref(\rho, \mathsf{C}) \subseteq Ref(\rho, \mathsf{A})$. Hence $\forall \rho \in Tr(\mathsf{A}).Ref(\rho, \mathsf{C}) \subseteq Ref(\rho, \mathsf{A})$. Hence from 1. and 2.

$Tr(\mathsf{A}) \supseteq Tr(\mathsf{C}\delta_{New}) \wedge \forall \rho \in Tr(\mathsf{A}).Ref(\rho, \mathsf{C}) \subseteq Ref(\rho, \mathsf{A}) \stackrel{\text{def}}{=} \mathsf{A} \sqsubseteq_{Fpro} \mathsf{C}$.

•

3 Things and contexts of interest

Our 'things' could be abstract data types, processes or even objects, all of which we introduce and consider later. Both things and the contexts in which we place them are given a labelled transition system semantics. Different kinds of things can be placed in different sets of contexts. The use of different sets of contexts for different kinds of thing can be seen in [4, 8].

Placing 'thing' T in a context X is written $[\mathsf{T}]_X$ and must model the synchronization between actions of things such as method m and actions of contexts such as calling method m, *i.e.* $\overline{m}$.

The resulting synchronized actions may be private, *i.e.* τ actions. Any action of the context that is not private is observable by an "independent observer". A consequence of this is that although communication between thing and context may be unobserved (τ) it is easy to amend any context by adding actions that make observable any of the unobservable synchronizations. Consequently, we will treat the synchronization of x and $\overline{x}$ as giving the observable $\overline{x}$. In order to allow the deletion of unsynchronized $\overline{x}$ actions we use $\_ \|_X \_ \stackrel{\text{def}}{=} (\_ \|_{\gamma_X} \_)\delta_{\overline{X}})Ren_X$ (see Section 2 above).

We assume that all observable actions of T require synchronization with some other thing in order to be performed. We can only view our things via their synchronization with the context and we can view all synchronization with the context. Hence, no observable action of T can be performed on its own (formalised by $(\_)\delta_{Act}$). So, we have

$$[\mathsf{T}]_X \stackrel{\text{def}}{=} (\mathsf{T} \|_{\alpha(\mathsf{T})} X)\delta_{Act}$$

Further, we assume that we can wait long enough so that if something observable will eventually happen we do see it. This amounts to an observation being a *complete trace* (the set of observable traces is not prefix closed).

Hence

$$Obs([\mathsf{T}]_X) \stackrel{\text{def}}{=} Tr^c([\mathsf{T}]_X).$$

Assumption 1 (*a*) *Things and their contexts can be given a lts semantics.* (*b*) *The kind of a thing can be characterized by the set of contexts it can be placed in.* (*c*) *A thing's actions can only be executed in synchronization with actions from the context.* (*d*) *All synchronizations of a thing with actions from the context are observed.* (*e*)*All that we can observe are the complete traces of context.*

4 Refinement, observation and contexts.

Refinement is a step in the construction of an implementation from a specification. The refinement of A (something abstract) into C (something more concrete) will be written $\mathsf{A} \sqsubseteq \mathsf{C}$. We allow the C to have new operations not found in A, $New \stackrel{\text{def}}{=} \alpha(\mathsf{C}) - \alpha(\mathsf{A})$. We will formalize refinement in two related styles:

Action-based style where the observation of an execution of $[\mathsf{A}]_X$ is interpreted as success or failure and refinement is based on a preorder representing improvement.

State-based style refinement based on "substitutability", C being a refinement of A when the substitution of A, in a context *where* A *was expected*, by C, cannot be observed.

The first style is a small modification of Hennessy's [11], and when applied to processes it gives the same refinements as Hennessy's. This style generates different refinements depending on the preorder used.

The second style appears [13, 14, 5, 15, 8] as behavioural subtyping and hence could be thought of as object refinement. The "*not being able to tell*" will be formalized as subset of observations. In the case when the contexts are programs this becomes equivalent to the definition of data refinement as subset of the relational semantics of programs as found in [16–18]. For data refinement where operations are undefined (and so can have any behaviour, sometimes referred to as chaos) outside pre-conditions the restriction of programs to those *where* A *was expected* is redundant But, for data refinement where operations are guarded it is this restriction that permits feature addition.

Because of the links between the two styles of definitions we will apply the notion of "*where* A *was expected*" to the first style, thereby introducing the feature addition permitted by the second style. When applied to ADTs this will result in a refinement weaker than LOTOS's ext refinement.

4.1 Action-based refinement

We are going to place A in a context $[\_]_X$ and observe then via $Obs([\mathsf{A}]_X)$. Depending upon the context we interpret an observation as being a success or failure. For concurrent and distributed systems there are a huge number of contexts that can be constructed. It is useful to construct a "core" set of contexts from which we can infer what we might see had we placed the thing in a context not in the core set.

A single observation of a thing T in a context X is a complete trace of $[\mathsf{T}]_X$ and will be interpreted as $\top$ (success) if and only if it is also a complete trace of the context X. Being interested in nondeterminism we assume that an observation consists of a set of single observations of the same thing and context. Such observations are given one of the following three interpretations: $\{\top\}$—always succeed; $\{\top, \bot\}$—may succeed or may fail; and $\{\bot\}$—always fail.

There are three powerdomains on the two point lattice $\top > \bot$ (see Figure 2). We are only interested in two of them: we will ignore the *Hoare* powerdomain[1] and use the other two powerdomains to impose a preorder on the observations.

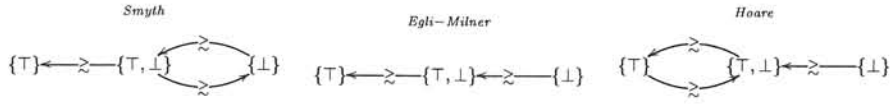


Fig. 2. Power domains

Definition 3 . $Obs([\mathsf{A}]_x) \stackrel{\text{def}}{=} Tr^c([\mathsf{A}]_x)$
$\top \in I([\mathsf{A}]_x) \Leftrightarrow \exists_{\rho \in Obs([\mathsf{A}]_x)} . \rho \in Tr^c(x)$
$\bot \in I([\mathsf{A}]_x) \Leftrightarrow \exists_{\rho \in Obs([\mathsf{A}]_x)} . \rho \in prefix(Tr^c(x))$ *and nothing else is in* $I([\mathsf{A}]_x)$.

$Obs([\mathsf{C}]_x) \gtrsim Obs([\mathsf{A}]_x) \stackrel{\text{def}}{=} I([\mathsf{C}]_x) > I([\mathsf{A}]_x) \vee (I([\mathsf{A}]_x) = I([\mathsf{C}]_x) \wedge Obs([\mathsf{A}]_x) \supseteq Obs([\mathsf{C}]_x))$

$\mathsf{A} \sqsubseteq \mathsf{C} \stackrel{\text{def}}{=} \forall_{[\_]_x \in [\_]} . Obs([\mathsf{C}]_x) \gtrsim Obs([\mathsf{A}]_x)$. •

This definition of refinement depends on :

1. the set of contexts used $[\_]$
2. and what preorder $>$ we apply to our interpretations $I([\_]_x)$

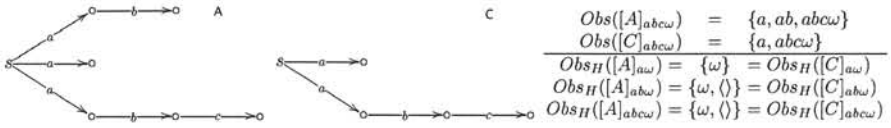
Hennessy [11] uses a "*success state*" approach in which tests formalize the notion of observation. A special action ω is introduced and use to decorate the success states $\{s \xrightarrow{\omega} \mid s \in Succ\}$. Then a test (an execution of a process in a context) is interpreted as being a success when it reaches a success state (when ω is observed).

Here end states ($\pi(n) = \varnothing$) can be viewed as our success states, but whereas Hennessy only allows ω to be visible, we allow the observation of the whole trace of executed actions. These two treatments can be shown (see Lemma 11 later) to result in the same refinement relation when applied to processes. But, as we now demonstrate, the two treatments define *different* refinements when applied to ADTs.

Applying the "*success state*" approach to ADTs (where contexts are traces) we interpret a test as being a success if the context reaches a success state. Clearly this is equivalent to restricting contexts to $\rho\omega$ and treating only ω as visible, $Obs_H(\_) \stackrel{\text{def}}{=} Obs(\_)\tau_{(Act-\{\omega\})}$. Using this definition of observation we

[1] The *Hoare* powerdomain has been used [11] to define 'may' testing, which is equivalent to trace refinement. Here we can achieve the same results by restricting the lts used to represent both things and contexts

can see that A and C in Figure 3 are observationally equivalent. But they are not observationally equivalent using our definition of observation as completed traces.



$$\begin{array}{rcl} Obs([A]_{abc\omega}) & = & \{a, ab, abc\omega\} \\ Obs([C]_{abc\omega}) & = & \{a, abc\omega\} \\ \hline Obs_H([A]_{a\omega}) = \{\omega\} & & = Obs_H([C]_{a\omega}) \\ Obs_H([A]_{ab\omega}) = \{\omega, \langle\rangle\} & & = Obs_H([C]_{ab\omega}) \\ Obs_H([A]_{abc\omega}) = \{\omega, \langle\rangle\} & & = Obs_H([C]_{abc\omega}) \end{array}$$

Fig. 3. $Obs(\mathsf{A}) \neq Obs(\mathsf{C})$ but $Obs_H(\mathsf{A}) = Obs_H(\mathsf{C})$

Although the "success state" approach seems a perfectly reasonable way to define refinement we do not pursue this here.

4.2 Contexts where A is expected

Our state-based notion of refinement is going to be based upon "substituting" A with C in contexts *where A was expected.* Here we will formalize this idea and apply it to our action-based definition of refinement.

Definition 4 $\mathsf{A} \sqsubseteq \mathsf{C}$ *iff* C *may be used in any context where* A *was expected, without the client being able to tell.*

Our contexts for things T are $[\mathsf{T}]_X \stackrel{\text{def}}{=} (\mathsf{T} \parallel_{\alpha(\mathsf{T})} X)\delta_{Act}$. Note the context synchronizes on actions of T and then all unsynchronized actions get deleted (δ_{Act}).

Assumption 2 *Contexts where* A *is expected can only synchronize with (call) actions of* A

Consequently: $$[\_]^A \subseteq \{(\_ \parallel_{\alpha(\mathsf{A})} X)\delta_{Act} \mid X \in \overline{lts}\}.$$

Assumption 3 *That A must fail in a certain context whereas C might succeed is not to sufficient to distinguish A from C.*

Hence "contexts where A is expected" are not contexts where A must fail.

Definition 5 *"contexts where* A *is expected"*

$$[\_]^A \stackrel{\text{def}}{=} \{(\_ \parallel_{\alpha(\mathsf{A})} X)\delta_{Act} \mid X \in \overline{lts} \wedge I((\_ \parallel_{\alpha(\mathsf{A})} X)\delta_{Act}) \neq \{\bot\}\} .$$

Definition 6 $\mathsf{A} \sqsubseteq^A \mathsf{C} \stackrel{\text{def}}{=} \forall_{[\_]_a \in [\_]^A} . Obs([\mathsf{C}]_a) \gtrsim Obs([\mathsf{A}]_a)$.

Prior to restricting the contexts, our two definitions of refinement, applied to processes, will be the same as two of Hennessy's testing refinements. When we apply this definition, with restricted contexts, to ADTs and processes we will find that our definitions of refinement are very similar to that of LOTOS's ext refinement.

4.3 State-based refinement

Early work [16] defines refinement as subset on the relational semantics and quantifies over all contexts (programs). This can be rephrased as: for all inputs (contexts), we must have a subset of outputs (what can be observed). In a similar fashion we define refinement by explicitly defining contexts $[\_]^A$ or $[\_]$ and use subsets of observations.

Definition 7 $\mathsf{A} \sqsubseteq_{State} \mathsf{C} \stackrel{\text{def}}{=} \forall_{[\_]_a \in [\_]} . Obs([\mathsf{C}]_a) \subseteq Obs([\mathsf{A}]_a)$.

$\mathsf{A} \sqsubseteq^A_{State} \mathsf{C} \stackrel{\text{def}}{=} \forall_{[\_]_a \in [\_]^A} . Obs([\mathsf{C}]_a) \subseteq Obs([\mathsf{A}]_a)$.

$[\![\mathsf{P}]\!]_R \stackrel{\text{def}}{=} \{\langle [\_]_x, o\rangle \mid o \in Obs([\mathsf{P}]_x), [\_]_x \in [\_]\}$ $\quad \mathsf{A} \sqsubseteq_R \mathsf{C} \stackrel{\text{def}}{=} [\![\mathsf{C}]\!]_R \subseteq [\![\mathsf{A}]\!]_R$

$[\![\mathsf{P}]\!]^A_R \stackrel{\text{def}}{=} \{\langle [\_]_x, o\rangle \mid o \in Obs([\mathsf{P}]_x), [\_]_x \in [\_]^A\}$ $\quad \mathsf{A} \sqsubseteq^A_R \mathsf{C} \stackrel{\text{def}}{=} [\![\mathsf{C}]\!]^A_R \subseteq [\![\mathsf{A}]\!]^A_R$ $\quad \circ$

Clearly $\mathsf{A} \sqsubseteq_{State} \mathsf{C} \Leftrightarrow \mathsf{A} \sqsubseteq_R \mathsf{C}$ and $\mathsf{A} \sqsubseteq^A_{State} \mathsf{C} \Leftrightarrow \mathsf{A} \sqsubseteq^A_R \mathsf{C}$ Definition 7 is, by design, closely related to Z data refinement [4].

It is easy to see that if we assume the Smyth powerdomain and use the previously computed contexts then the above definitions are a characterization of our previously defined action-based refinements.

Once we have restricted the contexts to *contexts where A is expected*, as in Definition 6, then $\stackrel{\gtrsim}{\leftarrow}\{\bot\}$ is redundant. Consequently using the restricted relations in Figure 4 will have the same effect as using the powerdomains.

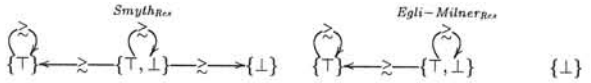


Fig. 4. Restrictions of powerdomains

Lemma 2 *Assume the Smyth powerdomain*

$$\mathsf{A} \sqsubseteq^A \mathsf{C} \Leftrightarrow \mathsf{A} \sqsubseteq^A_{State} \mathsf{C} \Leftrightarrow \mathsf{A} \sqsubseteq^A_R \mathsf{C}.$$

Proof Second equivalence is obvious. For first equivalence:

1. By definition if $Obs([\mathsf{C}]_a) \gtrsim Obs([\mathsf{A}]_a)$ then $I([\mathsf{C}]_a) > I([\mathsf{A}]_a) \vee (I([\mathsf{A}]_a) = I([\mathsf{C}]_a) \wedge Obs([\mathsf{A}]_a) \supseteq Obs([\mathsf{C}]_a))$. As if $I([\mathsf{C}]_a) > I([\mathsf{A}]_a)$ then from Figure 4 $Obs([\mathsf{C}]_a) \subseteq Obs([\mathsf{A}]_a)$. Consequently if $Obs([\mathsf{C}]_a) \gtrsim Obs([\mathsf{A}]_a)$ then $Obs([\mathsf{C}]_a) \subseteq Obs([\mathsf{A}]_a)$.

$\forall_{[\_]_a \in [\_]^A} . Obs([\mathsf{C}]_a) \gtrsim Obs([\mathsf{A}]_a) \Rightarrow Obs([\mathsf{C}]_a) \subseteq Obs([\mathsf{A}]_a)$

2. Similarly $\forall_{[\_]_a \in [\_]^A} . Obs([\mathsf{C}]_a) \subseteq Obs([\mathsf{A}]_a) \Rightarrow Obs([\mathsf{C}]_a) \gtrsim Obs([\mathsf{A}]_a)$

From 1. and 2.

$\forall_{[\_]_a \in [\_]^A} . Obs([\mathsf{C}]_a) \gtrsim Obs([\mathsf{A}]_a) \Leftrightarrow Obs([\mathsf{C}]_a) \subseteq Obs([\mathsf{A}]_a)$

$\mathsf{A} \sqsubseteq^A \mathsf{C} \Leftrightarrow \forall_{[\_]_a \in [\_]^A} . Obs([\mathsf{C}]_a) \subseteq Obs([\mathsf{A}]_a)$ $\quad \bullet$

Because what can be observed in one context may restrict what can be observed in other "similar" contexts, and because refinement quantifies over a

universe of contexts, we can show that, even without restricting the contexts, some of the powerdomain is redundant.

Lemma 3 *If* $\mathsf{A}]_\rho) = \{\bot\} \wedge I([\mathsf{C}]_\rho) \neq \{\bot\}$ $I([\mathsf{C}]_\rho) \xrightarrow{\gtrsim} I([\mathsf{A}]_\rho)$ *then* $Obs([\mathsf{A}]_{\rho a}) \not\gtrsim Obs([\mathsf{C}]_{\rho a})$

Proof

As $I([\mathsf{A}]_\rho) = \{\bot\}$ then $\rho \notin Obs([\mathsf{A}]_\rho)$ and as $I([\mathsf{C}]_\rho) \neq \{\bot\}$ then $\rho \in Obs([\mathsf{C}]_\rho)$. Hence $Obs([\mathsf{C}]_\rho) \not\subseteq Obs([\mathsf{A}]_\rho)$ i.e. $Tr^c([\mathsf{C}]_\rho) \not\subseteq Tr^c([\mathsf{A}]_\rho)$. Select an a such that $\rho a \notin Tr^c([\mathsf{C}]_{\rho a})$ hence $Tr^c([\mathsf{C}]_{\rho a}) = Tr^c([\mathsf{C}]_\rho)$. As $I([\mathsf{A}]_{\rho a}) = \{\bot\} = I([\mathsf{C}]_{\rho a})$ and $Obs([\mathsf{A}]_{\rho a}) \not\supseteq Obs([\mathsf{C}]_{\rho a})$ we have $Obs([\mathsf{A}]_{\rho a}) \not\gtrsim Obs([\mathsf{C}]_{\rho a})$.

•

Lemma 4 *Assume the Smyth powerdomain*

$$\mathsf{A} \sqsubseteq \mathsf{C} \Leftrightarrow \mathsf{A} \sqsubseteq_{State} \Leftrightarrow \mathsf{A} \sqsubseteq_R \mathsf{C}.$$

Proof From Lemma 2 and Lemma 3 •

Thus refinements based on both the Smyth powerdomain and the Egli-Milner powerdomain reduce nondeterminism, but refinements based on the Egli-Milner powerdomain (which will be a restriction of refinements based on the Smyth powerdomain) will also increase the likelihood of success.

An advantage of Definition 7 is that it is based on subsets of observations and not a more abstract interpretation of the observations and an ($\gtrsim$) improve relation. On the other hand, starting from a definition of $\gtrsim$ we find that: 1—$[\_]^A$ the set contexts where A is expected can, based on stated assumptions, be computed; and 2—we have not excluded the Egli-Milner powerdomain. Hence we have not excluded testing refinement $\sqsubseteq_{test}$ [11] nor have we excluded LOTOS's extensional refinement $\sqsubseteq_{ext}$ [7].

Summary Our action-based definition of refinement $\sqsubseteq$ depends upon:

1. what set of contexts $[\_]$ we use
2. what powerdomain we use—Smyth or Egli-Milner

The set of contexts $[\_]$ we use defines what kind of thing we model *e.g.* ADTs, processes *etc.*. If we use the Smyth powerdomain then we can characterize refinement as a subset of observations. Based on the stated assumptions we can compute the contexts "where A is expected" ($[\_]^A$). If we restrict ourselves to these contexts we have refinement $\sqsubseteq^A$ which permits "feature addition".

Our relational or state-based refinement $\sqsubseteq_{State}$ starts with a definition of the contexts "where A is expected" ($[\_]^A$). If we choose the same set as those computed in the action-based style then, by Lemma 4, our state-based refinement has been proven to be the same as the action-based refinement with the Smyth powerdomain.

5 Using Z to define data types

We might refer to a *one place buffer* as a data type, whether the buffer was empty or not. As an alternative we will follow the convention from the world of processes and regard a data type to define both its operational behaviour and an initial state. Thus, for us, strictly speaking, an *empty one place buffer* is a different data type from a *full one place buffer.*

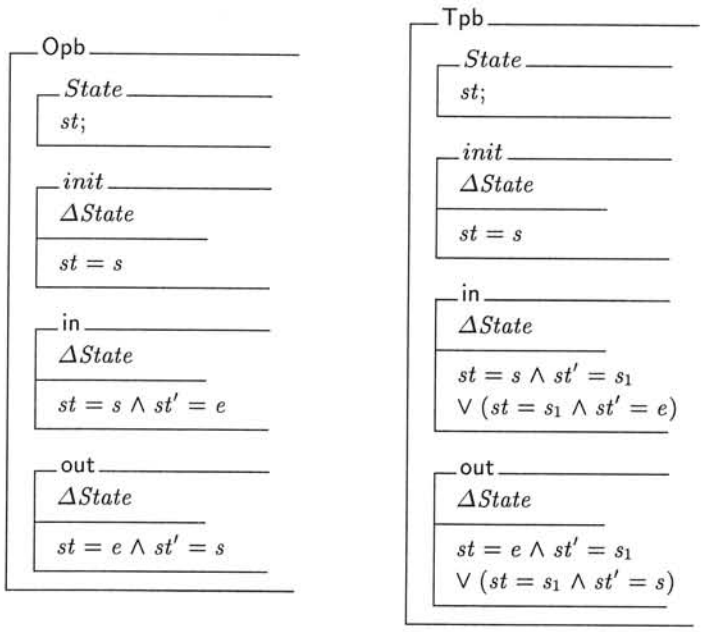


Fig. 5. Z-ADT Opb and Tpb

5.1 Two interpretations of Z actions

The normal interpretation of Z operational schemas, that they are *undefined* (*i.e.* specify arbitrary behaviour) outside their preconditions, has been given a relational semantics $_{ZR}[\![\_]\!]_u$ in [17]. Another interpretation (variously called *behavioural, abortive* or *guarded*) is given a relational semantics $_{ZR}[\![\_]\!]_g$ in [4]. A detailed comparison of refinement of both interpretations of Z operations can be found in [19]. Unless stated we will assume the *guarded* interpretation.

5.2 Z abstract data types

The state-and-operations style of Z specification can be *interpreted* as an ADT-specification style, but Z offers no structuring mechanisms to formalize this. Consequently we use the notation of [18].

Data types consist of a single state schema, an initialising operation schema and a set of operation schemas.

$$\mathsf{A} \stackrel{\text{def}}{=} (State_{\mathsf{A}}, init_{\mathsf{A}}, Op_{\mathsf{A}})$$

5.3 From data types to lts

We can assume the guarded semantics for a Z ADT A giving the lts ${}_Z[\![\mathsf{A}]\!]_g$, the semantics from [20–22] simplified by not having value passing.

1. $Nodes^A \stackrel{\text{def}}{=} State_{\mathsf{A}} \qquad Start^A \stackrel{\text{def}}{=} init_{\mathsf{A}} \wedge State_{\mathsf{A}}$
2. ${}_Z[\![\_]\!]_g\ Tran^A \stackrel{\text{def}}{=} \{x \xrightarrow{\mathsf{n}} y \mid (\mathsf{n} \in Op_{\mathsf{A}}) \wedge x \in State_{\mathsf{A}} \wedge \mathsf{n} \wedge y \in State'_{\mathsf{A}})\}.$

5.4 Z relational semantics and data Refinement

Data refinement, forward simulation and backward simulation are defined in [16]. Later on, *Z data refinement* [10] is defined and in [17] shown to be equivalent to forward simulation of [16]. We use data refinement as defined in [16–18] and think of forward and backward simulation to be techniques to compute refinement.

Data refinement of A is defined on *programs*, *i.e.* sequences ρ of 'calls' of operations. Each operation a is given a relational semantics ${}_R[\![\mathsf{a}]\!]$ and the semantics of the programs $\rho \stackrel{\text{def}}{=} \overline{\mathsf{a}^1}\,\overline{\mathsf{a}^2}\ldots$ on a data type is constructed from the relational semantics plus an initialization and finalization relation. Where operations cannot perform input and output the construction is simply relational compositions $[\![\mathsf{A}_\rho]\!]_Z = init \circ {}_R[\![\mathsf{a}^1]\!] \circ {}_R[\![\mathsf{a}^2]\!] \ldots \circ final$. What can be observed of any program is defined by the finalising operation. Slightly different ways to define the relational semantics of programs (that use operations with input and output) can be found in the literature. For details see [17, 18, 4].

Having defined the semantics of a program, $[\![\mathsf{A}_\rho]\!]_Z$, refinement is defined:

$$A \sqsubseteq_Z C \stackrel{\text{def}}{=} \forall_{\rho \in Prog} \cdot [\![[C]_\rho]\!]_Z \subseteq [\![[A]_\rho]\!]_Z.$$

For us there are two important questions: 1—what is the set of programs *Prog*?; and 2—what does finalising make observable?

When a program terminates finalising, as in [17, 18, 4], returns the output sequence of values (where blank _ is returned where no value is output by an operation). This contrasts with the approach taken in [23] where the refusal set is taken as observable.

But how programs that do not terminate are treated varies. We note that in [4] finalising returns a sequence of the same length as the sequence of operations that did not terminate. We write $[\![[A]_\rho]\!]_{Zg}$ for the relational semantics, defined in [4], of program ρ using data type A.

$$A \sqsubseteq_{Zg} C \stackrel{\text{def}}{=} \forall_{\rho \in Prog} \cdot [\![[C]_\rho]\!]_{Zg} \subseteq [\![[A]_\rho]\!]_{Zg}.$$

Clearly, knowing the program ρ and how many operations terminated (n) we can infer the complete observational trace $\rho \upharpoonright_n$. Had [4] defined finalising to return $\perp$ only when a program does not terminate, then, by Lemma 4, the definition of observation, would appear[2], to correspond to Hennessy's "*success state*" definition (see Section 4.1 above).

[2] The proof of Lemma 2 still applies but no the proof of Lemma 3 and henceLemma 4.

6 Sequential data types

An informal and common argument exists that a two place buffer Tpb is a refinement of a one place buffer Opb because "replacing a one place buffer with a two place buffer cannot be noticed" (see Figure 5). Similarly Opbdel Figure 6 can be seen as Opb with a delete feature added and hence we would like Opbdel to be a refinement of Opb. We consider these arguments further below.

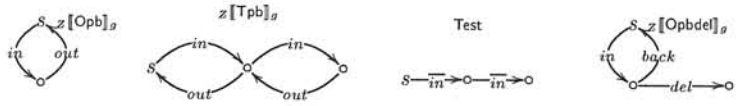


Fig. 6. Opb $\sqsubseteq^s_{Fext}$ Tpb Opb $\sqsubseteq^s_{Fext}$ Opbdel

ADT refinement In order to apply our approach from Section 4 we need to define the contexts in which ADTs can be placed.

$$[\_] \stackrel{\text{def}}{=} \{(\_ \|_{\alpha(\_)} \rho))\delta_{Act} \mid \rho \in \overline{Act}^*\}.$$

We use these contexts throughout (see Section 6) and apply Definition 5 to compute "contexts where A is expected" to be:

$$[\_]^A = \{(\_ \|_{\alpha(\mathsf{A})} \rho))\delta_{Act} \mid \overline{\rho} \restriction \alpha(\mathsf{A}) \in Tr(\mathsf{A})\}$$

Now we can define ADT refinement as:

$$\mathsf{A} \sqsubseteq_{DT} \mathsf{C} \stackrel{\text{def}}{=} \mathsf{A} \sqsubseteq \mathsf{C} \text{ and } \mathsf{A} \sqsubseteq_{DT^A} \mathsf{C} \stackrel{\text{def}}{=} \mathsf{A} \sqsubseteq^A \mathsf{C}$$

Assuming the Smyth powerdomain then from Lemma 2 and Lemma 4 we have:

$$\mathsf{A} \sqsubseteq_{DT} \mathsf{C} \Leftrightarrow \forall_{[\_]_a \in [\_]} . Obs([\mathsf{C}]_a) \subseteq Obs([\mathsf{A}]_a).$$
$$\mathsf{A} \sqsubseteq_{DT^A} \mathsf{C} \Leftrightarrow \forall_{[\_]_a \in [\_]_A} . Obs([\mathsf{C}]_a) \subseteq Obs([\mathsf{A}]_a).$$

Data refinements on Z We use the semantic mappings from Z to lts to give either a guarded or undefined interpretation to the operations.

$$\mathsf{A} \sqsubseteq_{DTg} \mathsf{C} \stackrel{\text{def}}{=} [\![\mathsf{A}]\!]_g \sqsubseteq_{DT} [\![\mathsf{C}]\!]_g \text{ and } \mathsf{A} \sqsubseteq_{DTg^A} \mathsf{C} \stackrel{\text{def}}{=} [\![\mathsf{A}]\!]_g \sqsubseteq_{DT^A} [\![\mathsf{C}]\!]_g$$

Although Opb $\not\sqsubseteq_{DT}$ Tpb and Opb $\not\sqsubseteq_{DT}$ Opbdel, the restriction on contexts with $\sqsubseteq_{DT^A}$ prevents $_Z[\![\mathsf{Opb}]\!]_g$ being placed in contexts (programs) such as $\overline{in};\overline{in}$ and $\overline{in};\overline{del}$. So, we have our desired results: Opb $\sqsubseteq_{DT^A}$ Tpb and Opb $\sqsubseteq_{DT^A}$ Opbdel.

Lemma 5 $\mathsf{A} \sqsubseteq_{DT} \mathsf{C} \Leftrightarrow \mathsf{A} \sqsubseteq^s_F \mathsf{C}$

Proof $\mathsf{A} \sqsubseteq_{DT} \mathsf{C} \stackrel{\text{def}}{=} \forall_{[\_]_a \in [\_]} . Obs([\mathsf{C}]_a) \subseteq Obs([\mathsf{A}]_a)$

1. $\forall_{[\_]_a \in [\_]} . Obs([\mathsf{C}]_a) \subseteq Obs([\mathsf{A}]_a) \Leftarrow \mathsf{A} \sqsubseteq^s_F \mathsf{C}$:

Let $[\_]_a \stackrel{\text{def}}{=} (\_ \|_{\alpha(C)} \rho)\delta_{Act}$ and $\hat{\rho} \stackrel{\text{def}}{=} \overline{\rho} \restriction \alpha(\mathsf{C})$ and $o \in Obs([\mathsf{C}]_a)$.

If $\hat{\rho} = o$ then $o \in Tr(\mathsf{C})$ and from $\mathsf{A} \sqsubseteq^s_F \mathsf{C}$ we have $o \in Tr(\mathsf{C}) \Rightarrow o \in Tr(\mathsf{A})$. hence we know that $o \in Obs([\mathsf{A}]_a)$

else if $\hat{\rho} \restriction_n = o$ then $\langle \hat{\rho} \restriction_n, \{\hat{\rho} \mid_{n+1}\}\rangle \in Ref_s(\mathsf{C})$.

hence $\langle \rho \restriction_n, \{\hat{\rho} \restriction_{n+1}\} \rangle \in Ref_s(\mathsf{A})$ and $o \in Obs([\mathsf{A}]_a)$.

2. $\mathsf{A} \sqsubseteq_{DT} \mathsf{C} \Rightarrow \mathsf{A} \sqsubseteq_F^s \mathsf{C}$: If $\langle \rho, \{a\} \rangle \in Ref_s(\mathsf{C})$ then $\rho \in Obs([\mathsf{C}]_{\rho a})$ and $\rho \in Obs([\mathsf{A}]_{\rho a})$ so $\langle \rho, \{a\} \rangle \in Ref_s(\mathsf{A})$ •

Lemma 6 $\mathsf{A} \sqsubseteq_{DT^A} \mathsf{C} \Leftrightarrow \mathsf{A} \sqsubseteq_{Fext}^s \mathsf{C}$

Proof The difference between this and Lemma 5 is that both sides of the equivalence are restricted to traces of A. Hence if $\rho \in Tr(\mathsf{A})$ the above proof holds and if $\rho \notin Tr(\mathsf{A})$ there is nothing to show. •

Comparing Z refinement on relational semantics $\sqsubseteq_{Zg}$ and operational semantics $\sqsubseteq_{DTg}$. In [4] they establish the following:

Lemma 7 $\mathsf{A} \sqsubseteq_{Zg} \mathsf{C} \Leftrightarrow \mathsf{A} \sqsubseteq_F^s \mathsf{C}$

Proof [4] •

Lemma 8 $\forall_\rho . [\![[\mathsf{C}]_\rho]\!]_{Zg} \subseteq [\![[\mathsf{A}]_\rho]\!]_{Zg} \Leftrightarrow Obs([\mathsf{C}]_a) \subseteq Obs([\mathsf{A}]_a)$

Proof In context ρ, *final* from [4] returns n blanks when n operations have terminated. Clearly this is true if and only if $\rho \restriction_n$ will have been observed in our formalization. •

Lemma 9 $\mathsf{A} \sqsubseteq_{Zg} \mathsf{C} \Leftrightarrow \mathsf{A} \sqsubseteq_{DTg} \mathsf{C}$.

Proof From Lemma 5 and Lemma 7 or Lemma 8 and definitions. •

Because of the very close relation between $\sqsubseteq_{DTg}$ and $\sqsubseteq_{Zg}$ (see Lemma 8) we know how to amend the definition of $\sqsubseteq_{Zg}$ so as to permit feature addition with the guarded semantics.

$$\mathsf{A} \sqsubseteq_{Zg}^A \mathsf{C} \stackrel{\text{def}}{=} \forall_{[\_]_\rho \in [\_]_A} . [\![[\mathsf{C}]_\rho]\!]_Z \subseteq [\![[\mathsf{A}]_\rho]\!]_Z$$

¿From Lemma 8 and definitions we can conclude $\mathsf{A} \sqsubseteq_{Zg}^A \mathsf{C} \Leftrightarrow \mathsf{A} \sqsubseteq_{DTg^A} \mathsf{C}$.

LOTOS's ext refinement also permits $\mathsf{Opb} \sqsubseteq_{ext} \mathsf{Tpb}$ and $\mathsf{Opb} \sqsubseteq_{ext} \mathsf{Opbdel}$, but our definition is less restricting in two ways: 1—it permits pruning, see Figure 1; and 2—it is characterized by singleton failure semantics, not failure semantics.

7 Processes in sequential branching contexts

Processes in general can be placed in either branching or concurrent contexts. We are going to consider a only sequential branching contexts.

A process can prevent a context from starting to execute an operation (action), whereas ADTs cannot prevent a context (program) from calling an operation, but it may be that the called operation will not terminate.

Because processes can be placed in more contexts than ADTs, we should expect process refinement to be different from ADT refinement (Test in Figure 7 is not an ADT context).

Consequently, we find a two place buffer to be a refinement of a one place buffer and, further, our definition of refinement, on processes, is going to be different from $\sqsubseteq_{Fext}$.

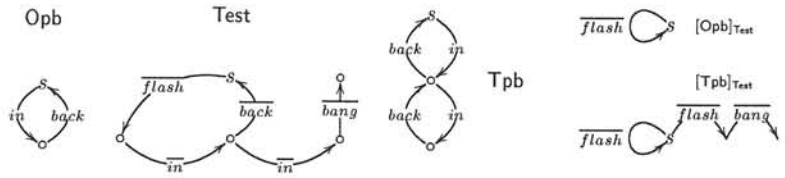


Fig. 7. Opb $\sqsubseteq_{ext}$ Tpb but $Obs([\mathsf{Tpb}]_{\mathsf{Test}}) \not\subseteq Obs([\mathsf{Opb}]_{\mathsf{Test}})$

Refinement of Process in sequential contexts on lts In order to apply our approach from Section 4 we need to define the contexts in which processes can be placed.

$$[\_] \stackrel{\text{def}}{=} \{(\_ \parallel_{\alpha(\_)} p)\delta_{Act} \mid p \in \overline{lts}\}$$

We use these contexts throughout Section 7 and apply Definition 5 to compute "contexts where A is expected" to be:

$$[\_]^A = \{(\_ \parallel_{\alpha(\mathsf{A})} p)\delta_{Act} \mid \overline{Tr^c(p)} \cap Tr(\mathsf{A}) \neq \varnothing\}$$

It can easily be seen (see Lemma 10) that we could have used $\{(\_ \parallel_{\alpha(\mathsf{A})} p)\delta_{Act} \mid true\}$ without affecting the refinement relation. This means that for processes, *i.e.* branching contexts, Assumption 3 has no effect on the definition of refinement. This is not very surprising as Assumption 3 originated from the state-based intuition of refinement and its relevance to process refinement is tenuous.

$$\mathsf{A} \sqsubseteq_P \mathsf{C} \stackrel{\text{def}}{=} \mathsf{A} \sqsubseteq \mathsf{C} \text{ and } \mathsf{A} \sqsubseteq_{P^A} \mathsf{C} \stackrel{\text{def}}{=} \mathsf{A} \sqsubseteq^A \mathsf{C}$$

Assuming the Smyth powerdomain then from Lemma 2 and Lemma 4 we have:

$$\mathsf{A} \sqsubseteq_P \mathsf{C} \Leftrightarrow \forall_{[\_]_a \in [\_]} . Obs([\mathsf{C}]_a) \subseteq Obs([\mathsf{A}]_a).$$
$$\mathsf{A} \sqsubseteq_{P^A} \mathsf{C} \Leftrightarrow \forall_{[\_]_a \in [\_]_A} . Obs([\mathsf{C}]_a) \subseteq Obs([\mathsf{A}]_a).$$

Lemma 10 $[\_]^A = \{(\_ \parallel_{\alpha(\mathsf{A})} p)\delta_{Act} \mid \overline{Tr^c(p)} \cap Tr(\mathsf{A}) \neq \varnothing\}$ *is a core set of contexts for* $[\_]^{A^+} = \{(\_ \parallel_{\alpha(\mathsf{A})} p)\delta_{Act} \mid true\}$

Proof We need to show that if $[\_]_x \in [\_]^{A^+} - [\_]^A$ then we can infer what $Obs([\_]_x)$ would be from the observations of $[\_]^A$. To do this we build a context $x + a$ where "+" is choice[3]. $s_{\mathsf{x+a}} \stackrel{\text{def}}{=} s_{\mathsf{x}}$ and $Tran_{\mathsf{x+a}} \stackrel{\text{def}}{=} Tran_{\mathsf{x}} \cup \{(s_{\mathsf{x}}, \mathsf{a}, x) \mid \{(s_{\mathsf{a}}, \mathsf{a}, x)\} = Tran_{\mathsf{a}}\}$.

Let $a \in \pi(\mathsf{A})$ then $a \notin Obs([\mathsf{A}]_x)$, $Obs([\mathsf{A}]_a) \stackrel{\text{def}}{=} \{a\}$ and $[\mathsf{A}]_a \in [\mathsf{A}]^A \wedge [\mathsf{A}]_{x+a} \in [\mathsf{A}]^A$. As $Obs([\mathsf{A}]_{x+a}) = Obs([\mathsf{A}]_x) \cup Obs([\mathsf{A}]_a)$ and $Obs([\mathsf{A}]_x) \cap Obs([\mathsf{A}]_a) = \varnothing$ we know $Obs([\mathsf{A}]_x) = Obs([\mathsf{A}]_{x+a}) - Obs([\mathsf{A}]_a)$. •

Lemma 11 $\mathsf{A} \sqsubseteq_P \mathsf{C} \Leftrightarrow \mathsf{A} \sqsubseteq_{Ftest} C\mathsf{C}$

[3] Althought we could have used choice from ACP we, for brevity of definition, use that from [24]

Proof $\mathsf{A} \sqsubseteq_P \mathsf{C} \Rightarrow \mathsf{A} \sqsubseteq_{Ftest} \mathsf{C}$ follows from the observation that Hennessy's "essential tests" [11] are all contained in our process contexts $[\_]$.

$\mathsf{A} \sqsubseteq_P \mathsf{C} \Leftarrow \mathsf{A} \sqsubseteq_{Ftest} \mathsf{C}$ follows directly from folklore monotonicity of failure refinement with respect to the basic process operators ([25, 11]).

•

Lemma 12 $\mathsf{A} \sqsubseteq_{PA} \mathsf{C} \Rightarrow \mathsf{A} \sqsubseteq_{Fpro} \mathsf{C}$

Proof Let $\overline{\rho;\ (\Sigma X)}$ be the smallest context such that $s \xrightarrow{\overline{\rho}} s^1 \wedge \forall_{x \in X}.s^1 \xrightarrow{\overline{x}}$ (or use ACP's sequential compositions ";" and choice "+").

We need to prove 1. $\mathsf{A} \sqsubseteq \mathsf{C} \Rightarrow \forall \sigma \in Tr(\mathsf{A}).Ref(\mathsf{C}, \sigma) \subseteq Ref(\mathsf{A}, \sigma)$

Using contexts $\rho;\ (\Sigma X)$ where $\rho \in Tr(\mathsf{C}\delta_{New}) \wedge X \subseteq \alpha(\mathsf{C}\delta_{New})$ we can see that $\rho \in Obs(\mathsf{C}) \Rightarrow \rho \in Obs(\mathsf{A})$ hence $\forall \sigma \in Tr(\mathsf{C}\delta_{New}).Ref(\mathsf{C}\delta_{New}, \sigma) \subseteq Ref(\mathsf{A}, \sigma)$.

and 2. $\mathsf{A} \sqsubseteq \mathsf{C} \Rightarrow Tr(\mathsf{A}) = Tr(\mathsf{C}\delta_{New})$

As failure refinement implies trace refinement ([11, 6]) from Lemma 11 above we have: $\mathsf{A} \sqsubseteq \mathsf{C} \Rightarrow Tr(\mathsf{A}) \subseteq Tr(\mathsf{C}\delta_{New})$

Finally we prove: $\mathsf{A} \sqsubseteq \mathsf{C} \Rightarrow Tr(\mathsf{A}) \supseteq Tr(\mathsf{C}\delta_{New})$

Use contexts ρ where $\rho \in Tr(\mathsf{C}\delta_{New})$ $\rho \in Obs([\mathsf{C}]_\rho)$ and by assumption $\rho \in Obs([\mathsf{A}]_\rho)$ hence $\rho \in Tr(\mathsf{A})$ •

Lemma 13 $\mathsf{A} \sqsubseteq_{Fpro} \mathsf{C} \Rightarrow \mathsf{A} \sqsubseteq_{Pa} \mathsf{C}$

Proof

From Lemma 1 $\mathsf{A} \sqsubseteq_{Fpro} \mathsf{C} \Leftrightarrow \mathsf{A} \sqsubseteq_{Ftest} \mathsf{C}\delta_{New}$.

From Lemma 11 $\mathsf{A} \sqsubseteq_{Ftest} \mathsf{C}\delta_{New} \Leftrightarrow \mathsf{A} \sqsubseteq_P \mathsf{C}\delta_{New}$.

From Lemma 10 the only differance between the contexts use in the definition of $\sqsubseteq_P$ and those in the definition of $\sqsubseteq_{Pa}$ is that one can synchronise with actions that do not appear in either A or $\mathsf{C}\delta_{New}$. Consequently $\mathsf{A} \sqsubseteq_P \mathsf{C}\delta_{New} \Leftrightarrow \mathsf{A} \sqsubseteq_{Pa} \mathsf{C}\delta_{New}$

As $Obs([\mathsf{C}\delta_{New}]^A_\rho) \stackrel{\text{def}}{=} Obs((\mathsf{C}\delta_{New} \parallel_{\alpha(\mathsf{A})} p)\delta_{Act})$. Because $New \cap \alpha(\mathsf{A}) = \varnothing$ we know: $Obs((\mathsf{C}\delta_{New} \parallel_{\alpha(\mathsf{A})} p)\delta_{Act}) = Obs((\mathsf{C} \parallel_{\alpha(\mathsf{A})} p)\delta_{Act})$

Hence from definition $\mathsf{A} \sqsubseteq_{Pa} \mathsf{C}\delta_{New} \Leftrightarrow \mathsf{A} \sqsubseteq_{Pa} \mathsf{C}$ •

8 Objects (without sharing)

There is no single definition of what constitutes an object nor a single definition of what constitutes the contexts in which they can be placed. In a similar situation to that for non-deterministic data types, the definition of data refinement is parameterised on the set of contexts in which an object can be placed ([26]), *i.e.* programs that use the object. In that case there is one set of contexts per definition of refinement, unlike here where the set of contexts depends upon the individual thing A that is being refined.

As with the work on non-deterministic data types [26], what constitutes refinement of objects depends upon the language used to define them. For example

[21] considers refinement of objects defined using a version of Object-Z that has a *pre* 'command' which when applied to an operation returns its boolean guard. Consequently [21] allows more contexts than we have and his object semantics is the more discriminating ready trace [9].

Definitions for refinement of shared objects such as [8, 13] start by defining models for non-shared objects. The non-shared objects are modelled as processes. That is to say they do not restrict the contexts to being programs. In [8] the resulting definition of refinement of the non-shared objects is the same as our definition of refinement of processes.

If we assume that non-shared objects like ADTs can only be called by *programs*, *i.e.* traces of operations, then it would be reasonable to apply our definition of ADT refinement on these non-shared objects.

9 Refinement in the literature

For a survey on the unification of Z with process algebras see [27].

Initially it was thought that Z data refinement and failure refinement were the same [28]. Then this was shown not be the case, *i.e.* that $\sqsubseteq_Z \neq \sqsubseteq_{Ftest}$, and singleton failures refinement $\sqsubseteq^s_{Ftest}$ was defined ([4]) and shown equivalent to $\sqsubseteq_Z$.

Data refinement, described in [17, page 241], uses the restricted contexts "every program of $P(\mathsf{A})$". Because the semantics mean an operation is undefined outside of its precondition, this restriction is redundant. Data refinement, described in [18], uses all contexts (programs) and the semantics says an operation is not undefined outside of its precondition. Of the two definitions only refinement of the *undefined* semantics [17] permits feature addition.

For us the important insight of [4] was that data types could only be placed in sequential contexts. Here we have extended the work of [4] by considering "feature addition" and shown differences between refinement with feature addition of processes (*i.e.* branching contexts) and ADTs (*i.e.* sequential contexts).

When we are considering the refinement not of individual operations but of the whole ADT/process then nondeterminism may still be unwanted, *i.e.* we may wish it to be designed away, yet the pruning of traces may not be desirable. Definitions of refinement that reflect this are testing refinement [11] and LOTOS's extension refinement [7]. These forms of refinement reduce nondeterminism and the set of contexts in which the process will not terminate.

The definition of refinement found in [5] does not restrict the "contexts where A is expected" to $\alpha(A)$, consequently a feature (action) addition like $\mathsf{a} + \mathsf{b} \not\sqsubseteq \mathsf{a} + \mathsf{b} + \mathsf{c}$ is not a refinement, whereas in ours and the definition in [14] it is.

In [5] they say "we'd like the two place buffer to be a subtype of a one place buffer" and they place their buffers in branching contexts. Like them we find this problematic. The solution suggested in [5] is that actions that are "not offered" are given an undefined semantics not a guarded semantics. Here we define data types that cannot be placed in branching contexts and for which a two place buffer is a refinement of a one place buffer. We give a separate definition of

processes that can be placed in branching contexts and for which a two place buffer is **not** a refinement of a one place buffer.

Nierstasz [14] defines subtyping in an equivalent way to extension refinement and obtains the result: Sequential clients (contexts), satisfied by an abstract object, will be satisfied by a subtype (refinement) of it. For concurrent (branching) contexts this result does not follow, whereas we restrict the contexts in which our processes can be placed.

In [8] several refinement definitions are given on the denotational semantics. **Weak** subtyping, for not-shared objects, is equivalent to our $\sqsubseteq_{PA}$. But if we assume that not-shared objects can only be placed in sequential contexts then we would choose $\sqsubseteq_{DTA}=\sqsubseteq^{s}_{Fext}$ as our definition of behavioural subtyping. Their other definitions **safe**, **optimal** and **optimistic** subtyping are for shared objects and extend **weak** subtyping by treating actions, of the sharing object, as (τ) internal actions(other versions can be found in [18, 13].

10 Conclusion

We have provided a common framework in which to compare some of the many definitions of refinement/subtyping.

In doing so we have come across a new definition of refinement of abstract data types $A \sqsubseteq^{s}_{Fext} C$ that is slightly weaker than LOTOS's extension. This definition of refinement can also be seen as a slight weakening of $A \sqsubseteq^{s}_{Ftest} C$ singleton failure semantics [4] which permits feature addition.

By formulating refinement of A in terms of improving the observation of "A in a context" and defining improvement as a preorder on the observations, we have compared the refinement of data types and processes. We then related our definitions of refinement to definitions found in the literature and showed some are minor improvements.

Finally we turn to our future agenda:

Value passing As data type operations cannot select the value returned, a program that calls pop on a stack must accept the value returned. A program cannot call pop(1) so that it is executed only if the value on the top of the stack is 1. Consequently, because we use the semantic mapping from Z to lts of [20–22], we would need a singleton version of the semantics in [29] rather than singleton failure of [4].

Mix guards with undefined Within the framework we have used this could be introduced in the semantic mapping from Z to lts and without changing our definition of refinement on lts.

Egli-Milner powerdomain The definitions of data refinement in the literature all correspond to the use of the Smyth powerdomain. We are currently considering situations where the restrictions of the Egli-Milner powerdomain might be of advantage.

Shared Objects There are many definitions of a shared object and its contexts. There are also many definitions of refinement of shared objects. It would be

interesting to see if our approach could relate definitions of refinement to the contexts in which they may be placed.

Acknowledgements Thanks to Greg Reeve, Moshe Deutsch, Mark Utting and Doug Goldson for many helpful dsicussions. Thanks to new Zealand Government's Foundation fro Reasearch, Science and Technology for funding to make this research possible.

References

1. Standard, I.: Open distributed processing - reference model. Technical Report ISO/IEC 10746-1(E), International Electro (1998)
2. Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In Akşit, M., Matsuoka, S., eds.: Proceedings European Conference on Object-Oriented Programming. Volume 1241. Springer-Verlag, Berlin, Heidelberg, and New York (1997) 220–242
3. Derrick, J., Bowman, H., Boiten, E., Steen, M.: Comparing LOTOS and Z refinement relations. In: FORTE/PSTV'96, Kaiserslautern, Germany, Chapman & Hall (1996) 501–516
4. Bolton, C., Davies, J.: A singleton failures semantics for Communicating Sequential Processes. Research Report PRG-RR-01-11, Oxford University Computing Laboratory (2001)
5. Bowman, H., Briscoe-Smith, C., Derrick, J., Strulo, B.: On behavioural subtyping in lotos (1997)
6. Hoare, C.: Communicating Sequential Processes. Prentice Hall International Series in Computer Science (1985)
7. Brinksma, E., Scollo, G.: Formal notions of implementation and conformance in lotos. Technical Report INF-86-13, Twente University of Technology, Department of Informatics, Enschede, The Netherlands (1986)
8. Fischer, C., Wehrheim, H.: Behavioural subtyping relations for object-oriented formalisms. Lecture Notes in Computer Science **1816** (2000) 469–??
9. Baeten, J.C.M., Weijland, W.P.: Process Algebra. Cambridge Tracts in Theoretical Computer Science 18 (1990)
10. Spivey, J.M.: The Z notation: A reference manual. Prentice Hall (1989)
11. Hennessy, M.: Algebraic Theory of Processes. The MIT Press (1988)
12. Bolton, C., Davies, J.: A comparision of refinement orderings and their associated simulation rules. Electronic Notes in Theoretical Computer Science **70** (2002) 14
13. Basten, T., van der Aalst, W.M.P.: Inheritance of behavior. JLAP **47** (2001) 47–145
14. Nierstrasz, O.: Regular types for active objects. In Nierstrasz, O., Tsichritzis, D., eds.: Object-Oriented Software Composition. Prentice-Hall (1995) 99–121
15. Liskov, B., Wing, J.: Family values: a behavioral notion of subtyping. Technical Report MIT/LCS/TR-562b, Massachusetts Institute of Technology (1993)
16. He, J., Hoare, C., Sanders, J.: Data refinement refined. ESOP 86 Lecture Notes in Computer Science **213** (1986) 187–196
17. Woodcock, J., Davies, J.: Using Z: Specification, Refinement and Proof. Prentice Hall (1996)

18. Derrick, J., Boiten, E.: Refinement in Z and Object-Z: Foundations and Advanced Applications. Formal Approaches to Computing and Information Technology. Springer (2001)
19. Deutsch, M., Henson, M., Reeves, S.: Results on formal stepwise design in z. In: Proceedings of APSEC2002, IEEE Computer Society (2002)
20. Smith, G.: A semantic integration of object-Z and CSP for the specification of concurrent systems. In Fitzgerald, J., Jones, C.B., Lucas, P., eds.: FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997). Volume 1313., Springer-Verlag (1997) 62–81
21. Smith, G.: A fully abstract semantics of classes for object-z. Formal Aspects of Computing **7** (1995) 289–313
22. Derrick, J., Boiten, E., Bowman, H., Steen, M.: Specifying and Refining Internal Operations in Z. Formal Aspects of Computing **10** (1998) 125–159
23. Boiten, E., Derrick, J.: Unifying concurrent and relational refinement. In Derrick, J., Boiten, E., Woodcock, J., von Wright, J., eds.: REFINE 02: The BCS FACS Refinement Workshop. Volume 70(3) of Electronic Notes in Theoretical Computer Science., Elsevier Science Publishers (2002) 38
24. Winskel, G., Nielsen, M.: Models for concurrency. Technical Report DAIMI PB 429, Computer Science Dept. Aarhus Universty (1992)
25. Roscoe, A.: The Theory and Practice of Concurrency. Prentice Hall International Series in Computer Science (1997)
26. Nipkow, T.: Non-deterministic data types: Models and implementations. Acta Informatica **22** (1986) 629–661
27. Fischer, C.: How to combine Z with a process algebra. In Bowen, J.P., Fett, A., Hinchey, M.G., eds.: 11th Int. Conf. ZUM'98: the Z Formal Specification Notation. LNCS 1492, Springer-Verlag (1998) 5–23
28. Bolton, C., Davies, J., Woodcock, J.: On the refinement and simulation of data types and processes. In Arak, K., Galloway, A., Taguchi, K., eds.: Proceedings of IFM '99. (1999) 273–292
29. Hennessy, M., Ingolfsdottir, A.: A theory of communicating processes with value passing. Information and computation **107** (1993) 202–236