Supporting ODP - Translating LOTOS to Z

John Derrick, Eerke Boiten, Howard Bowman and Maarten Steen
Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK.
(Email: jd1@ukc.ac.uk.)

Abstract

This paper describes a translation of full LOTOS into Z. A common semantic model is defined and the translation is proved correct with respect to the semantics.

The motivation for such a translation is the use of multiple viewpoints for specifying complex systems defined by the reference model of the Open Distributed Processing (ODP) standardization initiative.

Key words: Open Distributed Processing; Z; LOTOS; Consistency.

1 Introduction

The aim of this paper is to support the use of FDTs within distributed system design by providing a translation between full LOTOS and Z.

An important example of open object-based distributed systems is the Open Distributed Processing (ODP) Reference Model. The ODP standardization initiative is a natural progression from OSI, broadening the target of standardization from the point of interconnection to the end-to-end system behaviour. The objective of ODP [12] is to enable the construction of distributed systems in a multi-vendor environment through the provision of a general architectural framework that such systems must conform to. One of the cornerstones of this framework is a model of multiple viewpoints which enables different participants each to observe a system from a suitable perspective and at a suitable level of abstraction. There are five separate viewpoints presented by the ODP model: Enterprise, Information, Computational, Engineering and Technology. Requirements and specifications of an ODP system can be made from any of these viewpoints.

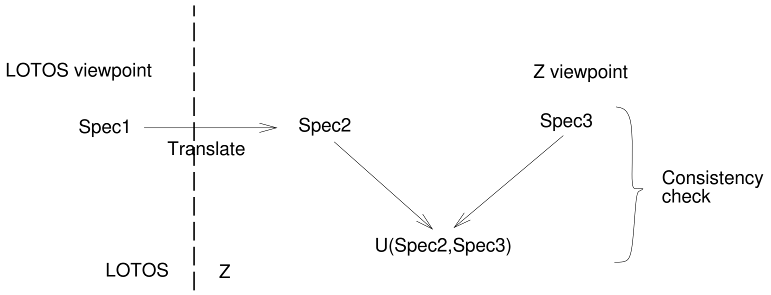
The ODP reference model (RM-ODP) recognises the need for formalism, with Part 4 of the RM-ODP defining an architectural semantics which describes the application of formal description techniques (FDTs) to the specification of ODP systems. Of the available FDTs, Z is likely to be used for at least the information, and possibly other,

[0]This work was partially funded by British Telecom Labs., Martlesham, Ipswich, U.K. and partially by the U.K. Engineering and Physical Sciences Research Council under grant number GR/K13035.

viewpoints (the ODP Trader specification is being written using Z for the information viewpoint), whilst LOTOS is a strong candidate for use in the computational viewpoint.

One of the consequences of adopting a multiple viewpoint approach to specification is that descriptions of the same or related entities can appear in different viewpoints and must co-exist. *Consistency* of specifications across viewpoints thus becomes a central issue. Similar consistency properties arise outside ODP, see for example [9]. We have shown how consistency checking may be performed within a single FDT, [3, 6, 7, 18], however, the real challenge lies in checking for consistency across language boundaries, and this requires translation between FDTs.

The strategy we envisage to check the consistency of one ODP viewpoint written in Z with another written in LOTOS is as follows. First translate the LOTOS specification to an observationally equivalent one in Z (thus preserving meaning), then use the mechanisms defined in [6, 7, 1] to check the consistency of the two viewpoints now both expressed in Z. (Note that this does not assume the two viewpoints are written at the same level of abstraction.)



The work described here makes a first step towards a solution, by defining a translation of full LOTOS into Z. A common semantic model is defined and the translation is proved correct with respect to this semantics. Section 2 explains the model. Sections 3 and 4 then provide a semantics for LOTOS and Z in this model. Section 5 defines the LOTOS to Z translation, which is verified in section 6.

2 Definitions

In [19] extended transition systems (ETS) are used to define a semantics for full LOTOS, and we will use them as our common semantic model. An ETS combines a labelled transition system with an abstract data definition.

2.1 Extended Transition Systems

An extended transition system provides a semantic model for the data in addition to the control behaviour of a system. Given a signature Σ, and a set of variables V, the set of terms over Σ and V is denoted $T_\Sigma(V)$ (we assume it includes all boolean terms).

Definition 1 *An extended transition system is a 6-tuple $ETS = \langle S, E, A, R, s_0, f_0 \rangle$ where S is a set of states of the ETS; $E \subseteq S \times Id$ is a finite set of extensions on ETS, and*

Id a finite set of identifiers; A is a set of actions on ETS (see below); R is a set of transition relations on ETS (see below); s_0 *is the initial state of the system;* f_0 *is the initial assignment of the variables.*

An ETS may be extended by substitution with another ETS for every extension in the set E, thus the identifiers in an extension $\langle s, P\rangle$ act as temporary placeholders representing that at state s the ETS behaves like specification P. The translation from LOTOS to ETS uses these extensions to describe process instantiation and recursion whilst generating a final extension free ETS.

Definition 2 *Let G be a set of gates over which an extended transition system can communicate. Actions are elements of G with a finite list of attributes: either a value or variable declaration of the form $!e$, or a variable declaration of the form $?v : t$. Let I be a set of internal (unobservable) actions. Elements of I are denoted i. The set of actions of an ETS is the set*

$$A = \{g?v_1 : t_1 \ldots ?v_m : t_m \; !e_1 \ldots !e_n \mid g \in G \cup I, e_i \in T_\Sigma(V), v_j \in V\}$$

The function name(a) returns the gate name in action a (either observable or internal).

Definition 3 *Each element of the set of transition relations R is a 5-tuple $r = \langle a, s, s', p, f\rangle$ where a is an enabling action; $s, s' \in S$ are states of the ETS (not necessarily distinct); $p \in T_\Sigma(V)$ is an enabling predicate associated with r; $f : V \to T_\Sigma(V)$ is an action function associated with r.*

The intuitive meaning of a transition relations r is that if the ETS is in state s and the enabling action a is offered, then the enabling predicate is evaluated on the current assignment of variables. When p is true, the ETS will go into the new state s' and the variables are updated by the action function f.

In order that we may use extended transition systems to provide an operational semantics for Z, we have relaxed the condition from [19] that the set of transition relations be finite, and we have extended the attributes of a gate to include variable (as well as value) declarations of the form $!v$.

2.2 Observational Equivalence in Extended Transition Systems

The use of observational equivalence and bisimulation lie at the heart of process algebras [2, 13], allowing equivalence between specifications to be asserted on the basis of observed behaviour. However, it has traditionally suffered from the disadvantage that for value-passing processes, where the values are taken from an infinite data-space, in order to check for equivalence infinite transition graphs must be compared. The solution to infinite transition graphs is to use symbolic bisimulations as the means to assert equivalence [10, 19].

In [19], Chanson defines a relation $\xrightarrow{\mu}$ between states as the obvious extension of the transition relation to action sequences where each observable action in μ includes output and/or input primitives with zero or more actual parameters. The induced equivalence corresponds to the early bisimulation of [10].

Definition 4 *Let* $ETS = \langle S, E, A, R, s_0, f_0 \rangle$ *be an extended transition system.*

(a) Let $s, s' \in S$*;* $a_1, \ldots, a_n \in A$*; and* μ *denote a string of actions* $a_1, \ldots, a_n$*. Each observable action includes output and/or input primitives with zero or more actual parameters. The relation* $\xrightarrow{\mu}$ *is defined by:* $s \xrightarrow{\mu} s'$ *iff there exists* $s_0, \ldots, s_n \in S$ *such that* $s = s_0 \xrightarrow{a_1} s_1 \ldots s_{n-1} \xrightarrow{a_n} s_n = s'$*. Note that* $s \xrightarrow{\epsilon} s$ *for all states* s*.*

(b) Let $s, s' \in S$*;* $g_1, \ldots, g_n \in A - I$*; and* μ *denote a string of observable actions* $g_1, \ldots, g_n$*, and* i^k *a sequence of* k *internal actions. The observable sequence relation* $\stackrel{\mu}{\Longrightarrow}$ *is defined by:* $s \stackrel{\mu}{\Longrightarrow} s'$ *iff there exists a sequence* $\alpha = i^{k_0} g_1 i^{k_1} g_2 \ldots g_n i^{k_n}$ *of actions such that* $s \xrightarrow{\alpha} s'$*. Note that* $s \stackrel{\epsilon}{\Longrightarrow} s'$ *whenever* $s \xrightarrow{i^k} s'$*, and that* $s \stackrel{\epsilon}{\Longrightarrow} s$ *for all states* s*.*

We can now define weak bisimulation for extended transition systems.

Definition 5 *Let* $ETS_1 = \langle S_1, E_1, A_1, R_1, s_{01}, f_{01} \rangle$ *and* $ETS_2 = \langle S_2, E_2, A_2, R_2, s_{02}, f_{02} \rangle$ *be extended transition systems, and* $(names(A_1) - I) = (names(A_2) - I)$*, ie the sets of observable gate names in* ETS_1 *and* ETS_2 *are equal. A relation* $\mathcal{R} \subseteq S_1 \times S_2$ *is a weak bisimulation relation if for all pairs* $(s_1, s_2) \in \mathcal{R}$ *and for any string* μ *of observable actions:*

a *whenever* $s_1 \stackrel{\mu}{\Longrightarrow} s_1'$*, there exists* $s_2' \in S_2$ *such that* $s_2 \stackrel{\mu}{\Longrightarrow} s_2'$ *and* $(s_1', s_2') \in \mathcal{R}$*; and*

b *whenever* $s_2 \stackrel{\mu}{\Longrightarrow} s_2'$*, there exists* $s_1' \in S_1$ *such that* $s_1 \stackrel{\mu}{\Longrightarrow} s_1'$ *and* $(s_1', s_2') \in \mathcal{R}$*.*

Definition 6 *Let* $ETS_1 = \langle S_1, E_1, A_1, R_1, s_{01}, f_{01} \rangle$ *and* $ETS_2 = \langle S_2, E_2, A_2, R_2, s_{02}, f_{02} \rangle$ *be extended transition systems. Two states* s_1 *and* s_2 *are weak bisimulation equivalent (written* $s_1 \approx s_2$*), if there exists a weak bisimulation relation* $\mathcal{R} \subseteq S_1 \times S_2$ *such that* $(s_1, s_2) \in \mathcal{R}$*.* ETS_1 *and* ETS_2 *are weak bisimulation equivalent (written* $ETS_1 \approx ETS_2$*), if there exists a weak bisimulation relation* $\mathcal{R} \subseteq S_1 \times S_2$ *such that* $(s_{01}, s_{02}) \in \mathcal{R}$*.*

We use the term observationally equivalent as a synonym for weak bisimulation equivalence.

3 Translation from LOTOS to ETS

A LOTOS specification of a system defines the temporal relationships among the interactions that constitute the externally observable behaviour of the system [2]. A specification consists of two parts: the *behaviour expression* describes the process behaviour and its interaction with the environment whilst the *abstract data type* (ADT) describes the data structures and value expressions.

The translation from LOTOS to ETS given in [19] is based on the standard transition derivation system defined in [11] extended to cover data representation and value passing in full LOTOS. The algorithm generates an extended transition system with a finite set of transition relations.

The transition rules work bottom-up beginning with the LOTOS terminals. A translation algorithm is then developed using the transition rules (full details are given in [19]). The transition rules generate a new extended transition system *ETS* for a behaviour *B* which is generated from B' and B'' by application of LOTOS operators. Let B' and B'' be LOTOS behaviour expressions. Assume there exists an extended transition system $ETS' = \langle S', E', A', R', s'_0, f'_0 \rangle$ associated with B', and similarly for B'', where S' and S'' are disjoint. As an example, the transition rules for *stop*, choice and action prefix are:

1. For inaction, $B = stop$, we have $ETS = \langle \{s_0\}, \varnothing, \varnothing, \varnothing, s_0, \epsilon \rangle$, where we use ϵ to stand for the null function.

2. For choice, $B = B'[]B''$, let $R'(s'_0)$ denote the transitions enabled from s'_0, then

$$ETS = \langle \{s_0\} \cup (S' - \{s'_0\}) \cup (S'' - \{s''_0\}), E, A' \cup A'', R, s_0, \epsilon \rangle$$

where $E = \{\langle s_0, X \rangle \mid X \in (E'(s'_0) \cup E''(s''_0))\} \cup (E' \setminus E'(s'_0)) \cup (E'' \setminus E''(s''_0))$ and $R = \{\langle a, s_0, s, p, f \rangle \mid \langle a, s, p, f \rangle \in (R'(s'_0)) \cup (R''(s''_0))\} \cup (R' \setminus R'(s'_0)) \cup (R'' \setminus R''(s''_0))$.

3. For action prefix of the form $B = gd_1 \ldots d_n[BE];\ B'$, we have

$$ETS = \langle \{s_0\} \cup S', E', A' \cup \{gd_1 \ldots d_n\}, R, s_0, \epsilon \rangle$$

where $R = \{\langle gd_1 \ldots d_n, s_0, s'_0, BE, \epsilon \rangle\} \cup R'$.

4 An ETS semantics for Z

The Z specification language [17] has gained acceptance as one of the viewpoint specification languages for ODP, particularly for the information viewpoint. Because ODP is object-based, there is a need to provide object-oriented capabilities in FDTs used within ODP. ZEST [4] is an extension to Z to support specification in an object-oriented style, developed by British Telecom specifically to support distributed system specification.

ZEST does not increase the expressive power of Z, and a flattening to Z is provided. What ZEST provides is structuring at a suitable level of abstraction by associating individual operations with one state schema. A *class* is a state schema together with its associated operations and attributes. A class is a template for objects: each object of the class has a state which conforms to the class state schema, and is subject to state transitions which conform to the class operations. In many ways ZEST is similar to Object-Z [8], although the latter does not provide a flattening to Z.

We use ZEST here to provide structuring at the right level and because it facilitates a process algebraic view of Z based specification. Since a flattening to Z is provided, the work we derive here can be applied equally to Z itself. The standard semantics for Z is denotational [16]. Consideration of object-oriented issues, however, leads naturally to viewing objects as processes and hence to an *observational* view of the semantics of the specification. Z state changes occur by application of Z operation schemas, thus an observational view regards invocation of a Z operation as a transition in a labelled transition system (LTS).

We will provide an ETS for each ZEST specification, in such a way that a LOTOS specification and its ZEST translation are observationally equivalent in the ETS semantics. We are not alone in providing an LTS interpretation to object oriented versions of Z, [5, 15]. However, beyond describing such an interpretation, little work has been done on its exploitation. In [5] and [15] the basic idea used is that labels in the transition system are operation schema names together with any input/output values. A transition is added whenever an operation is applicable at a node, which represents a particular binding of state variables. We differ from previous work in the labels we attach to transitions in the system. Instead of using *values* as labels, we use *variables* and *expressions* as the labels. This enables us to derive a symbolic transition system, and to represent a schema such as $[out! : \mathbb{Z} \mid out! \geq 0]$ as a single transition as opposed to an infinite choice of transitions.

An internal event will be specified either as a private operation schema [14] or by a distinguished schema operation name, eg i, as in LOTOS. This is a matter of convention rather than semantic difference, and we adopt the latter here.

The semantics of a ZEST specification is defined to be the ETS of the top level object. We assume that all inheritance has been expanded out in the given ZEST class. The set of variables in the ETS consists of all state variables defined together with all inputs and outputs declared in the operation schemas.

The ETS of a ZEST object is derived from considering the application of the last operation schema defined in the object to the ETS derived from the object excluding that schema. Unlike the LOTOS to ETS mapping which generates a finite set of transitions, the ETS we shall derive from a ZEST specification has a possibly infinite set of transitions, however, the derivation suffices for verification of our LOTOS translation. The purpose of the ETS semantics for ZEST is purely to verify the LOTOS to Z translation, so while it was necessary to generate a finite ETS from LOTOS, such considerations do not matter for the ZEST semantics. Once the ZEST semantics has been used to verify the translation in section 6, one does not need to refer to the ETS semantics for ZEST when performing the LOTOS to ZEST mapping.

The base ETS

To start, the ETS of an object with no operations is defined. Consider the ZEST object:

P

$Attributes$
$State$
$Initial\_State$

the ETS of this is given by $ETS = \langle \{s_0\}, \varnothing, \varnothing, \varnothing, s_0, f_0 \rangle$ where f_0 is the assignment of $Initial\_State$, ie the predicate. (The LOTOS translation always produces such an assignment.)

The inductive case

To calculate the effect of operations on the transition system, suppose that the ZEST object

$$\begin{array}{|l}
\hline
\multicolumn{1}{l}{P'} \\
\textit{Attributes} \\
\textit{State} \\
\textit{Initial_State} \\
\textit{Operation}_1 \\
\vdots \\
\textit{Operation}_n \\
\hline
\end{array}$$

has an associated ETS of $ETS' = \langle S', E', A', R', s_0', f_0' \rangle$. Then we calculate the ETS of

$$\begin{array}{|l}
\hline
\multicolumn{1}{l}{P} \\
\textit{Attributes} \\
\textit{State} \\
\textit{Initial_State} \\
\textit{Operation}_1 \\
\vdots \\
\textit{Operation}_n \\
A \\
\hline
\end{array}$$

(where A is an operation schema) in terms of ETS' in the following fashion.

Consider each $s' \in S'$ in turn. Given such an $s' \in S'$, we evaluate $\mathrm{pre}\, A$ at that state (ie on the current assignment of the variables). If A is not applicable, no new relation is added to R' and the ETS is not extended. If A is applicable at s', then a new transition is added to R' and the ETS is extended. We calculate the transitions as follows.

Calculating a transition from an operation schema

An operation schema A given by

$$\begin{array}{|l}
\hline
\multicolumn{1}{l}{A} \\
\Delta(\textit{state_vars}) \\
\text{Declarations} \\
\hline
\text{OpPred} \\
\hline
\end{array}$$

maps to a transition $r = \langle a, s', s, p, f \rangle$ where

1. $a = A?x_1? : t_1 \ldots ?x_n? : t_n \; !y_1! : u_1 \ldots !y_m! : u_m$ for declarations $x_1? : t_1, \ldots, x_n? : t_n, y_1! : u_1, \ldots, y_m! : u_m$ within A;
2. p is the precondition of OpPred at the current assignment of variables;
3. f gives the effect on state and output variables of performing operations A; and
4. If the effect of f on s' produces an assignment of variables that corresponds to a state $s'' \in S'$, then $s = s''$. If not (or it is undecidable), then a new state, s is added ($S = S' \cup \{s\}$).

For all states added which are not in S', the effect of the object has to be calculated on those states because an existing operation may be applicable at the new state. Therefore all the operations $Op_1, \ldots, Op_n, A$ are applied to these new states to extend the ETS further.

The result of this process is an ETS containing a (not necessarily finite) set of transition relations R. The final ETS consists of the updated set of states and transitions, together with $E = E'$, $A = A' \cup \{a\}$, $s_0 = s_0'$, $f_0 = f_0'$.

5 Translation from full LOTOS to Z

The essential idea behind the translation is to turn LOTOS processes into ZEST objects, and hence if necessary into Z. The ADT component of a LOTOS specification is translated directly into the Z type system. For the behaviour expression of a LOTOS specification, we first derive the ETS from the LOTOS, and use this to generate the Z specification. This will involve translating each LOTOS action into a ZEST operation schema with explicit pre- and post-conditions to preserve the temporal ordering.

For example, given a LOTOS process $in?x : nat;\ out!(x+2);\ stop$, this will be translated into a ZEST object which contains operation schemas with names in and out. The operation schemas have appropriate inputs and outputs to perform the value passing defined in the LOTOS process. Each operation schema includes a predicate (derived from the ETS) to ensure that it is applicable in accordance with the temporal behaviour of the LOTOS specification. Because a finite ETS is generated from any LOTOS specification (see [19]), a ZEST specification can be generated which fully describes the LOTOS correctly.

Thus we are in fact embedding part of an intermediate semantics for LOTOS within Z (to preserve the temporal ordering). The operation schemas (apart from the temporal ordering) could in fact be generated directly from the LOTOS specification without recourse to the ETS semantics.

Because we are using the ETS of a LOTOS specification, none of the original syntactic structure is preserved. All the processes are expanded out into one possible behaviour, and this generates one ZEST object. Thus, in particular, communication has been resolved before translation into Z. Clearly work needs to be done to ensure preservation of as much syntactic structure as possible.

5.1 Translation Algorithm for Behaviour Expressions

Let $ETS = \langle S, E, A, R, s_0, f_0 \rangle$ be the unique finite extended transition system associated with the LOTOS behaviour expression P. The translation $T(P)$ of the behaviour expression P will be the ZEST object given by:

$[Id]$

P

$States == S$

$s : States$
$Ext : States \times Id$
Declarations derived from the transition relation R

$Ext = E$

$INIT$

$\Delta(s)$

$s = s_0 \wedge f_0$

$Operation_1$

$\vdots$

$Operation_n$

Whenever $Ext = \varnothing$, the translation will omit Ext from the state schema completely.

Operation Schemas

The operation schemas contained within the ZEST object are derived from the finite set of transition relations generated from the LOTOS specification. For each $r \in R$ we generate a (partial) operation schema, and when all relations in R have been considered we merge together operation schemas which have the same name in a manner we describe below.

Let $r = \langle a, s_1, s_2, p, f \rangle \in R$ with $g = name(a)$. Then r will define a template schema of the form:

g

$\Delta(s)$
Declarations derived from a

(transition condition derived from s_1, s_2)$\wedge$
(pre-constraint derived from p)$\wedge$
(post-condition derived from f)

The constituent parts of this are:

1. Transition condition: The transition predicate will be $(s = s_1 \wedge s' = s_2)$.

2. Declarations: An action of the form $g?x_1 : t_1 \ldots ?x_n : t_n!E_1 \ldots !E_m$ is translated to the declaration

g

$\Delta(s), \Delta(x_1, \ldots, x_n)$
$t_1\, ch_1? : t_1, \ldots, t_n\, ch_n? : t_n$
$t_{n+1}\, ch_{n+1}! : t_{n+1}, \ldots, t_{n+m}\, ch_{n+m}! : t_{n+m}$

$\ldots$

where $t_{n+i} = type(E_i)$, and the appearance of t_j in a declaration $t_j\,ch_j?$ or $t_j\,ch_j!$ is its syntactic representation as a string of characters. This is needed for technical reasons.

In addition, the state schema is amended to include the declarations: $x_1 : t_1, \ldots, x_n : t_n$.

3. Pre-constraint: The pre-constraint is derived from the input/output of an action together with the predicate p. For an action of the form above, the pre-constraint is:

$$(x_1' = t_1\,ch_1? \wedge \ldots \wedge x_n' = t_n\,ch_n?) \wedge (t_{n+1}\,ch_{n+1}! = E_1 \wedge \ldots \wedge t_{n+m}\,ch_{n+m}! = E_m) \wedge$$
$$p[t_1\,ch_1?/x_1] \ldots [t_n\,ch_n?/x_n]$$

where $p[u/v]$ denotes substitution in the standard fashion. A further relabelling is also applied to p and the expressions E_i: for any variable, x say, which is bound when considering the schema alone (ie its binding occurrence occurs at the gate under consideration), any other subsequent occurrence of x in that action are replaced by x'. Furthermore, for any free variable, say y, that appears in the expressions E_i we conjoin $(y = y')$ to the predicate p. An example will make this clear:

(a) $g?x : t_1!(x + 2)$ will become:

$$\begin{array}{|l}
\multicolumn{1}{l}{g} \\
\hline
\Delta(s), \Delta(x) \\
t_1\,ch_1? : t_1 \\
t_1\,ch_2! : t_1 \\
\hline
(x' = t_1\,ch_1? \wedge t_1\,ch_2! = (x' + 2)) \\
\ldots \\
\hline
\end{array}$$

where here the relabelling has been applied to the expression $E_1 = (x + 2)$.

4. Post-condition: By construction, the action function f in the transition relation r will consist of a finite number of assignments of the form $v \leftarrow E$. These are re-written as $v' = E$. Binding occurrences of a variable are relabelled as in the predicate p described above.

Merging Schemas together

Given two partial operation schemas with the same name, built from two different transition relations, we combine them by merging the declarations in the usual fashion (there can be no clashes by construction) and taking the disjunction of the predicates.

For example, given the behaviour $input?x : t;\ a?y : u;\ input!(x + 2)!y;\ stop$, we generate two partial schemas describing the operation $input$:

$$\begin{array}{|l}
\multicolumn{1}{l}{input} \\
\hline
\Delta(s), \Delta(x) \\
tch_1? : t \\
\hline
(x' = tch_1? \wedge s = s_0 \wedge s' = s_1) \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{input} \\
\hline
\Delta(s) \\
tch_1! : t, uch_2! : u \\
\hline
(tch_1! = (x + 2) \wedge uch_2! = y \wedge s = s_2 \wedge s' = s_3) \\
\wedge (x' = x) \wedge (y' = y) \\
\hline
\end{array}$$

the combined schema will be:

input

$\Delta(s), \Delta(x)$
$tch_1? : t$
$tch_1! : t, uch_2! : u$

$(x' = tch_1? \land s = s_0 \land s' = s_1) \lor$
$((tch_1! = (x+2) \land uch_2! = y \land s = s_2 \land s' = s_3) \land (x' = x) \land (y' = y))$

To derive a ZEST translation from a LOTOS specification, we apply the translation algorithm to derive a unique finite ETS from the LOTOS specification, then apply the above translation rule to derive the ZEST object.

5.2 Translation of Data Types

In LOTOS, data types are specified using the language for abstract data types ACT ONE. ACT ONE is an algebraic specification method to write parameterized as well as unparameterized ADT specifications. These can be translated directly into the Z type system by writing the algebraic equations as axiomatic declatations in Z. The translation is straightforward in comparison with the translation of LOTOS behaviour expressions, and we illustrate the approach in the example given later. Z has the ability to represent all ACT ONE data types within it, however, two features cannot be modelled within the Z type system at this level of abstraction, namely those of naming a data type specification and the renaming of types.

6 Proof of Translation

A translation from LOTOS to Z must preserve the ETS semantics. We denote the mapping of LOTOS into an ETS by $[\![]\!]_{LOTOS}$, and that of Z into an ETS by $[\![]\!]_Z$. Within the semantics we are not concerned with exact denotations, but rather that the denotations are observationally equivalent. Thus we need to verify that $[\![Spec_L]\!]_{LOTOS} \approx [\![T(Spec_L)]\!]_Z$ for every LOTOS specification $Spec_L$.

Let ETS_L and ETS_Z be the extended transition systems derived from $Spec_L$ and $T(Spec_L)$ respectively. We construct a relation $\mathcal{R}$ between the states of ETS_L and ETS_Z which will define a bisimulation (in fact it defines a strong bisimulation) by defining $\mathcal{R}$ to contain the initial states, and then adding subsequent pairs of states as needed.

Let ETS_L and ETS_Z have initial states s_0 and u_0 respectively. Then set $(s_0, u_0) \in \mathcal{R}$. We shall show that if $(s_1, u_1) \in \mathcal{R}$ and $s_1 \xrightarrow{a} s_2$ in ETS_L, then we can find a state u_2 in ETS_Z such that $u_1 \xrightarrow{a} u_2$. The other half of the bisimulation is similar.

Let $(s_1, u_1) \in \mathcal{R}$, and suppose that $s_1 \xrightarrow{a} s_2$, where the action a is of the form $g?x : t_1!y$ and y has sort t_2 (wlog we can assume just one input and output, the argument generalises to any finite number of inputs and outputs). Then $\exists\, r$ with $r = \langle a, s_1, s_2, p, f\rangle$ in ETS_L.

Then what does the Z specification derived from ETS_L contain? The relation r gives rise to a partial schema with name g, viz:

$$
\begin{array}{|l}
\hline
g \\
\Delta(s), \Delta(x) \\
t_1\,ch_1? : t_1 \\
t_2\,ch_2! : t_2 \\
\text{Other declarations derived from other transition relations} \\
\hline
((s = s_1 \land s' = s_2) \land (x' = t_1\,ch_1?) \land (t_2\,ch_2! = y^+) \land p^+[t_1\,ch_1?/x] \land f^+) \qquad \lor \\
\text{Other predicates from other transition relations} \\
\hline
\end{array}
$$

where $+$ denotes the relabelling of x to x' in y, p and f.

When we calculate the ETS of the Z specification, this schema will give rise to one or more transition relations within ETS_Z. To find the relations in ETS_Z, we have to find out whether g is applicable at this state u_1. Now this schema is applicable whenever

$$
\begin{aligned}
(\exists\, s \bullet (s = s_1) \land \operatorname{pre} g) &\equiv \operatorname{pre} g(s_1/s) \\
&\Leftarrow \exists\, s', x', t_2\,ch_2! \bullet ((s' = s_2 \land x' = t_1\,ch_1?) \land (t_2\,ch_2! = y^+) \land p^+[t_1\,ch_1?/x] \land f^+) \\
&\equiv p^+[t_1\,ch_1?/x]
\end{aligned}
$$

is true.

Now since p evaluates to true at s_1 in ETS_L, $p^+[t_1\,ch_1?/x]$ will be true. Thus the relation

$$\langle g?t_1\,ch_1? : t_1!t_2\,ch_2!,\ u_1,\ u,\ p^+[t_1\,ch_1?/x],\ F\rangle$$

will be added to ETS_Z for some possibly new state u. Call this state u_2.

What is the action function F? F is the predicate that gives the effect on state and output variables of performing operation schema g at s_1, so F will be

$$(s' = s_2 \land x' = t_1\,ch_1?) \land (t_2\,ch_2! = y^+) \land f^+$$

What is the effect of invoking action a in ETS_L with a particular input? Let a be invoked with input $x = n$. Then the result is output $y[n/x]$ and the effect on variables is $f[n/x]$. Does this happen in ETS_Z? If g is invoked with input n, then the result is $(x' = n) \land (t_2\,ch_2! = y^+) \land f^+$. Hence the effect both in terms of output and effect on variables is the same in ETS_Z as in ETS_L.

Set $(s_2, u_2) \in \mathcal{R}$. Then by construction, the relation $\mathcal{R}$ is the desired bisimulation.

7 Example

We illustrate the translation algorithm and the semantic mappings by an example. Consider the LOTOS specification:

Specification Max3 [in1, in2, in3, out]
type natural **is**
  sorts *nat*
  opns $0 :\rightarrow nat$

$succ : nat \rightarrow nat$
$largest : nat, nat \rightarrow nat$

eqns

forall $x, y : nat$
ofsort nat
$largest(0, x) = x;$
$largest(x, y) = largest(y, x);$
$largest(succ(x), succ(y)) = succ(largest(x, y));$

endtype

behaviour
hide mid **in** (Max2[$in1, in2, mid$] $|[mid]|$ Max2[$mid, in3, out$])
where
process Max2[a, b, c] : **noexit** :=
$a?x : nat;\ b?y : nat;\ c!largest(x, y);$ Max3[$in1, in2, in3, out$]
[]
$b?y : nat;\ a?x : nat;\ c!largest(x, y);$ Max3[$in1, in2, in3, out$]
endproc
endspec

Renaming x and y to x_1, y_1 and x_2, y_2 to avoid name clashes in calls to Max2, we derive the extended transition system [19]:

$$\langle \{s_0, \ldots, s_{16}\}, \varnothing, \{in1, in2, in3, out, i\}, R, s_0, \epsilon \rangle$$

where the transition relations in the ETS are:

$$\begin{array}{l}
\langle in1?x_1 : nat, s_0, s_1, true, \epsilon\rangle, \langle in2?y_1 : nat, s_0, s_2, true, \epsilon\rangle, \langle in3?y_2 : nat, s_0, s_3, true, \epsilon\rangle,\\
\langle in2?y_1 : nat, s_1, s_4, true, \epsilon\rangle, \langle in3?y_2 : nat, s_1, s_5, true, \epsilon\rangle, \langle in1?x_1 : nat, s_2, s_6, true, \epsilon\rangle,\\
\langle in3?y_2 : nat, s_2, s_7, true, \epsilon\rangle, \langle in1?x_1 : nat, s_3, s_5, true, \epsilon\rangle, \langle in2?y_1 : nat, s_3, s_7, true, \epsilon\rangle,\\
\langle i, s_4, s_8, true, x_2 \leftarrow largest(x_1, y_1)\rangle, \langle in3?y_2 : nat, s_4, s_9, true, \epsilon\rangle, \langle in2?y_1 : nat, s_5, s_9, true, \epsilon\rangle,\\
\langle i, s_6, s_{10}, true, x_2 \leftarrow largest(x_1, y_1)\rangle, \langle in3?y_2 : nat, s_6, s_{11}, true, \epsilon\rangle, \langle in1?x_1 : nat, s_7, s_{11}, true, \epsilon\rangle,\\
\langle in3?y_2 : nat, s_8, s_{12}, true, \epsilon\rangle, \langle i, s_9, s_{13}, true, x_2 \leftarrow largest(x_1, y_1)\rangle, \langle in3?y_2 : nat, s_{10}, s_{14}, true, \epsilon\rangle,\\
\langle i, s_{11}, s_{15}, true, x_2 \leftarrow largest(x_1, y_1)\rangle, \langle out!largest(x_2, y_2), s_{12}, s_0, true, \epsilon\rangle,\\
\langle out!largest(x_2, y_2), s_{13}, s_0, true, \epsilon\rangle, \langle out!largest(x_2, y_2), s_{14}, s_0, true, \epsilon\rangle,\\
\langle out!largest(x_2, y_2), s_{15}, s_0, true, \epsilon\rangle
\end{array}$$

The translation algorithm will produce a ZEST specification with the following representation of the type *natural*:

$[nat]$

$$\begin{array}{|l}
0 : nat\\
succ : nat \rightarrow nat\\
largest : nat \times nat \rightarrow nat\\
\hline
\forall\, x, y : nat \bullet largest(0, x) = x\\
\forall\, x, y : nat \bullet largest(x, y) = largest(y, x)\\
\forall\, x, y : nat \bullet largest(succ(x), succ(y)) = succ(largest(x, y))
\end{array}$$

Notice that in the translation of constants we remove the arrow, as in $\rightarrow nat$. The commas in an n-ary operation are replaced by $\times$ in the Z translation. The **ofsort** *nat* is superfluous in the Z specification. The one aspect which is not translated is the name given to encapsulated signature plus equations.

The behaviour in the LOTOS specification is represented in the ZEST specification as an object (after a small amount of simplification):

P

$$States == \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{12}, s_{13}, s_{14}, s_{15}, s_{16}\}$$

$$s : States$$
$$x_1, x_2, y_1, y_2 : nat$$

$INIT$

$$\Delta(s)$$
$$s = s_0$$

$in1$

$$\Delta(s), \Delta(x_1)$$
$$natch_1? : nat$$
$$natch_1? = x_1' \wedge ((s = s_0 \wedge s' = s_1) \vee (s = s_2 \wedge s' = s_6) \vee (s = s_3 \wedge s' = s_5) \vee (s = s_7 \wedge s' = s_{11}))$$

$in2$

$$\Delta(s), \Delta(y_1)$$
$$natch_1? : nat$$
$$natch_1? = y_1' \wedge ((s = s_0 \wedge s' = s_2) \vee (s = s_1 \wedge s' = s_4) \vee (s = s_3 \wedge s' = s_7) \vee (s = s_5 \wedge s' = s_9))$$

$in3$

$$\Delta(s), \Delta(y_2)$$
$$natch_1? : nat$$
$$natch_1? = y_2' \wedge ((s = s_0 \wedge s' = s_3) \vee (s = s_1 \wedge s' = s_5) \vee (s = s_2 \wedge s' = s_7) \vee$$
$$(s = s_4 \wedge s' = s_9) \vee (s = s_6 \wedge s' = s_{11}) \vee (s = s_8 \wedge s' = s_{12}) \vee (s = s_{10} \wedge s' = s_{14}))$$

i

$$\Delta(s), \Delta(x_2)$$
$$x_2' = largest(x_1, y_1) \wedge$$
$$((s = s_4 \wedge s' = s_8) \vee (s = s_6 \wedge s' = s_{10}) \vee (s = s_9 \wedge s' = s_{13}) \vee (s = s_{11} \wedge s' = s_{15}))$$

out

$$\Delta(s)$$
$$natch_1! : nat$$
$$natch_1! = largest(x_2, y_2) \wedge (s = s_{12} \vee s = s_{13} \vee s = s_{14} \vee s = s_{15}) \wedge s' = s_0$$

8 Conclusions

The work described here aims to provide a first step in defining a translation between LOTOS and Z. The translation mechanism was defined, together with a common semantic framework that verifies the translation algorithm.

Extended transition systems provided the common semantic framework and the relationship between the ETS semantics for LOTOS and the standard LTS semantics needs to be explored. However, although we have used an ETS semantics for LOTOS, any LTS semantics for LOTOS that could be embedded in a finite ETS will produce a translation to Z correct with respect to that semantics.

References

[1] 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.

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

[3] H. Bowman, J. Derrick, P. Linington, and M. Steen. FDTs for ODP. *Computer Standards and Interfaces*, 17:457–479, September 1995.

[4] 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.

[5] 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.

[6] J. Derrick, H. Bowman, and M. Steen. Maintaining cross viewpoint consistency using Z. In K. Raymond and L. Armstrong, editors, *IFIP TC6 International Conference on Open Distributed Processing*, pages 413–424, Brisbane, Australia, February 1995. Chapman and Hall.

[7] J. Derrick, H. Bowman, and M. Steen. Viewpoints and Objects. In J. P. Bowen and M. G. Hinchey, editors, *Ninth Annual Z User Workshop*, LNCS 967, pages 449–468, Limerick, September 1995. Springer-Verlag.

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

[9] A. Fantechi, S. Gnesi, and C. Laneve. Two standards means problems : A case study on formal protocol descriptions. *Computer Standards and Interfaces*, 9:11–19, 1989.

[10] M. Hennessy and H. Lin. Symbolic bisimulations. Technical Report 1-92, Computer Science, University of Sussex, Brighton, England, 1992.

[11] ISO. Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour, 1989. IS 8807.

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

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

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

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

[16] J. M. Spivey. *Understanding Z: A specification language and its formal semantics.* Cambridge University Press, 1988.

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

[18] M. W. A. Steen, H. Bowman, and J. Derrick. Composition of LOTOS specifications. In P. Dembinski and M. Sredniawa, editors, *Protocol Specification, Testing and Verification, XV*, pages 73–88, Warsaw, Poland, 1995. Chapman & Hall.

[19] J-P. Wu and S. Chanson. Translation from LOTOS and Estelle specifications to extended transition system and its verification. In S. T. Voung, editor, *Formal Description Techniques, II*, pages 533–549, Vancouver, Canada, December 1989. North-Holland.