A comparison of refinement orderings and their associated simulation rules

Christie Bolton, Jim Davies\textsuperscript{1,2}

\textit{Oxford University Computing Laboratory}
\textit{Wolfson Building, Parks Road}
\textit{Oxford OX1 3QD, England}

\textbf{Abstract}

In this paper we compare the refinement orderings, and their associated simulation rules, of state-based specification languages such as Z and Object-Z with the refinement orderings of event-based specification languages such as CSP. We prove with a simple counter-example that data refinement, established using the standard simulation rules for Z, does not imply failures refinement in CSP. This contradicts accepted results.

Having explored the differences between the simulation rules for establishing data refinement and those for establishing the refinement of action systems and state-transition systems—models in which refinement is equivalent to failures refinement within CSP—we present a new set of simulation rules for data types. These alternative rules are both sound and jointly complete with respect to the stable failures refinement ordering. Furthermore we present an alternative refinement ordering for CSP, one in which refinement is equivalent to data refinement in Z.

\textbf{1 Introduction}

During the development of large complex systems it is often desirable to construct a series of models, each model resolving ambiguities or areas of undefinedness in the previous model. In such a methodology we may wish to prove formally that each model is refined by the next. Our intuition tells us that one model is refined by another if the second model satisfies all the properties and constraints imposed by the first. As we will illustrate, the precise definition of refinement depends on our chosen formalism.

In this paper we focus on the refinement of abstract data types expressed in the state-based specification languages Z [22] and Object-Z [10], and the
refinement of processes expressed in the event-based specification language CSP [13]. Refinement within both Z and Object-Z generally employs sets of simulation rules [14], although it may be established by considering the relational semantics within Z or by considering the histories semantics [19] within Object-Z.

We expose and discuss the differences between data refinement [8] and refinement within the stable failures model [18] for CSP, and we show that whilst both models give information about possible sequences of interaction and both models give refusal information, they are not equivalent. The stable failures and failures-divergences [18] models within CSP give information about the availability of combinations of events whereas the relational semantics of Z gives refusal information only about individual operations. Hence, data refinement does not imply failures refinement.

In Section 2 we present a simple counter-example proving that the simulation rules derived to correspond precisely to the blocking interpretation of the relational semantics [5] of data types expressed in Z are not sound with respect to the stable failures model [18] of CSP. From this it follows immediately that data refinement does not imply stable failures refinement, contradicting accepted results [21,6,5,11,9].

Having explored the differences between these two refinement orderings, we have two options if we wish to compare state-based and event-based models: either we must identify a semantic model for CSP that has the same information content as the relational semantics of data types, or we must identify a set of simulation rules that correspond to the stable failures model for CSP. We adopt the former approach in Section 5 and, having explored the differences between the simulation rules for establishing data refinement and those for establishing the refinement of action systems [1] and state-transition systems [17,16]—models in which refinement is equivalent to failures refinement within CSP—in Section 3, we adopt the latter approach in Section 4. The main contributions of the paper are as follows:

• we demonstrate that data refinement in Z\(^3\) does not imply stable failures refinement within CSP\(^4\);
• we present a set of simulation rules for data types expressed in Object-Z\(^5\) that are both sound and jointly complete with respect to the stable failures refinement ordering of CSP;
• we describe an alternative semantic model for CSP whose information content is precisely that of the relational semantics of data types;

\(^3\) As illustrated in [4] this results extends to Object-Z.
\(^4\) In this paper we assume knowledge of Z, Object-Z and CSP. Readers unfamiliar with these notations should see [22,23,9], and [20] and [13,18] respectively.
\(^5\) Object-Z rather than Z is our chosen paradigm as, for divergence-free systems, these simulation rules correspond precisely to the established histories model for Object-Z.
we conclude with a table distinguishing between those refinement orderings and sets of simulation rules that give information about the availability of combinations of events or operations, and those that give refusal information only on an operation-by-operation basis.

2 Data refinement does not imply failures refinement

We prove that data refinement is not equivalent to stable failures refinement with a simple example as described in [2]. Let data types $P$ and $Q$ be defined as follows: both data types have the same internal state space $State$ containing the integers from 0 up to 4; given environment $Env$ containing the single element $e$, both data types have the same finalisation operations mapping the entire state space onto $Env$;

\[
\begin{array}{c}
\text{State} \\
\text{s : 0..4}
\end{array}
\quad
\begin{array}{c}
\text{Final} \\
\text{State; Env'}
\end{array}
\]

and both data types share the operations $A$, $B$ and $C$ as illustrated in Figure 1. Operation $A$ maps state 1 onto state 0; operation $B$ maps states 2 and 3 onto state 0; and operation $C$ maps states 2 and 4 onto state 0. More formally,

\[
\begin{array}{c}
A \\
\Delta State \\
s = 1 \land s' = 0
\end{array}
\quad
\begin{array}{c}
B \\
\Delta State \\
s \in \{2, 3\} \land s' = 0
\end{array}
\quad
\begin{array}{c}
C \\
\Delta State \\
s \in \{2, 4\} \land s' = 0
\end{array}
\]

The difference between the two data types lies in the range of their initialisation relations: data type $P$ can initially be in states 1 or 2 whereas data type $Q$ can initially be in states 1, 3 or 4.
We prove in Lemma 2.1 below that data type $P$ is refined by data type $Q$ by proving that $P$ simulates $Q$ within the context of the blocking semantics\(^6\); more specifically we identify a retrieve relation $Retr$ such that the backwards simulation rules corresponding to the blocking semantics, as presented in Appendix A, all hold. Data refinement follows automatically due to the soundness of the simulation rules [5].

**Lemma 2.1** Data type $P$ is refined by $Q$ where $P$ and $Q$ are as defined above.

**Proof.** For clarity, we distinguish syntactically between the shared components of the data types; we define $State_P$ and $State_Q$ to be equal to $State; A_P$ and $A_Q$ to be equal to $A$ and so on. Then letting retrieve relation $Retr$ from the state space of $Q$ to the state space of $P$ be as illustrated in Figure 2 and formally defined below:

\[
\begin{align*}
Retr & : \text{State}_P \\
p & : \text{State}_P \\
q & : \text{State}_Q \\
( q \in \{0, 1, 2\} \land p = q ) & \lor ( q \in \{3, 4\} \land p \in \{1, 2\} )
\end{align*}
\]

we simply need to prove that the following predicates are true.

(i) $\forall State_P' ; State_Q' \bullet QInit \land Retr' \Rightarrow PInit$

(ii) (a) $\forall State_Q \bullet ( \forall State_P \bullet Retr \Rightarrow pre A_P ) \Rightarrow pre A_Q$

(b) $\forall State_Q \bullet ( \forall State_P \bullet Retr \Rightarrow pre B_P ) \Rightarrow pre B_Q$

(c) $\forall State_Q \bullet ( \forall State_P \bullet Retr \Rightarrow pre C_P ) \Rightarrow pre C_Q$

(iii) (a) $\forall State_P ; State_Q ; State_Q' \mid A_Q \land Retr' \bullet ( \exists State_P \bullet Retr \land A_P )$

\(^6\) We work within the blocking framework to facilitate our comparison in Section 3.
The first rule reduces to

$$\forall p', q' : State \bullet \left( \left( q' = 1 \land p' = 1 \right) \lor \left( q' = 3 \lor q' = 4 \right) \land \left( p' = 1 \lor p' = 2 \right) \right) \Rightarrow \left( p' = 1 \lor p' = 2 \right)$$

which is equivalent to true. In the case of operation $A$, the inner clause of the second rule, or to be precise the predicate $\forall StateP \bullet Retr \Rightarrow \text{pre } A_P$, is equivalent to

$$\forall p : State \bullet \left( \left( q = 0 \land p = 0 \right) \lor \left( q = 1 \land p = 1 \right) \lor \left( q = 2 \land p = 2 \right) \lor \left( q = 3 \land \left( p = 1 \lor p = 2 \right) \right) \lor \left( q = 4 \land \left( p = 1 \lor p = 2 \right) \right) \right) \Rightarrow \left( p = 1 \right)$$

which reduces to $(q \neq 0) \land (q \neq 2) \land (q \neq 3) \land (q \neq 4)$ and hence to $q = 1$.

In the case of operation $A$, the second rule is therefore equivalent to

$$\forall q : State \bullet q = 1 \Rightarrow q = 1$$

which is of course true. Similarly, in the case of operation $B$, the inner clause of the second rule is equivalent to

$$\forall p : State \bullet \left( \left( q = 0 \land p = 0 \right) \lor \left( q = 1 \land p = 1 \right) \lor \left( q = 2 \land p = 2 \right) \lor \left( q = 3 \land \left( p = 1 \lor p = 2 \right) \right) \lor \left( q = 4 \land \left( p = 1 \lor p = 2 \right) \right) \right) \Rightarrow \left( p = 2 \lor p = 3 \right)$$

which reduces to $(q \neq 0) \land (q \neq 1) \land (q \neq 3) \land (q \neq 4)$ and hence to $q = 2$.

In the case of operation $B$, the second rule is therefore equivalent to

$$\forall q : State \bullet q = 2 \Rightarrow \left( \left( q = 2 \right) \lor \left( q = 3 \right) \right)$$

which is once again true. Furthermore, by symmetry it follows immediately that the second rule must also be true for operation $C$. 
We have shown that rule one holds and that rules two holds for each of the operations on the data types. Next we consider the third rule. In the case of operation $A$ this rule reduces to

$$\forall p', q, q' : State \mid q = 1 \land q' = 0 \land p' = 0 \bullet$$

$$\exists p : State \bullet p = 1 \land p' = 0 \land (q = 1 \lor q = 3 \lor q = 4)$$

which is equivalent to true. In the case of operation $B$, the third rule reduces to

$$\forall p', q, q' : State \mid (q = 2 \lor q = 3) \land q' = 0 \land p' = 0 \bullet$$

$$\exists p : State \bullet p = 2 \land p' = 0 \land (q = 2 \lor q = 3 \lor q = 4)$$

$$\lor p = 3 \land p' = 0 \land false$$

which is also equivalent to true. Once more we appeal to symmetry to establish immediately that the third rule must also be true for operation $c$.

Finally, observing that the fourth simulation rule is trivially true, we conclude that all the backwards simulation rules hold and hence that $P$ simulates $Q$. Hence, by the soundness of the simulation rules [5], we deduce that data type $P$ is refined by data type $Q$.

Applying the natural translation from data types within a blocking context to CSP processes (see: [24] for further details) we obtain $\text{process}_b(P)$, the process corresponding to data type $P$.

$$\text{process}_b(P) = \text{let}$$

$$P(s) = (s == 0) \land \text{Stop}$$

$$\lor (s == 1) \land \text{op.a \rightarrow P(0)}$$

$$\lor (s == 2) \land (\text{op.b \rightarrow P(0) \land op.c \rightarrow P(0)})$$

$$\lor (s == 3) \land \text{op.b \rightarrow P(0)}$$

$$\lor (s == 4) \land \text{op.c \rightarrow P(0)}$$

$$\lor \text{final.e \rightarrow Stop}$$

within

$$\square e : \{e\} \land \square s : \{1, 2\} \land \text{init.e \rightarrow P(s)}$$
With a little rearranging and some obvious renaming, this process may equivalently be written as process $\text{ProcP}$ below:

$$
\text{ProcP} = i \rightarrow ( ( a \rightarrow f \rightarrow \text{Stop} )
\begin{array}{c}
\land \\
\lor \\
\lor
\end{array}
( b \rightarrow f \rightarrow \text{Stop} ) \square ( c \rightarrow f \rightarrow \text{Stop} ))
\square
f \rightarrow \text{Stop })
$$

Applying similar techniques we may obtain the process $\text{ProcQ}$ corresponding to data type $Q$.

$$
\text{ProcQ} = i \rightarrow ( ( a \rightarrow f \rightarrow \text{Stop} )
\begin{array}{c}
\land \\
\lor \\
\lor
\end{array}
b \rightarrow f \rightarrow \text{Stop }
\begin{array}{c}
\land \\
\lor
\end{array}
c \rightarrow f \rightarrow \text{Stop }
\square
f \rightarrow \text{Stop }
$$

**Observation 1** The process corresponding to $P$ is not refined within the stable failures model by the process corresponding to $Q$ where data types $P$ and $Q$ are as defined above.

**Proof.** We see that after event $i$, the process $\text{ProcP}$ can refuse any set of events that does not contain $f$ and that does not contain both $a$ and $b$, or both $a$ and $c$, whereas, after event $i$ the process $\text{ProcQ}$ can refuse any set of events that does not contain $f$ and that contains at most two of $a$, $b$ and $c$. In particular, $\text{ProcQ}$ can refuse the sets $\{a, b\}$ and $\{a, c\}$ whereas $\text{ProcP}$ cannot: the trace-refusal pairs $(\langle i \rangle, \{a, b\})$ and $(\langle i \rangle, \{a, c\})$ both lie in the failures of $\text{ProcQ}$ but not of $\text{ProcP}$.

We have shown that the set of failures of the process corresponding to $Q$ is not a subset of the set of failures of the process corresponding to $P$. Hence the process corresponding to $P$ is not refined within the stable failures model by the process corresponding to $Q$.  

\[ \square \]

**Theorem 2.2** Data refinement does not imply stable failures refinement

**Proof.** We have identified two data types $P$ and $Q$. In Lemma 2.1 we proved that $P$ is refined by $Q$ and in Observation 1 we proved that the process corresponding to $P$ is not refined within the stable failures semantic model by the process corresponding to $Q$. From this it follows that data refinement does not imply stable failures refinement: data refinement and stable failures refinement are not equivalent. We proved this result within the context of the blocking semantics. A similar proof could be shown to establish the same result in the context of the non-blocking semantics.  

\[ \square \]
3 The simulation rules of Josephs, He, Woocock and Morgan

Simulation rules for abstract data types expressed in Z may be given either in terms of the schemas defining the components—state space, initialisation, finalisation and named operations—of the data types, or in terms of the underlying relations. Furthermore simulation rules may be derived to correspond to either a blocking or a non-blocking semantic model (see [6]). Whichever context we choose, two sets of simulation rules, forwards and backwards (sometimes referred to as downwards and upwards), will be presented. If some of the non-determinism has been resolved then we look to establish a forwards simulation, whereas if some of the non-determinism of the more abstract of the two data types has been postponed then we look to establish a backwards simulation. The non-blocking and blocking versions of the simulation rules are shown to be sound and jointly complete in [23] and [5] respectively.

In this section we compare the information content of simulation rules for establishing data refinement within Z to the simulation rules of Josephs [15], He [12], and Woocock and Morgan [25]. Woodcock and Morgan’s simulation rules are for establishing the refinement of action systems and He and Josephs rules are for establishing the refinement of state transition systems.

These models each adopt a blocking approach, hence our choice of context in Section 2.

Josephs, He, and Woocock and Morgan have each proved that their simulation rules are sound and jointly complete with respect to Roscoe’s failure-divergences refinement ordering; it follows that, for divergence-free systems, they are sound and jointly complete with respect to his stable failures refinement ordering. Syntax aside, the difference between these rules and those corresponding to the data refinement in Z is that the rules of Josephs, He, and Woocock and Morgan each capture the availability of combinations of operations whereas the simulation rules derived from the relational semantics of Z provide information only about the availability of individual operations. This discrepancy occurs only in one rule, the backwards simulation rule concerning the availability of operations. The relevant rules of Josephs, He, and Woocock and Morgan ensure that if a state within the more concrete model lies outside the preconditions (union of the domains) of any given set of actions or transitions, then there must be a corresponding state within the more abstract model that also lies outside the preconditions of that set of operations. The corresponding rule for data types is weaker: it requires that if a state within the more concrete model lies outside the domain of an individual operation then there must be a corresponding state within the more abstract model that also lies outside the domain of that individual operation.

Consider once more data types $P$ and $Q$ as presented in Section 2. State

\footnote{We assume knowledge of the action systems formalism and of state transition systems. Readers unfamiliar with these notations should see [7].}
3 on data type \( Q \) lies outside the domains of both operations \( A \) and \( C \). We identified corresponding state 2 on data type \( P \) which lies outside the domain of operation \( A \) as well as corresponding state 1 on \( P \) which lies outside the domain of \( C \); hence the third backwards simulation rule held. However, had we been required to identify a single corresponding state lying outside the domain of both these operations, we would have failed: there is no such corresponding state. Had we translated \( P \) and \( Q \) to the appropriate formalisms, the simulation rules of Josephs, He, and Woodcock and Morgan would not have held.

4 Simulation rules corresponding to the histories semantic model for Object-Z

In this section we present rules for verifying that one data type is refined by another within the stable failures model. We do this in the context of Object-Z [10] since that has traditionally been given a histories semantics that is equivalent to Roscoe’s failures-divergences semantic model [19]. These rules were introduced in [2,4].

If \( A \) and \( C \) are Object-Z classes with the same set of operation names \( X \), then given a retrieve relation \( Retr \) relating the state spaces of the two classes, the forwards/downwards and backwards/upwards simulation rules corresponding to the histories semantic model are respectively as follows.

\[
\forall C \cdot C \cdot \text{INIT} \Rightarrow (\exists A \cdot A \cdot \text{INIT} \land Retr) \quad (F_{OZ-h1})
\]

\[
\forall Op : X \cdot A \cdot \text{STATE} \cdot C \cdot \text{STATE} \\
Retr \Rightarrow (\text{pre } A \cdot Op \Leftrightarrow \text{pre } C \cdot Op) \quad (F_{OZ-h2})
\]

\[
\forall Op : X \cdot A \cdot \text{STATE} \cdot C \cdot \text{STATE} \cdot C \cdot \text{STATE'} \\
Retr \land C \cdot Op \Rightarrow \exists A \cdot \text{STATE'} \cdot Retr' \land A \cdot Op \quad (F_{OZ-h3})
\]

\[
\forall A \cdot \text{STATE} \cdot C \cdot \text{STATE} \cdot C \cdot \text{INIT} \land Retr \Rightarrow A \cdot \text{INIT} \quad (B_{OZ-h1})
\]

\[
\forall C \cdot \text{STATE} \cdot \exists A \cdot \text{STATE} \\
\forall Op : X \cdot Retr \land (\text{pre } A \cdot Op \Rightarrow \text{pre } C \cdot Op) \quad (B_{OZ-h2})
\]

\[
\forall Op : X \cdot A \cdot \text{STATE'} \cdot C \cdot \text{STATE} \cdot C \cdot \text{STATE'} \\
Retr' \land C \cdot Op \Rightarrow \exists A \cdot \text{STATE} \cdot Retr \land A \cdot Op \quad (B_{OZ-h3})
\]

The soundness and join completeness of these rules follows directly from the corresponding result for Josephs’ rules [4]. They differ from the stan-
standard [9] simulation rules for Object-Z as described in Appendix B only in the rule $B2$. The difference lies in the scope of the quantifications: in rule $B_{OZ-r}2$ we have the universal quantification of all operations over the existential quantification of all abstract states ($\forall Op : X \bullet \exists A.\text{STATE} \bullet \ldots$) whereas in rule $B_{OZ-h}2$ we have a stronger predicate, the existential quantification of all abstract states over the universal quantification of all operations ($\exists A.\text{STATE} \bullet \forall Op : X \bullet \ldots$).

5 The singleton failures semantic model

Having identified the discrepancy between data refinement and stable failures refinement, and having identified simulation rules for Object-Z that are both sound and jointly complete with respect to Roscoe’s failures divergences refinement ordering, we now present a simplified version of the singleton failures semantic model for CSP - a model defined such that its information content is precisely that of the relational semantics of data types. This model was introduced, and shown to be equivalent to the relational semantics of data types in [3,2].

The singleton failures semantic model, mirroring the information content of the relational semantics of data types, records availability of events on an individual basis. The semantic function is essentially a projection of the stable failures semantic function, recording only those trace-refusal pairs in which the cardinality of the refusal set is at most one. Given the set $\text{Refusal}_1$ containing sets of events of cardinality at most one, and the set $\text{Trace}$ containing all possible sequences of events, the simplified version of the semantic function $S$ is defined as follows:

$$ S[P] = F[P] \cap (\text{Trace} \times \text{Refusal}_1). $$

where $P$ is a basic process, that is a process constructed using only the operators $\text{Stop}, \rightarrow, \sqcap, \square,$ and $\downarrow x \parallel y$.

As with the other models within CSP, refinement is reverse containment. Given any basic processes $P$ and $Q$,

$$ P \sqsubseteq_s Q \iff S[Q] \subseteq S[P]. $$

Furthermore, all laws that hold for the stable failures model and that are expressible only in terms of the basic operators hold for the singleton failures model when applied to basic processes.

---

8 The exception is the hiding operator with which, for simplicity, we will not concern ourselves in this paper. The interested reader should see [3,2].
6 Conclusions

The new ideas contained within this paper all stem from the observation that data refinement is not equivalent to stable failures refinement. We presented a simple example that exposed the discrepancy between the models: the relational semantics of data types records the availability of operations individually whereas the stable failures semantic model records the availability of combinations of events.

If we wish to compare state-based and event-based models, either we must adopt a set of simulation rules that are both sound and jointly complete with respect to stable failures or failures-divergences models (we took this approach in Section 4, presenting such a set of rules for Object-Z), or we must adopt a semantic model for CSP that has the same information content as the relational semantics of data types (we took this approach in Section 5, introducing the singleton failures semantic model).

We conclude the paper with a table that provides a comparison between the information content of the refinement orderings and simulation rules that we have considered in this paper. We split them into two categories: those that provide refusal information on an individual basis, and those that provide refusal information for combinations of events or operations.

<table>
<thead>
<tr>
<th>Refinement model</th>
<th>Simulation rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Individual refusal information</strong></td>
<td></td>
</tr>
<tr>
<td>Data refinement eg [8]</td>
<td>Z eg [23,5,9]</td>
</tr>
<tr>
<td>Data refinement in Z eg [23,5,9]</td>
<td>Object-Z eg [21,9]</td>
</tr>
<tr>
<td>Singleton failures model for CSP eg [3,2]</td>
<td></td>
</tr>
<tr>
<td><strong>Combined refusal information</strong></td>
<td></td>
</tr>
<tr>
<td>Stable failures model for CSP eg [18]</td>
<td>State transition systems eg [15,12]</td>
</tr>
<tr>
<td>Concurrent rules for Object-Z eg [4]</td>
<td></td>
</tr>
</tbody>
</table>
A  Simulation rules for Z

In this section we present the simulation rules for Z within the context of the blocking semantic model\(^9\). Given data types \(A\) and \(C\) with state spaces \(A\text{State}\) and \(C\text{State}\), with initialisations and finalisations \(A\text{Init}\) and \(A\text{Final}\), and \(C\text{Init}\) and \(C\text{Final}\) respectively and with corresponding operations \(A\text{Op}\) and \(C\text{Op}\), and given retrieve relation \(Retr\), the schema versions of the blocking forwards simulation rules are:

\[
\forall C\text{State}' \cdot C\text{Init} \Rightarrow \exists A\text{State}' \mid A\text{Init} \cdot Retr \tag{F_Z 1}
\]

\[
\forall A\text{State} \mid \text{pre } A\text{Op} \cdot \forall C\text{State} \mid Retr \cdot \text{pre } C\text{Op} \tag{F_Z 2}
\]

\[
\forall A\text{State} \cdot \forall C\text{State} \cdot Retr \cdot \\
\forall C\text{State}' \mid C\text{Op} \cdot \exists A\text{State}' \mid A\text{Op} \cdot Retr' \tag{F_Z 3}
\]

\[
\forall A\text{State}; \text{Env} \cdot Retr \land C\text{Final} \Rightarrow A\text{Final} \tag{F_Z 4}
\]

Similarly, the schema versions of the blocking backwards simulation rules are as follows.

\[
\forall A\text{State}'; C\text{State}' \cdot C\text{Init} \land Retr' \Rightarrow A\text{Init} \tag{B_Z 1}
\]

\[
\forall C\text{State} \cdot (\forall A\text{State} \cdot Retr \Rightarrow \text{pre } A\text{Op}) \Rightarrow \text{pre } C\text{Op} \tag{B_Z 2}
\]

\[
\forall A\text{State}'; C\text{State}; C\text{State}' \mid C\text{Op} \land Retr' \cdot \\
(\exists A\text{State} \cdot Retr \land A\text{Op}) \tag{B_Z 3}
\]

\[
\forall C\text{State}; \text{Env} \mid C\text{Final} \cdot \exists A\text{State} \mid Retr \cdot A\text{Final} \tag{B_Z 4}
\]

B  Simulation rules for Object-Z

In this section we present the simulation rules for Object-Z. These rules are sound and jointly complete with respect to the relational semantics given to Object-Z: see [9]. If \(A\) and \(C\) are Object-Z classes with the same set of operation names \(X\), then given a retrieve relation \(Retr\) relating the state spaces of the two classes, the forwards/downwards and backwards/upwards

\(^9\) For a more detailed description of this model and the derivation of the simulation rules, see [5,2].
simulation rules for Object-Z, as presented in [21,9], are respectively as follows:

\[ \forall \text{C} \cdot \text{C.Init} \Rightarrow (\exists \text{A} \cdot \text{A.Init} \land \text{Retr}) \]  
\( (\text{F}_{\text{OZ}-r} \ 1) \)

\[ \forall \text{Op} : X; \text{A.State}; \text{C.State} \bullet \]  
\[ \text{Retr} \Rightarrow (\text{pre A.Op} \leftrightarrow \text{pre C.Op}) \]  
\( (\text{F}_{\text{OZ}-r} \ 2) \)

\[ \forall \text{Op} : X; \text{A.State}; \text{C.State}; \text{C.State'} \bullet \]  
\[ \text{Retr} \land \text{C.Op} \Rightarrow \exists \text{A.State'} \bullet \text{Retr'} \land \text{A.Op} \]  
\( (\text{F}_{\text{OZ}-r} \ 3) \)

\[ \forall \text{A.State}; \text{C.State} \bullet \text{C.Init} \land \text{Retr} \Rightarrow \text{A.Init} \]  
\( (\text{B}_{\text{OZ}-r} \ 1) \)

\[ \forall \text{Op} : X; \text{C.State} \bullet \]  
\[ \exists \text{A.State} \bullet \text{Retr} \land (\text{pre A.Op} \Rightarrow \text{pre C.Op}) \]  
\( (\text{B}_{\text{OZ}-r} \ 2) \)

\[ \forall \text{Op} : X; \text{A.State'}; \text{C.State}; \text{C.State'} \bullet \]  
\[ \text{Retr'} \land \text{C.Op} \Rightarrow \exists \text{A.State} \bullet \text{Retr} \land \text{A.Op} \]  
\( (\text{B}_{\text{OZ}-r} \ 3) \)

References


