Understanding, explaining, and deriving refinement

Eerke Boiten and John Derrick

Abstract Much of what drove us in over twenty years of research in refinement, starting with Z in particular, was the desire to understand where refinement rules came from. The relational model of refinement provided a solid starting point which allowed the derivation of Z refinement rules. Not only did this explain and verify the existing rules – more importantly, it also allowed alternative derivations for different and generalised notions of refinement. In this chapter, we briefly describe the context of our early efforts in this area and Susan Stepney's role in this, before moving on to the motivation and exploration of a recently developed primitive model of refinement: concrete state machines with anonymous transitions.

1 Introduction: Z Refinement Theories of the Late 1990s

At the Formal Methods Europe conference at Oxford in 1996 [20], there was a reception to celebrate the launch of Jim and Jim's (Woodcock and Davies) book on Understanding Z [30]. This was a fascinatingly different book on Z for those with a firm interest in Z refinement like ourselves, one as aspirational and inspirational as the slightly earlier "Z in Practice" [5]. It contained a full derivation of the downward simulation rules for states-and-operations specifications, *with* inputs and outputs, all the way from Hoare, He and Sanders' relational refinement rules [22], with the punny "relaxing" and "unwinding" important steps of the derivation process. In addition, unlike most Z textbooks, it also included upward simulation rules to achieve completeness – we were told that these had turned out to be necessary in an exciting but mostly confidential industry project called "Mondex" [29]. There was

Eerke Boiten
De Montfort University, Leicester, LE1 9BH e-mail: eerke.boiten@dmu.ac.uk

John Derrick
University of Sheffield, Sheffield, S1 4DP e-mail: J.Derrick@sheffield.ac.uk

also a strong hint then that the Mondex team couldn't tell us yet about everything they had discovered about refinement while doing this research.

At that same conference, we presented the most theoretical Z refinement paper we had produced so far [8], which constructed a common refinement of two Z specifications, in support of our work on viewpoint refinement, extending ideas of Ainsworth, Cruickshank, Wallis and Groves [3] to also cover *data* refinement. To satisfy our funders EPSRC that we were being practical and building prototype tools, we had also implemented [6] this construction in the Generic version of the Z Formaliser tool [19]. This tool was being developed concurrently by Susan Stepney, who provided us with advice and debugging, and we all found out more about Smalltalk in the process.

Our viewpoint unification technique, as we called it, gradually relaxed the constraints on different specifications of the same Z operation that we needed a common refinement of, to constructively show consistency, and continue from there. If the postconditions were different but the preconditions identical, conjunction of operations was sufficient. For where the postconditions differed, a form of disjunction delivered a common refinement. If the state spaces were different, we could use a "correspondence relation" to still find a common data refinement. But that is where our desire to allow viewpoints to take different perspectives hit the buffers as far as conventional Z refinement went. In particular, two viewpoint operations with different inputs or outputs could never have a common refinement according to the theory as presented in [30] or Spivey's earlier Z bible [27]. Which was odd, as both of these books already contained "refinement" examples that added inputs or outputs – not least Spivey's "birthday book" running example.

Based on all this, we set out to reconstruct a sensible theory for how refinement in Z might also include changes to inputs and outputs. The examples in the noted textbooks formed a starting point for conservative generalisation of the standard rules. We extracted some of the informal and common sense rationales, which can be found in the paper "IO Refinement in Z" [7]. The final version of this paper, inspired by the derivations in [30], reverted from common sense reasoning to solid mathematics to establish the rules for IO refinement. The steps in the Woodcock and Davies derivation of Z refinement from relational refinement where concrete ("global") input and output sequences were equated to their abstract ("local") counterparts were ripe for generalisation. So, initialisation included potential transformation of inputs, and finalisation transformation of outputs – with constraints, such as avoiding the loss of information in outputs[1].

Almost in parallel, the additional output from the Mondex project that had been hinted at appeared. "More Powerful Data Refinement in Z" by Stepney, Cooper and Woodcock [28] was presented at the yearly Z conference. Its central observation was that in the derivation of Z refinement from relational rules, the *finalisation* was a powerful instrument for generalisation. In the standard approach, it would throw away the abstract state and any remaining inputs, and directly copy the output se-

[1] Our little joke was to call this the "every sperm is sacred" principle, in reference to Monty Python.

quence across to the global state. Changing the type of outputs was one obvious generalisation, and had indeed been necessary in the Mondex case study.

In our work on viewpoint specification and consistency, it had been clear from the beginning that we would need to be looking at reconciling behaviour-centred and state-centred specifications, as both of these were expected to be used in the Open Distributed Processing reference model [11]. We explored the cross-over between the world of Z and the world of process algebras in a variety of ways: translating LOTOS to Z [14], comparing the respective refinement relations [18], integrating CSP and Object-Z [26], and adding process algebra features such as internal operations to Z [17]. However, neither of these felt like the definitive solution or provided a comprehensive, let alone complete, refinement basis – until this thread of research was also infected by the derivation concept. Using relational data types, and their finalisations as making the correct observations (often: refusals) was a critical step forward, represented in a series of papers deriving concurrent refinement relations and simulation rules to verify them from relational characterisations, under the heading of "relational concurrent refinement" [15, 12].

2 Concrete State Machines with Anonymous Transitions

Our first book on refinement [16] continued from the work on generalising refinement that we had done to support viewpoint specification. It grew almost like a bunch of flowers, with nearly a new generalisation per chapter, plus an extra chapter of unopened buds, "Further Generalisations" that we had envisaged but did not develop in detail or with examples. Relational concurrent refinement gets a brief mention in the second (2014) edition of the book, as our preferred method of integrating state-focused and behaviour-focused methods.

We recently completed our second book on refinement [13], in which we take a rather different approach. We again conclude with relational concurrent refinement, but this time from a more inclusive perspective. We aimed to provide a comprehensive story of different refinement relations, mostly not of our own construction, and how they are related and reflected in existing formal methods and languages. In relating different refinement notions, of course we considered generalisation hierarchies as established by Van Glabbeek [21] and Leduc [24], but also the more conceptual relationships between them. In that dimension, it almost becomes a genealogy of refinement relations.

The first regular chapter in the new book [13] covers labeled transition systems as the obvious basic model for behavioural formalisms. There are states (including initial ones), and transitions between states, labelled with actions from some alphabet. Observations (traces, refusals, etc.) are in terms of these actions, and the states themselves contain no information beyond the behaviour from that point on.

When later in [13] we get to the basic relational refinement model that has been central to our work for the last twenty years, there is relevance both to states and to actions. Observations are defined via finalisation of the states at the end of a trace;

refinement is inclusion of such observations, universally qualified over all traces, where a trace consists of a sequence of actions. So these are *abstract* state machines (as finalisation modulates the state observations) with *visible* transitions – labelled with an action for every transition step.

Clearly that is a few steps away from the labeled transition model. How do we naturally get to that point? Looking ahead to explaining refinement in formalisms such as Event-B and ASM, how do we justify that these methods do not seem to care as much about the labels on transitions as Z (or labeled transition systems, for that matter) does? Will a deeper understanding of this improve our coverage of what "stuttering steps" and"refining skip" really mean?

We decided this called for a basic system model that is in some sense dual to labeled transition systems. Namely, we wanted a model in which the observations are based on *states*, and transitions do occur but have no individual meaning or observability other than through the effect they have on the state. So from a change of state we can draw the conclusion that "something must have happened" but no more than that, and in particular we also cannot assume the converse, that nothing can have happened if the state is unchanged between two observations.

Has such a model been described previously? It comes close to an abstract view of sequential programs, with possibilities for observation only crystal clear once the program has terminated. There are some candidates of formal methods in the literature which take related views, but they are all a bit more concrete than we would like in terms of the state spaces they assume. Action systems [4] have a rather concrete view of the state space, as being made up of variables, modified by assignments. The refinement theories of Abadi and Lamport [1] also have anonymous transitions, including stuttering ones, on a state space made up of variables. Hoare and He's Unifying Theories of Programming [23] (UTP) in their basic form come close to what we were looking for, also on state spaces made up of variables, although the better known variants of UTP are the ones with auxiliary variables encoding behaviour.

The model we defined has states, initial states, and a transition relation that only records that some states occur before some other states. We call it CSMAT: Concrete State Machine with Anonymous Transitions.

Definition 1 (CSMAT). A CSMAT is a tuple $(State, Init, T)$ where *State* is a non-empty set of states, $Init \subseteq State$ is the set of initial states, and $T \subseteq State \times State$ is a reflexive and transitive transition relation.

We write $p \longrightarrow_T q$ for $(p,q) \in T$, leaving out T when it is clear from the context. This is in close analogy with $\longrightarrow$ in LTSs, and at the same time also with $\Longrightarrow$ because T is reflexive and transitive, and thus equal to T^*.

In [13] we explain some, but not all, of the "design decisions" of this definition, and even then not in great detail. The intended contribution of this article is to highlight and explore these. Given that this is an artificial intermediate station in the theory development, all these decisions are up for discussion. Their best defence is if they provide some additional insight into refinement, or illuminate and foreshadow issues cropping up later in the theory development.

State machine: This is justifiable already as we have states and transitions. A restriction to *finite* states seems unnecessary here. Expressiveness matters, but will always be secondary in a basic model where anything complex will look clunky anyway; computability or Turing-completeness of the model is not an important concern. Some of the more eccentric problems in refinement, around infinite traces and possible unsoundness of upward simulation, disappear if our model does not allow for infinite branching.

We have not imagined models so abstract that they do not in some way contain that-what-is and that-what-happens, especially not when that-what-is is potentially represented by the possible futures, i.e. that-what-may-still-happen. Most machine models in theoretical computer science (Turing machines, stacks, registers, evaluation models for lambda calculus) are state machines with a particular structure of state anyway.

Non-empty set of states: This is a somewhat arbitrary choice – the trivial CSMAT-like structure with no states and hence no initial states or transitions is excluded. However ...

Initial states: We do not insist on the set of *initial states* being non-empty or even just a singleton. Allowing the empty set means we have a large collection of trivial state machines that would behave very interestingly if it wasn't for the fact they could never start; however, for a given transition relation, set inclusion on initial states might induce some lattice-like structure, and retaining an extremal element in that may be useful.

We had initially not been sure about allowing multiple initial states in the preceding chapter on LTSs. It seemed an unnecessary restriction to insist on a single initial state, but then we found that not doing so meant we needed to talk about internal versus external choice earlier than we wished to, and in a non-orthogonal way: LTSs with multiple initial states can be viewed as modelling the possibility of internal choice in initial states *only*. For CSMATs, the decision was forced towards multiple initial states by wanting a non-trivial notion of a state that could or would (not) lead to termination – effectively introducing *external* choice at initialisation *only*, which makes more sense for CSMATs than for LTSs, as we will explain below.

Non-determinism: As well as coming in via multiple initial states, non-determinism is implicitly present when transitions are characterised by a *relation*. Our excuse is that we want to use this model for refinement – if descriptions are deterministic, there is nothing left to refine[2].

Concrete: We call this a concrete state machine but after this single definition that remains entirely a statement of intent. Comparing the definition to that of

[2] One of our most enlightening paper rejections was one for a 1990s ZUM conference, where we had argued the opposite, namely that data refinement could introduce non-determinism, but a reviewer explained how this was entirely illusory, as such non-determinism could never be made visible in external observations. Of course this holds particularly for formal methods like Z where the final refinement outcome is only beholden to the initial specification and not to any detail introduced along the way like it is in for example Event-B [2], where refinement of deterministic systems can indeed be entirely meaningful.

labeled transition systems, we have merely removed information that *might* be observed: the labels on transitions. That abstraction by itself does not make the model concrete, of course.

The real contrast is with *abstract* state machines, as in the standard relational refinement theory, where the state is not directly observable – we signpost here that it will be the states themselves that will occur in observations, and definitions of observations of CSMATs will be seen to comply with that.

Anonymous: Transitions are *anonymous*, omitting the labels that are included in the LTS transition relation. As a consequence, virtually all of the notions of observation that LTSs provide and the refinement relations that are based on such observations become trivial in this model. At a deeper level this means that we should not look at this as a *reactive* model: we have removed the handle for the environment to be interacting with the system. This makes it a model of passive observation instead.

Transitive: Our reasoning for making the transition relation transitive is the thought that if we cannot observe transitions, this implies that we also cannot *count* transitions as individual steps. This is definitely a design choice where we could have gone the other way. Turing machines, for example, are not normally viewed as labeling their steps; but the associated theory of time complexity relies on being able to count them. A bit later in the theory development, the decision on transitivity will prove to have an adverse effect: it will not be preserved under abstraction functions.

If we are going to be looking at action refinement later, or at m-to-n simulation diagrams in ASM [25] where the labels do not matter so much, as we do later in [13], it is convenient to be able to move between looking at a single step and multiple steps.

The main justification for transitivity is closely tied to reflexivity. If we observe a system in a way that is (unlike Turing machine time complexity) not synchronised with the system's internal evolution, or maybe even in a continuous time model, there may be consecutive observations of the same state value. A transitive and reflexive transition relation between such observations allows us to not distinguish between the three different cases of this – which has to be the more abstract view.

Reflexive: These three cases are:

- nothing has happened;
- something has happened, but it is not visible at the level of abstraction we are observing the system at;
- multiple state changes have happened, returning us to the initial state.

Look at this as different versions of someone at a traffic light. The light was red, they blinked, and when they opened their eyes again it was red. The state of the traffic lights might be identical; their light might have remained red but the other flows of traffic might have changed in the meantime; or they might have blinked long enough for their light to go through an entire cycle.

Implicitly the third case will be noticeable in the transition relation anyway, as it also records what we would have seen if we had opened our eyes a little earlier. The first two really do not need to be distinguished. "Nothing happens" is often an abstraction anyway, for example the empty statement in a busy-waiting loop in a program is an abstraction of passing time.

The first two cases might be called "stuttering steps". The second case in particular relates to "refining skip", which serves a variety of roles in different formal methods, sometimes causing significant problems. We have analysed this previously [9, 10] and called the second case a "perspicuous" operation. At the more abstract level, the operation has no visible effect; but a refinement might provide some behaviour at a greater level of detail.

The book contains a separate chapter on perspicuous operations and whether and how they relate to internal operations and the consequences this has for refinement. It is also the place where we deal with livelock or divergence: the idea that nothing visible or externally controllable happens infinitely often.

Reflexivity of the transition relation has an important side effect: it means that the transition relation is *total*, i.e. from every state there is a possible "transition", if only to that state itself. So if we wanted to define a notion of a computation that stops for some (positive or negative) reason, we could not do that by finding states where T fails to define a next state.

No final states: Finite state machines (the ones that accept regular languages) have final or accepting states. It would be possible to add those to CSMATs, but then observations would have to respect that, at some cost of complexity of description. Looking forward to relational data types in later chapters having finalisations which are (typically) applicable in every state, we decided against it here.

Having explained our decisions in defining CSMATs, we now briefly consider the possible notions of observation that go with it and form the bases for refinement on CSMATs.

The most elementary of these simply characterises the states that are reachable in M, starting from a state in *Init* and following T.

Definition 2 (CSMAT observations). For a CSMAT $M = (State, Init, T)$ its observations are a set of states defined by

$$\mathcal{O}(M) = \{s : State \mid \exists\, init : Init \bullet (init, s) \in T\}$$

The standard method of deriving a refinement relation when the semantics generates sets is set inclusion:

Definition 3 (CSMAT safety refinement). For CSMATs $C = (S, CI, CT)$ and $A = (S, AI, AT)$, C is a safety refinement of A, denoted $A \sqsubseteq_S C$, iff $\mathcal{O}(C) \subseteq \mathcal{O}(A)$.

This is called "safety refinement" because the concrete system cannot end up in states that the abstract system disallows. In common with other safety-oriented refinement relations, doing nothing is always safe, so if either CI or CT is empty

then so is $\mathscr{O}(C)$ and hence (S, CI, CT) refines any CSMAT on the same state space S. Comparing only CSMATs on the same state space is based on a form of "type correctness", as the state doubles up ("concrete"!) as the space of observations.

Given reflexivity and totality of T, the best method we have come up with for characterising "termination" is the absence of *non-trivial* behaviour, so a terminating state is one which only allows stuttering, i.e. it is linked by T only to itself.

Definition 4 (CSMAT terminating states and observations). The terminating states and terminating observations of a CSMAT $M = (S, Init, T)$ are defined by

$$term(M) = \{s \in S \mid \forall t \in S \bullet (s,t) \in T \Rightarrow s = t\}$$
$$\mathscr{O}_T(M) = \mathscr{O}(M) \cap term(M)$$

Set inclusion on these observations we have called "partial correctness" as it is very close to that traditional correctness relation for programs: if the computation terminates, it delivers the correct results; and when it does not, we impose no constraints.

Definition 5 (CSMAT partial correctness refinement).
For CSMATs $C = (S, CI, CT)$ and $A = (S, AI, AT)$, C is a partial correctness refinement of A, denoted $A \sqsubseteq_{PC} C$, iff $\mathscr{O}_T(C) \subseteq \mathscr{O}_T(A)$.

To get a definition of "total correctness", we would normally have to add that the concrete computation is only allowed to not terminate whenever that is also allowed by the abstract one.

The concrete computation not (ever) terminating is characterised by $\mathscr{O}_T(C) = \varnothing$, and the same for the abstract computation is then $\mathscr{O}_T(A) = \varnothing$. The former condition should then imply the latter. But this is very much an all-or-nothing interpretation of termination, that does relate closely to ideas of termination and refinement in action systems and Event-B.

The word "whenever" in the informal description implies a quantification of some kind. If we take that over the entire (shared) state space, i.e. that a state must be a terminating one in the concrete system whenever the same state is terminating in the abstract system, this forces equality between the sets of terminating states in the refinement definition, so is not very useful. A different way of looking at it is that "whenever" implies a quantification over all initial states – and this invites a different view of what the different initial states represent.

Our definition of observations above only considers whether states (or terminating states) can be reached from *some* initial state. This implies that, when there are multiple initial states, we do not know or we do not care in which of these the computation started. Effectively, the CSMAT starts its operation by a (internal) nondeterministic choice of one of the possible initial states. We could also makes this an external choice: so different initial states represent a variable input to the system. This would make observations a relation between the initial state chosen and the final state observed – in other words, we get *relational* observations.

Thus, we can define relational observations that connect an initial state to another (final) state as follows.

Definition 6 (Relational observations of a CSMAT). For a CSMAT $M = (State, Init, T)$ its relational observations and terminating relational observations are relations defined as

$$\mathscr{R}(M) = (Init \times State) \cap T$$
$$\mathscr{R}_T(M) = (Init \times term(M)) \cap T$$

Analogous refinement relations can be defined using these observations.

Definition 7 (Relational refinements for CSMATs). For CSMATs C and A,

- C is a (relational) trace refinement of A, denoted $A \sqsubseteq_{RT} C$, iff $\mathscr{R}(C) \subseteq \mathscr{R}(A)$;
- C is a relational partial correctness refinement of A, denoted $A \sqsubseteq_{R} C$, iff $\mathscr{R}_T(C) \subseteq \mathscr{R}_T(A)$.

We can now extend partial correctness meaningfully to total correctness, see [13] for a calculation justifying the additional condition.

Definition 8 (Total correctness refinement for CSMATs). For CSMATs C and A, C is a total correctness refinement of A, denoted $A \sqsubseteq_{R} C$, iff $\mathscr{R}_T(C) \subseteq \mathscr{R}_T(A)$ and $\mathrm{dom}\,\mathscr{R}_T(A) \subseteq \mathrm{dom}\,\mathscr{R}_T(C)$.

Although we have now defined relational observations, we cannot yet use simulations to verify them. This is due to the state values being directly observable, i.e. we could only ever link fully identical states in a simulation anyway.

At this point we felt we had explored the space of meaningful refinement on CSMATs. ([13] also contains also a state trace variant of the semantics.) What are the baby steps that take us towards *abstract* data types?

First, observations restricted us to considering the same state space between concrete and abstract systems. Echoing our earlier work on output refinement, if we allowed ourselves to transform output types "at the edge of the system", we could relax that restriction. So what properties should such a transformation have? It should certainly apply to every possible "concrete" state, as otherwise we would have concrete observations that had no abstract counterparts. For a given concrete observation, we should also be able to reconstruct the corresponding abstract observation uniquely – this is the same "no information loss" principle for output transformations. As our observations *are* states, this together implies that the transformation is a total function from concrete to abstract states. Reassuringly, these tend to crop up in the most elementary definitions of simulations as well, for example in automata.

Can we define simulations between CSMATs on this basis? As it turns out, not quite – more on this below. Instead, we define the application of such a "state abstraction" on a CSMAT.

Definition 9 (State abstraction on CSMATs). Given a (total) function $f : S \to S'$, the state abstraction of a CSMAT $M = (S, Init, T)$ under f is the state machine $f(M) = (S', Init', T')$ defined by

$$Init' = \{f(s) \mid s \in Init\}$$
$$T' = \{(f(s), f(s')) \mid (s, s') \in T\}$$

Here is where we might regret an earlier design decision, namely transitivity. The state abstraction image is *not* necessarily a CSMAT, as its transition relation may not be transitive when the function is not injective. If states s_1 and s_2 have the same image under f, a path ending in s_1 may join up with a path beginning in s_2, creating a connection that may not have existed in the original CSMAT.

Injectivity of the abstraction function is not the correct fix for this. Elementarily, it would not establish an abstraction but merely a renaming, an isomorphism. In looking at the effect of an abstraction function on *reflexivity* we see why non-injectivity may actually be required. State abstraction does preserve reflexivity of the transition relation. Moreover, it can introduce stuttering steps in the abstraction that were actual changes of state in the original: when both the before state and the after state of a step are abstracted to the same state. This links a non-change in the abstracted machine to a change in the concrete machine, i.e. it highlights a perspicuous step as discussed above.

The effect on "termination" is problematic again, though. A system that never terminates, moving between its two states, can be abstracted to a one state system that by our definition of termination (no transitions except to itself) always terminates. Abstract systems are typically expected to terminate less often, rather than more often, than concrete ones in refinement relations. Similarly, by collapsing parts of the state space to a single point, unbounded concrete behaviour (possibly interpreted as divergence) can be collapsed to stuttering in the abstract model. Again, we would expect concrete systems to have less rather than more divergence.

A next step from considering abstraction functions in general is to look at the structure of the state space, and define specific abstraction functions from it. For example, if the state space is made up of the values of a fixed collection of named variables, projection onto the set of observable (global) variables is a meaningful abstraction function in the sense described above.

In [13], this particular refinement model has turned out to be illuminating when thinking about divergence, about internal operations and perspicuous operations – as well as when looking at refinement in notations (such as ASM, B, and Event-B) that do not fully conform to the abstract relational datatype model as used in Z.

Success in this undertaking at the most abstract theoretical level was not quite achieved. Ideally, we would have found a model M such that the abstract relational model is in some sense the minimal common generalisation of M and labeled transition systems. Maybe those models are just too subtly different. Time will tell. What we do know is that the theory of refinement in state-based languages is a lot richer than first appeared in the 1990s. Susan and colleagues' work in the late 90s initiated a line of thinking that has developed the theory and practice in a number of ways that were probably not foreseen when the first generalisations appeared.

References

1. Abadi, M., Lamport, L.: The existence of refinement mappings. Theoretical Computer Science **2(82)**, 253–284 (1991)
2. Abrial, J.R.: Modelling in Event-B. CUP (2010)
3. Ainsworth, M., Cruickshank, A.H., Wallis, P.J.L., Groves, L.J.: Viewpoint specification and Z. Information and Software Technology **36**(1), 43–51 (1994)
4. Back, R.J.R., Kurki-Suonio, R.: Distributed cooperation with action systems. ACM Transactions on Programming Languages and Systems **10**(4), 513–554 (1988)
5. Barden, R., Stepney, S., Cooper, D.: Z in Practice. BCS Practitioner Series. Prentice Hall (1994)
6. Boiten, E.: Z unification tools in Generic Formaliser. Tech. Rep. 10-97, Computing Laboratory, University of Kent at Canterbury (1997)
7. Boiten, E., Derrick, J.: IO-refinement in Z. In: A. Evans, D. Duke, T. Clark (eds.) 3rd BCS-FACS Northern Formal Methods Workshop. Springer-Verlag (1998). URL https://ewic.bcs.org/content/ConWebDoc/4354
8. Boiten, E., Derrick, J., Bowman, H., Steen, M.: Consistency and refinement for partial specification in Z. In: Gaudel and Woodcock [20], pp. 287–306
9. Boiten, E.A.: Perspicuity and granularity in refinement. In: Proceedings 15th International Refinement Workshop, *EPTCS*, vol. 55, pp. 155–165 (2011)
10. Boiten, E.A.: Introducing extra operations in refinement. Formal Aspects of Computing **26**(2), 305–317 (2014)
11. Boiten, E.A., Derrick, J.: From ODP viewpoint consistency to Integrated Formal Methods. Comput. Stand. Interfaces **35**(3), 269–276 (2013). DOI 10.1016/j.csi.2011.10.015. URL http://dx.doi.org/10.1016/j.csi.2011.10.015
12. Boiten, E.A., Derrick, J., Schellhorn, G.: Relational concurrent refinement II: Internal operations and outputs. Formal Aspects of Computing **21**(1-2), 65–102 (2009). URL http://www.cs.kent.ac.uk/pubs/2007/2633
13. Derrick, J., Boiten, E.: Refinement – Semantics, Languages and Applications. Springer (2018)
14. Derrick, J., Boiten, E., Bowman, H., Steen, M.: Viewpoints and consistency: translating LOTOS to Object-Z. Computer Standards and Interfaces **21**, 251–272 (1999)
15. Derrick, J., Boiten, E.A.: Relational concurrent refinement. Formal Aspects of Computing **15**(1), 182–214 (2003)
16. Derrick, J., Boiten, E.A.: Refinement in Z and Object-Z, 2nd edn. Springer-Verlag (2014). DOI 10.1007/978-1-4471-0257-1
17. Derrick, J., Boiten, E.A., Bowman, H., Steen, M.W.A.: Specifying and Refining Internal Operations in Z. Formal Aspects of Computing **10**, 125–159 (1998)
18. Derrick, J., Bowman, H., Boiten, E., Steen, M.: Comparing LOTOS and Z refinement relations. In: FORTE/PSTV'96, pp. 501–516. Chapman & Hall, Kaiserslautern, Germany (1996)
19. Flynn, M., Hoverd, T., Brazier, D.: Formaliser — an interactive support tool for Z. In: J.E. Nicholls (ed.) Z User Workshop, pp. 128–141. Springer London, London (1990)
20. Gaudel, M.C., Woodcock, J.C.P. (eds.): FME'96: Industrial Benefit of Formal Methods, Third International Symposium of Formal Methods Europe, *Lecture Notes in Computer Science*, vol. 1051. Springer-Verlag (1996)
21. van Glabbeek, R.J.: The linear time - branching time spectrum I. The semantics of concrete sequential processes. In: J. Bergstra, A. Ponse, S. Smolka (eds.) Handbook of Process Algebra, pp. 3–99. North-Holland (2001)
22. He Jifeng, Hoare, C.A.R., Sanders, J.W.: Data refinement refined. In: B. Robinet, R. Wilhelm (eds.) Proc. ESOP 86, *Lecture Notes in Computer Science*, vol. 213, pp. 187–196. Springer-Verlag (1986)
23. Hoare, C.A.R., Jifeng, H.: Unifying Theories of Programming. Prentice Hall (1998)
24. Leduc, G.: On the role of implementation relations in the design of distributed systems using LOTOS. Ph.D. thesis, University of Liège, Liège, Belgium (1991)

25. Schellhorn, G.: ASM refinement and generalizations of forward simulation in data refinement: A comparison. Theor. Comput. Sci. **336**(2-3), 403–435 (2005). DOI 10.1016/j.tcs.2004.11.013. URL http://dx.doi.org/10.1016/j.tcs.2004.11.013
26. Smith, G., Derrick, J.: Specification, refinement and verification of concurrent systems - an integration of Object-Z and CSP. Formal Methods in Systems Design **18**, 249–284 (2001)
27. Spivey, J.M.: The Z notation: A reference manual, 2nd edn. International Series in Computer Science. Prentice Hall (1992)
28. Stepney, S., Cooper, D., Woodcock, J.: More powerful data refinement in Z. In: J.P. Bowen, A. Fett, M.G. Hinchey (eds.) ZUM'98: The Z Formal Specification Notation, *Lecture Notes in Computer Science*, vol. 1493, pp. 284–307. Springer-Verlag (1998)
29. Woodcock, J., Stepney, S., Cooper, D., Clark, J., Jacob, J.: The certification of the mondex electronic purse to ITSEC Level E6. Formal Aspects of Computing **20**(1), 5–19 (2008). DOI 10.1007/s00165-007-0060-5. URL https://doi.org/10.1007/s00165-007-0060-5
30. Woodcock, J.C.P., Davies, J.: Using Z: Specification, Refinement, and Proof. Prentice Hall (1996)