Softw Syst Model (2017) 16:1083–1115
DOI 10.1007/s10270-015-0504-y

REGULAR PAPER

Towards an integrated formal method for verification of liveness properties in distributed systems: with application to population protocols

Dominique Méry[1] · **Michael Poppleton**[2]

Received: 15 November 2013 / Revised: 4 July 2015 / Accepted: 16 October 2015 / Published online: 29 December 2015

Abstract State-based formal methods [e.g. Event-B/RODIN (Abrial in Modeling in Event-B—system and software engineering. Cambridge University Press, Cambridge, 2010; Abrial et al. in Int J Softw Tools Technol Transf (STTT) 12(6):447–466, 2010)] for critical system development and verification are now well established, with track records including tool support and industrial applications. The focus of proof-based verification, in particular, is on safety properties. Liveness properties, which guarantee *eventual*, or converging computations of some requirements, are less well dealt with. Inductive reasoning about liveness is not explicitly supported. Liveness proofs are often complex and expensive, requiring high-skill levels on the part of the verification engineer. Fairness-based temporal logic approaches have been proposed to address this, e.g. TLA Lamport (ACM Trans Program Lang Syst 16(3):872–923, 1994) and that of Manna and Pnueli (Temporal verification of reactive systems—safety. Springer, New York, 1995). We contribute to this technology need by proposing a fairness-based method integrating temporal and first-order logic, proof and tools for modelling and verification of safety and liveness properties. The method is based on an integration of Event-B and TLA. Building on our previous work (Méry and Poppleton in Integrated formal methods, 10th international conference, IFM 2013, Turku, Finland, pp 208–222, 2013. doi:10.1007/978-3-642-38613-8_15), we present the method via three example population protocols Angluin et al. (Distrib Comput 18(4):235–253, 2006). These were proposed as a theoretical framework for computability reasoning about Wireless Sensor Network and Mobile Ad-Hoc Network algorithms. Our examples present typical liveness and convergence requirements. We prove convergence results for the examples by integrated modelling and proof with Event-B/RODIN and TLA. We exploit existing proof rules, define and apply three new proof rules; soundness proofs are also provided. During the process we observe certain repeating patterns in the proofs. These are easily identified and reused because of the explicit nature of the reasoning.

Keywords Refinement · Formal method · Distributed sytems · Verification · Liveness · Fairness

1 Introduction

Event-B/RODIN [7] is a leading, well-tooled formal method for critical systems development. Event-B is a state-based formal specification language in FOL, supported by the rich RODIN toolkit of provers, animator, model checkers, graphical modelling front-ends, and infrastructural support for composition–decomposition in development. Functional and safety verification is provided by automatically generated proof obligations (POs) for invariant preservation and refinement.

Lamport's temporal logic of actions (TLA) is a trace-based language for specifying both the structure of an action system and required properties of its behaviour: safety, liveness and fairness [38]. Its basis in temporal logic and a trace-

Communicated by Prof. Einar Broch Johnsen and Luigia Petre.

✉ Dominique Méry
dominique.mery@loria.fr

Michael Poppleton
mrp@ecs.soton.ac.uk

[1] LORIA, Université de Lorraine, BP 239, 54506 Vandoeuvre lès Nancy, France

[2] School of Electronics and Computer Science, University of Southampton, Highfield, Southampton SO17 1BJ, UK

based semantics makes specification of temporal properties straightforward to specify. The supporting proof system is well defined. There is a longstanding model-checker TLC [53] and a more recent proof tool TLAPS [34].

We present an integrated framework for temporal and first-order logic for modelling and verification of both safety and liveness properties. Our objective is to extend the scope of a refinement-based development method to prove liveness properties under fairness [38] assumptions. As experimental sandpit we choose the formal development of three simple population protocols [12]. The interesting questions about these protocols concern liveness and convergence properties and to what extent we can specify, reason about and prove such properties formally. A first-order scheme like Event-B cannot explicitly support this; we apply Lamport's TLA for such reasoning, since it allows the specification of trace-based properties.

The first two example protocols, the lights and the dancers [17], were presented in [58]. The lights gave a simple illustration of the proposed method and revealed the interplay of strong and weak fairness assumptions in proving convergence. The lights also showed how proof of a compound *leadsto* property under weak and strong fairness assumptions can be done first-order, e.g. by RODIN provers. The second protocol, the dancers, was extended beyond its source work [17] to give two stages of convergence. The second stage required intricate reasoning; a new proof rule GF1 based on *global* fairness [30] was sketched.

Here we give a more considered history of the development of the Event-B language, focussing on its capability for verification of liveness properties. Since we précis the fairness-based TLA, we give some background to fairness-based approaches. We formalise our framework for verification of safety and liveness in an integrated Event-B/TLA framework. We define diagrammatic proof rules PP-SF1 and PP-SF2 to capture styles of proof that arise in an example addressing the leader election protocol. We then present the lights with explicit liveness properties and proof through a refinement development.

The dancers development is presented in the same formalised manner to make liveness properties and proof more explicit. A diagrammatic representation of the case-split proof, through two stages of convergence, is given. A soundness proof for a new proof rule GF1 is given. A full proof of the complex convergence stage is given in the "Appendix".

The third protocol presented, self-stabilising leader election [37], is for the first time modelled and verified in a state-based formal method. It suggests a style of proof that might characterise self-stabilising systems. Here we demonstrate the utility of the diagrammatic proof rules PP-SF1 and PP-SF2.

Population Protocols: The design of a WSN or MANET is challenging, both functionally and in terms of quality assurance; Yick et al. [67] is a thorough recent survey of WSNs. Design requires demanding optimisation against, e.g. node power availability, message latency, throughput, per cent messages delivered, unpredictable node and communication reliability. Verification, given unreliable hardware and harsh operating environments, remains very challenging. From the perspective of Software Engineering, the "code-and-fix" nature of WSN development was identified [62]; in response the Software Engineering for Sensor Network Applications (SESENA) workshop series of ICSE was established in 2010.

A recent and relevant theoretical response to this need is the population protocol (PP) [12,17]. It "aims to represent sensor networks consisting of tiny computational devices with sensing capabilities that follow some unpredictable and uncontrollable mobility pattern" [65]. The PP is a model of simple agents, passively mobile in the sense of Angluin et al. [12], where an agent does not determine a priori who it interacts with. A dynamic interaction graph models mobility, under control of some external adversarial scheduler. The graph determines which pairs of agents are *adjacent*, i.e. may interact at any given time. The agent is simple and can be described with a finite-state machine. It has minimal storage, which is independent of network size—this is the *uniformity* property of PP. An agent has no unique identifier—the *anonymity* property. The third distinguishing characteristic of PP within distributed systems is the inherent *nondeterminism* of interaction between agents.

We consider a finite set V of agents with an irreflexive relation $E \subseteq V \times V$ defining the *interaction graph*, which specifies which agent pairs may interact. In the basic PP model, each agent establishes an initial state by reading a single input. Each agent produces an externally observable output as a function of its state. After initialisation, agents interact pairwise, atomically and instantaneously. The choice of which pair to interact at any time is nondeterministic. During an interaction, each participating agent will update its state and thus its output.

The basic population protocol over (V, E) is the tuple (Q, X, I, Y, O, δ), where

- Q is the finite set of states for each agent
- X is a finite alphabet of inputs, and $I : X \to Q$ maps each input to an initial state
- Y is the finite output alphabet, and $O : Q \to Y$ maps each state to an output
- transition relation $\delta : Q \times Q \to Q \times Q$ defines the interaction of two agents

A *configuration* is a function $C : V \to Q$ giving the state of each agent. We say that a configuration C leads to a configuration C', i.e. $C \to C'$, when C moves through some single δ-interaction between two agents to C'. That is,

for two interacting agents u, v, C' is precisely C updated with $\{u \mapsto \delta_1(u, v), v \mapsto \delta_2(u, v)\}$. We consider a *trace* of the PP to be the sequence of configurations C_i associated with some sequence of interactions. The aim is to prove that the corresponding output trace $O(C_i)$ converges. Note that the trace itself need not converge, but the output trace must.

The interaction of any two agents is under a *global fairness* assumption [30], which expresses that a trace $T = C_0 \rightarrow C_1 \rightarrow \cdots$ is globally fair, when for every configuration C and C' such that $C \rightarrow C'$, if $C = C_i$ for infinitely many i in T, then $C_{i+1} = C'$ for infinitely many i. This globally fair interaction happens in the *complete* interaction graph in the basic PP; this assumption on the graph has been relaxed in subsequent work. Existing work mostly makes stronger, probabilistic assumptions [13], e.g. in the basic model assuming that interacting pairs are scheduled randomly, independently and uniformly gives a *conjugating automaton* which converges with probability 1.

Various extensions of the basic model, bringing it closer to real-world requirements in various ways, have been proposed: instantaneous two-way interaction is replaced with one-way anonymous message-passing, immediate or delayed delivery, recording of sent messages, and queuing of incoming messages [13]. In a *self-stabilising* system [14] the protocol acts on input streams, mimicking sensor nodes.

Fairness, Scheduling and Methodology: In our work, the nondeterministic interaction of a PP chosen by one or more adversarial but *fair schedulers* is the focus. Chatzigiannakis et al. [33] have examined the performance of large PPs in simulation experiments, using various fair scheduler designs and interaction graph assumptions. This complements the mostly theoretical work to date and emphasises the importance of fairness and scheduling in this domain.

In closed system modelling, the allocation of model elements to device versus environment is a key early stage. For example, for the WSN node software developer, radio transceiver, sensor and A2D circuits on the node are environment elements.[1] Following established rely/guarantee practice [48] we make *assumptions* about the environment and *assertions* about the devices in this setting; we must demonstrate or prove these assertions.

There is a rich history of fairness and scheduling notions in methods for modelling and reasoning about distributed and concurrent systems. Apt and Olderog's early proof-theoretic approach [15,59] applied weak and strong fairness for Dijkstra's nondeterministic do-od programs. Lamport's TLA [50–52] uses fairness to prove liveness properties as originally defined by Alpern and Schneider [10,11]. While TLA allows a notion of refinement and provides intricate rules for fairness refinement, this is not straightforward and not much used.

UNITY [32] is a similar methodology, including refinement and a programming notation based on action systems [19] and a temporal logic-based specification language. The goal was a method to derive concurrent/distributed solutions, proving safety and liveness under implicit weak fairness assigned to each action. A UNITY action is executed under weak fairness assumption and is non-blocking.

The evolution of classical B [3] into Event-B was inspired inter alia by CSP [44], action systems, DISCO [18,47], TLA and Manna and Pnueli [54]. Event-B is a state transition language in FOL whose usual semantic model is the labelled transition system (LTS). Its strength in verification is in safety properties. The evolution from classical B was motivated in part by the need to model and assure certain liveness properties or "dynamic constraints" [8]. In liveness, *leadsto* properties are key: $P \rightsquigarrow Q$ means that whenever P holds, Q is guaranteed to hold at some later point. Leadsto properties were to be made syntactically explicit in a MODALITIES clause: each modality would implement a $P \rightsquigarrow Q$ leadsto property as a loop structure naming participating events and generating associated loop POs. In the eventual Event-B language the treatment of liveness is more implicit; the proof system only has weak support for liveness. A simple $P \rightsquigarrow Q$ property can be modelled by a new, skip-refining iterative event e that eventually terminates, having established the guard for event f, which establishes Q. A VARIANT expression, which must be proved to be reduced by all new events, encodes the termination of such new events as a simple induction proof. The deadlock freedom PO for a model—that the guard of at least one event is always enabled—encodes a weak fairness assumption over the disjunction of all events.

Our approach is in the spirit of an earlier unifying model [39] for different formal methods. In that work an example showed the integration of a temporal specification plus liveness proof in TLA, with a refinement-based action systems model in UNITY proving invariance, i.e. safety properties. In a similar way we combine specification and proof in TLA and Event-B. Since a TLA model specifies both the action system and its required properties, we interpret the Event-B model [16] as the action system and prove liveness under fairness assumptions in TLA, using Event-B/RODIN in the normal way for safety proof. Previous work on patterns for refinement-based algorithm development [57] has also influenced this work.

Event-B does not address fairness in any explicit way, which is understandable in an action systems scheme with

[1] We do not consider the hybrid interface of this discrete model to the wireless radio environment.

LTS but not trace semantics. The usual practice is to assume that some scheduler external to the Event-B model chooses fairly between enabled events at any time. To encode progress, the scheduler's hand is forced by restricting enablement of non-priority events with flags or counters. While this is a reasonable informal approach based on modelling heuristics to ensure progress, it is not a proof method. Also, this approach is more questionable when applied to environment components. Modelling control problems in this manner [29] involves disabling a sensor event after reading environment data until the control action is completed. While this approach is consistent with periodic sensor reading at most once per processor execution cycle, it relies on engineering judgement about whether the partial, approximate view the controller has of the environment is sufficiently accurate and safe.[2]

The implicit Event-B approach to progress and liveness relies on heuristics and experience in such modelling and design. Such expertise is only transferrable by study of best-practice models and will be expensive to develop in engineers. We propose an explicit method for progress and liveness, based on classical TLA specification of liveness and fairness requirements of the devices under design, and fairness assumptions on the environment. Concrete scheduling design can be undertaken at a suitable point and verified to meet the device fairness requirements. Proof support is available [34].

In contrast to related work, we make no language extensions. In classical B, [41] encodes an LTL property, guaranteed by construction, as a Büchi automaton within the B model. The incorporation of the property in this form complicates verification. In the same spirit as the original idea [8] for liveness modelling in Event-B, Hoang and Abrial [43] extend the sequent calculus over new POs to define liveness proof rules. New variant-based (thus, implicitly inductive) POs encode notions of "convergence" and "divergence". Together with deadlock freedom (implicitly, weak fairness), these POs are assembled into liveness proof rules for $\Box\Diamond$ (always eventually), $\Diamond\Box$ (eventually always) and $\rightsquigarrow$ (leadsto) properties. Building on implicitly encoded inductive and fairness logic, an apparently complex proof system results. Illustrative future work will be to compare a proved example in this method, with ours.

Hoang and Abrial [43] extend the proof system, generating an implicit trace semantics. They exploit the sparseness of the Event-B language and its basic set of POs which makes it a flexible and extensible foundation for richer semantic interpretations [42]. On the other hand, Hudon and Hoang's Unit-B [45] is a more ambitious language extension of Event-B inspired by UNITY. It gives a Dijkstra's computation calculus [35] semantics to Event-B and enables fairness assumptions to be specified at event level. Each event has both a coarse (weak fairness assumption) and a fine schedule (strong fairness assumption). A liveness-preserving refinement scheme is presented. This is effectively a new language and proof system proposal with all the associated development costs. Finally, Schneider et al. [64] characterise a class of LTL properties that are preserved by Event-B refinement.

In the next section we introduce the syntax, specification and integrated proof scheme for TLA and Event-B. In some detail we give a proof scheme for liveness properties and their refinement; we prove a composite proof rule PP-SF1 and outline its extended form as PP-SF2. Sections 3 and 4 then overview Event-B developments for two example population protocols from the literature, respectively. Section 3 gives the lights, a simple example protocol over a dynamic interaction graph revealing a reusable proof pattern. Section 4 gives the more complex dancers example, extended from the literature, and proposes a new notion of general fairness to prove convergence. A significant new proof rule GF1 is defined and proved sound. Section 5 gives the leader election example. In this self-stabilising system an elaboration of the proof pattern of Sect. 3 is revealed; we apply the composite rules PP-SF1 and PP-SF2 and show how to address the proof of self-stabilisation. Sect. 6 concludes.

2 TLA and Event-B

Leslie Lamport's TLA (**Temporal Logic of Actions**) [51] is designed for the specification and verification of reactive systems in terms of their actions and behaviours (traces). It can be thought of as structured in four *tiers* [2]: (i) constants and *constant formulas*—functions and predicates—over these, (ii) *state formulas* for reasoning about states, expressed over *variables* as well as constants, (iii) *transition* or *action formulas* for reasoning about (before-after) pairs of states and (iv) *temporal predicates* for reasoning about *behaviours*, i.e. traces of states; these are constructed from the other tiers and certain *temporal operators*.

An action formula expresses some fact or function about a system transition between one state and its successor, as made available by some system action. An action predicate is very like a before–after predicate in Event-B. A state formula is an action formula where either all flexible variables are unprimed or all are primed. A state predicate is true in a behaviour iff it is true in the first

[2] Note that the more sophisticated modelling of the hybrid discrete/continuous controller/environment interface offered by the Hybrid Systems community is under active consideration by the Event-B community [9,20–22].

state of that behaviour. If F, G are temporal predicates, then so are $\neg F, F \vee G, F \wedge G, F \Rightarrow G, \Box P, \Diamond P$. The latter two are temporal operators. We write $\Box P$—called "always P"—to mean P is always true over a given behaviour and define $\Diamond P$—called "eventually P"—to be $\neg\Box\neg P$.

For action predicate A, state formula f (usually, a list of state variables) we define $[A]_f$ (called "square A sub f") to be true for states s, t iff $s[\![A \vee f' = f]\!]t$, that is, if either A defines a transition from s to t, or variables f remain unchanged from s to t. Dually we define $\langle A\rangle_f$ (called "angle A sub f") to be true for states s, t iff $s[\![A \wedge f' \neq f]\!]t$, that is, A defines a transition from s to t, *and* state f changes from s to t.

This logic enables us to specify the state transition-based behaviour of a system, as well as assert properties over that behaviour, in one notation and logic. In general we wish to specify systems in the form

$$\Phi \mathrel{\widehat{=}} Init_\Phi \wedge \Box[Next]_f \wedge \mathrm{WF}_f(N_1) \wedge \mathrm{SF}_f(N_2)$$

where $Next \mathrel{\widehat{=}} N_1 \vee N_2 \vee \ldots$ is the disjunction of all system actions, i.e. the "next" transition, denoting progress subject to possible stuttering. Stuttering is required to allow us to specify and prove refinements. The WF and SF constraints are the weak and strong fairness constraints required by the system actions in order to progress.

Consideration of whether an action eventually stabilises to always enabled or not determines the choice of a weak or strong fairness requirement in specification. We say that action A is *weakly fair* if, provided it is eventually always enabled, it is then guaranteed to fire infinitely often. Alternatively, it may be infinitely often disabled and never fire. A is *strongly fair* if, provided it is infinitely often enabled, it is then guaranteed to fire infinitely often. Alternatively, it may be eventually always disabled and never fire. With the weaker antecedent in its implicative form, SF is the stronger fairness property. *General* fairness GF is defined as a stronger form of strong fairness. Figure 1 gives the fairness properties.

Finally, the *leadsto* operator: $P \rightsquigarrow Q \mathrel{\widehat{=}} \Box(P \Rightarrow \Diamond Q)$ states that whenever P holds then Q is guaranteed to hold at some later time ... eventually.

$$\begin{aligned}
\mathrm{WF}_f(A) &\mathrel{\widehat{=}} \Diamond\Box Enabled\langle A\rangle_f \Rightarrow \Box\Diamond\langle A\rangle_f \\
&\equiv \Box\Diamond\neg Enabled\langle A\rangle_f \vee \Box\Diamond\langle A\rangle_f \\
\mathrm{SF}_f(A) &\mathrel{\widehat{=}} \Box\Diamond Enabled\langle A\rangle_f \Rightarrow \Box\Diamond\langle A\rangle_f \\
&\equiv \Diamond\Box\neg Enabled\langle A\rangle_f \vee \Box\Diamond\langle A\rangle_f \\
\mathrm{GF}_f(P, A, Q) &\mathrel{\widehat{=}} \Box\Diamond Enabled\langle P \wedge A \wedge Q'\rangle_f \Rightarrow \Box\Diamond\langle P \wedge A \wedge Q'\rangle_f \\
&\equiv \Diamond\Box\neg Enabled\langle P \wedge A \wedge Q'\rangle_f \vee \Box\Diamond\langle P \wedge A \wedge Q'\rangle_f
\end{aligned}$$

Fig. 1 Fairness properties in TLA

Figure 2 presents some of Lamport's proof rules for simple TLA [51] and three lemmas we require—IMPLICATION, REWRITING and SIMPLIFICATION. This list is semantically complete for liveness proof in TLA. TRANSITIVITY and CONFLUENCE are well-known rules for manipulating leadsto properties. LATTICE is an induction rule. Provided H_c leads to either the goal G or H_d for some d strictly smaller than c and then the induction is guaranteed to converge to G. WF1 gives the conditions under which weak fairness of action A is enough to guarantee that $P \rightsquigarrow Q$. A stuttering *progress step* produces either P or Q in the next state, nonstuttering action $\langle A\rangle_f$ takes the *inductive step* to produce Q, and under P, *inductive action* $\langle A\rangle_f$ is always enabled. SF1 is the strong fairness equivalent to prove $P \rightsquigarrow Q$: a strong fairness assumption on A is made and the same first two conditions hold as in WF1. A third condition elaborated with a fairness assumption $\Box F$ ensures that $\langle A\rangle_f$ is *eventually*—rather than always—enabled.

Liveness properties under fairness assumptions require the application of one of the two rules WF1 and SF1 in Fig. 2. We aim to facilitate the derivation of liveness properties using these rules, when considering population protocols. While constructing proofs, we identify specific patterns for improving the proof process and for helping the modeller to identify what fairness assumption should be assigned to each event. In the next subsection, we introduce two proof rules in diagrammatic form, to clarify their intended usage. We will apply these rules later, in the example sections.

2.1 Two practical proof rules as diagrams

The two rules WF1 and SF1 may be annotated by special diagrams corresponding to specific patterns of use of these rules. We first specialise the SF1 rule as follows. We formalise it as a new proof rule PP-SF1, using the rules of Fig. 2 and give a diagrammatic form in Fig. 3. We assume the following: (i) P_1, P_2 and Q are three assertions of state properties of a system specified by N and (ii) $N \equiv \alpha \vee \beta \vee \gamma \vee \delta \vee \epsilon$ where $\alpha, \beta, \gamma, \delta, \epsilon$ are modelling the possible events modifying state variables x.

– PROVE: $\Box[N]_x \wedge \mathrm{SF}_x(\gamma) \wedge \mathrm{WF}_x(\alpha) \Rightarrow (P \rightsquigarrow Q)$
PROOF:

$\Box[N]_x \wedge \mathrm{WF}_x(\alpha) \Rightarrow P_1 \rightsquigarrow P_2$	1 ... H1-3, WF1
$\Box[N]_x \wedge \mathrm{WF}_x(\alpha) \Rightarrow P \rightsquigarrow P_2$	2 ... 1, CONFLUENCE
$\Box P \wedge \Box[N]_x \wedge \mathrm{WF}_x(\alpha) \Rightarrow \Diamond P_2$	3 ... 2, REWRITING
$\Box P \wedge \Box[N]_x \wedge \mathrm{WF}_x(\alpha) \Rightarrow \Diamond Enabled\langle\gamma\rangle_x$	4 ... 3, H3b, temporal logic
$\Box[N]_x \wedge \mathrm{SF}_x(\gamma) \wedge \mathrm{WF}_x(\alpha) \Rightarrow (P \rightsquigarrow Q)$	5 ... 4, H4-5, SF1 □

Implicitly we start with a WF1 proof (H1-3): since P_1 is the starting state, weak fairness of α suffices to show $P_1 \rightsquigarrow P_2$. On the other hand, β returns P_2 to P_1, disabling γ. The

Fig. 2 Main rules for liveness properties in TLA

IMPLICATION.

$$\frac{F \Rightarrow (P \Rightarrow Q)}{F \Rightarrow (P \rightsquigarrow Q)}$$

REWRITING.

$$\frac{F \Rightarrow (P \rightsquigarrow Q)}{F \Rightarrow (\Box P \Rightarrow \Diamond Q)}$$

SIMPLIFICATION.

$$\frac{F \Rightarrow (P \wedge I \rightsquigarrow Q) \quad F \Rightarrow \Box I}{F \Rightarrow (P \rightsquigarrow Q)}$$

TRANSITIVITY.

$$\frac{F \Rightarrow (P \rightsquigarrow Q) \quad F \Rightarrow (Q \rightsquigarrow R)}{F \Rightarrow (P \rightsquigarrow R)}$$

CONFLUENCE.

$$\frac{\forall i \in I : F \Rightarrow (P_i \rightsquigarrow Q)}{F \Rightarrow ((\exists i \in P_i) \rightsquigarrow Q)}$$

LATTICE. $\succ$ a well-founded partial order on a set S

$$\frac{F \wedge c \in S \Rightarrow (H_c \rightsquigarrow (G \vee \exists d \in S \cdot (c \succ d) \wedge H_d))}{F \Rightarrow ((\exists c \in S \cdot H_c) \rightsquigarrow G)}$$

FAIRNESS.

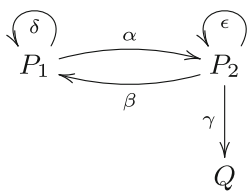
WF1.

$$\frac{P \wedge [N]_f \Rightarrow (P' \vee Q') \quad P \wedge \langle N \wedge A \rangle_f \Rightarrow Q' \quad P \Rightarrow Enabled\langle A \rangle_f}{\Box[N]_f \wedge \mathrm{WF}_f(A) \Rightarrow (P \rightsquigarrow Q)}$$

SF1.

$$\frac{P \wedge [N]_f \Rightarrow (P' \vee Q') \quad P \wedge \langle N \wedge A \rangle_f \Rightarrow Q' \quad \Box P \wedge \Box[N]_f \wedge \Box F \Rightarrow \Diamond Enabled\langle A \rangle_f}{\Box[N]_f \wedge \mathrm{SF}_f(A) \wedge \Box F \Rightarrow (P \rightsquigarrow Q)}$$

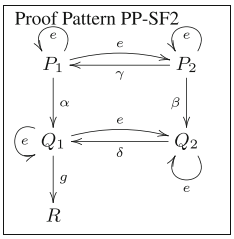
Fig. 3 PP-SF1 proof rule



$$\begin{array}{rl} H0 & P \mathrel{\widehat{=}} P_1 \ \vee \ P_2 \\ H1 & P_1 \wedge [N]_x \Rightarrow (P_1' \vee P_2') \\ H2 & P_1 \wedge \langle N \wedge \alpha \rangle_x \Rightarrow P_2' \\ H3a,b & P_1 \Rightarrow Enabled\langle \alpha \rangle_x \ ; \ P_2 \Rightarrow Enabled\langle \gamma \rangle_x \\ H4 & P \wedge [N]_x \Rightarrow \ (P' \ \vee \ Q') \\ H5 & P \wedge \langle N \wedge \gamma \rangle_x \Rightarrow Q' \\ \hline C & \Box[N]_x \wedge \mathrm{SF}_x(\gamma) \wedge \mathrm{WF}_x(\alpha) \Rightarrow (P \rightsquigarrow Q) \end{array}$$

third hypothesis of SF1 (proof line 4), guaranteeing eventual enablement of γ, is derived from $P \rightsquigarrow P_2$ and H3b. This PP-SF1 proof pattern simplifies the proofs and shows the role of α in proving convergence.

Now we consider another possible pattern of predicates related through events and we define the PP-SF2 rule as follows.



$$
\begin{array}{rl}
H0 & P \mathrel{\widehat{=}} P_1 \vee P_2 \vee Q_1 \vee Q_2 \\
H1a & P_1 \wedge [N]_x \Rightarrow (P_1' \vee P_2' \vee Q_1') \\
H1b & P_2 \wedge [N]_x \Rightarrow (P_1' \vee P_2' \vee Q_2') \\
H2a & P_1 \wedge \langle N \wedge \alpha \rangle_x \Rightarrow Q_1' \\
H2b & P_2 \wedge \langle N \wedge \beta \rangle_x \Rightarrow Q_2' \\
H2c & Q_2 \wedge [N]_x \Rightarrow (Q_2' \vee Q_1') \\
H3a,b & P_1 \Rightarrow Enabled\langle \alpha \rangle_x \; ; \; P_2 \Rightarrow Enabled\langle \beta \rangle_x \\
H3c,d & Q_1 \Rightarrow Enabled\langle g \rangle_x \; ; \; Q_2 \Rightarrow Enabled\langle \delta \rangle_x \\
H3e & P_2 \Rightarrow Enabled\langle \gamma \rangle_x \\
H4a & P_2 \wedge \langle N \wedge \gamma \rangle_x \Rightarrow P_1' \\
H4b & P_2 \wedge \langle N \wedge \beta \rangle_x \Rightarrow Q_2' \\
H4c & Q_2 \wedge \langle N \wedge \delta \rangle_x \Rightarrow Q_1' \\
H5 & Q_1 \wedge \langle N \wedge g \rangle_x \Rightarrow R' \\
\hline
C & \Box[N]_x \wedge \mathrm{WF}_x(\gamma) \wedge \mathrm{WF}_x(\beta) \wedge \mathrm{WF}_x(\delta) \wedge \mathrm{SF}_x(\alpha) \wedge \mathrm{SF}_x(g) \Rightarrow (P \rightsquigarrow R)
\end{array}
$$

- PROVE: $\Box[N]_x \wedge \mathrm{WF}_x(\gamma) \wedge \mathrm{WF}_x(\delta) \wedge \mathrm{SF}_x(\alpha) \wedge \mathrm{WF}_x(\beta) \Rightarrow (P \rightsquigarrow R)$

  PROOF:

  1 $P_2 \wedge [N]_x \Rightarrow (P_2' \vee P_1' \vee Q_2')$
  H1b
  2 $P_2 \wedge \langle N \wedge \gamma \rangle_x \Rightarrow P_1'$
  $H4a$
  3 $P_2 \wedge \langle N \wedge \beta \rangle_x \Rightarrow Q_2'$
  $H4b$
  4 $P_2 \Rightarrow Enabled\langle \gamma \rangle_x$
  $H3c$
  5 $P_2 \Rightarrow Enabled\langle \beta \rangle_x$
  $H3b$
  6 $\Box[N]_x \wedge \mathrm{WF}_x(\gamma) \wedge \mathrm{WF}_x(\beta) \Rightarrow P_2 \rightsquigarrow (P_1 \vee Q_2)$
  1,2,3,4,5 with WF1
  7 $Q_2 \wedge [N]_x \Rightarrow (Q_2' \vee Q_1')$
  $H2c$
  8 $Q_2 \Rightarrow Enabled\langle \delta \rangle_x$
  $H3d$
  9 $Q_2 \wedge \langle N \wedge \delta \rangle_x \Rightarrow Q_1'$
  $H4c$
  10 $\Box[N]_x \wedge \mathrm{WF}_x(\delta) \Rightarrow (Q_2 \rightsquigarrow Q_1)$
  7,8,9 with WF1
  11 $(P_1 \vee P_2) \wedge [N]_x \Rightarrow (P_1' \vee P_2' \vee Q_1' \vee Q_2')$
  $H1a, H1b + REWRITING$
  12 $(P_1 \vee P_2) \wedge \langle N \wedge \alpha \rangle_x \Rightarrow Q_1'$
  $H2a + REWRITING$
  13 $(P_1 \vee P_2) \wedge \langle N \wedge \alpha \rangle_x \Rightarrow (Q_1' \vee Q_2')$
  $LOGICAL\ RULE$
  14 $\Box(P_1 \vee P2) \wedge \Box[N]_x \wedge \mathrm{SF}_x(\alpha) \Rightarrow \Diamond Enabled\langle \alpha \rangle_x$
  $TEMPORAL\ DERIVATION$
  15 $\Box[N]_x \wedge \mathrm{SF}_x(\alpha) \Rightarrow (P_1 \vee P_2) \rightsquigarrow (Q_1 \vee Q_2)$
  11,12,13,14 + SF
  16 $\Box[N]_x \wedge \mathrm{SF}_x(\alpha) \wedge \mathrm{WF}_x(\beta) \wedge \mathrm{WF}_x(\gamma) \wedge \mathrm{WF}_x(\delta) \Rightarrow (P_1 \vee P_2 \vee Q_2) \rightsquigarrow Q_1$
  6,10,15 + REWRITING
  17 $\Box(Q_1 \vee Q_2) \wedge [N]_x \Rightarrow (Q_1' \vee Q_2' \vee R')$
  $H2c, H4c, H5 + REWRITING$
  18 $(Q_1 \vee Q_2) \wedge \langle N \wedge g \rangle_x \Rightarrow R'$
  $H5 + REWRITING$
  19 $\Box(Q_1 \vee Q_2) \wedge \Box[N]_x \wedge \mathrm{SF}_x(g) \Rightarrow \Diamond Enabled\langle g \rangle_x$
  $SFproperties$
  20 $\Box[N]_x \wedge \mathrm{SF}_x(g) \Rightarrow ((Q_1 \vee Q_2) \rightsquigarrow R)$
  17, 18, 19 + SF
  21 $\Box[N]_x \wedge \mathrm{SF}_x(g) \wedge \mathrm{SF}_x(\alpha) \wedge \mathrm{WF}_x(\delta) \wedge \mathrm{WF}_x(\gamma) \wedge \mathrm{WF}_x(\beta) \Rightarrow (P \rightsquigarrow R)$16,20 □

This rule will be applied in Sect. 5.4; its proof is similar to that of PP-SF1. In fact, it emerged while we were proving the leader election protocol. The choice of the fairness assumption is due to the possible activation of e which is delaying the convergence. The patterns are a way to drive the development of protocols because one can introduce *phases* in the development. One can also think on a general result for helping users to derive proofs of liveness properties using fairness assumptions which may be very *exotic* when reading protocol descriptions.

2.2 Structures for EVENT B models

Event-B is designed for long-running *reactive* hardware/software systems that respond to stimuli from user and/or environment. In this set-theoretic language in FOL, guarded *events* provide state transition behaviour. The usual semantic model is the LTS. The two syntactic units of structuring are the static *context* and the dynamic *machine*. The

context is the static part of the model, comprising sets, constants, axioms and any theorems that must be derived from those axioms. The machine is the dynamic part, comprising dynamic variables and the events that update them. Safety properties are expressed as either invariants or theorems. Every machine *sees* at least one context.

An event e acting on (a list of) state variables v, subject to enabling *guard* over local variable(s) t and state-updating *action*, has the following syntax and semantics. We call the latter a *before–after predicate*:

$$e \mathrel{\widehat{=}} \text{ANY } t \text{ WHERE } Q(t,v) \text{ THEN } v := F(t,v) \text{ END}$$
$$BA(e)(v,v') \mathrel{\widehat{=}} \exists t \cdot (Q(t,v) \wedge v' = F(t,v))$$

This defines a t-indexed nondeterministic choice between those transitions $v' = F(t, v)$ for which $Q(t, v)$ is true. t can be interpreted as either an input or an output to the event.

An event works in a model with constants c and sets s subject to *axioms* $P(s, c)$ and an *invariant* $I(s, c, v)$. Consistency POs require that events are well-defined, feasible and maintain invariants. The term *refinement* is overloaded, referring both to the process of transforming models and to the more concrete model which refines the abstract one. When model $N(w)$ refines $M(v)$, it contains a refinement relation, or "gluing invariant" $J(s, c, v, w)$. New events may be introduced in refinement to act on new variables, effectively refining stuttering steps (called "skip" in Event-B). The refinement POs enforce the standard forward simulation refinement rule [4] that every concrete step of a refining event reestablishes the gluing invariant subject to some corresponding step of the abstract refined event, or skip.

In this work, the modelling process deals with various languages, as seen by considering the triptych of Bjoerner [24–27]: $\mathcal{D}, \mathcal{S} \longrightarrow \mathcal{R}$. Here, the domain $\mathcal{D}$ deals with properties, axioms, sets, constants, functions, relations and theories. The system model $\mathcal{S}$ expresses a model or a refinement-based chain of models of the system. Finally, $\mathcal{R}$ expresses requirements for the system to be designed.

2.3 Contexts

The first structure is called a context ($\mathcal{D}$ in Fig. 4), and it defines sets, constants, axioms and theorems derivable from those axioms. The abstract context $\mathcal{AD}$ is a previous context that has already been defined, and it is seen by the current context $\mathcal{D}$. A context is validated when sets $S_1, \ldots, S_n$, constants $C_1, \ldots, C_m$ and axioms $ax_1, \ldots, ax_p$ are well formed and when all theorems $th_1, \ldots, th_q$ are proved.

A context clearly states the static properties of the (system) model under construction. The SEES construct enables re-use by extending a previously defined context.

The proof process is based on the management of sequents, with an associated environment for proof called $\Gamma(\mathcal{D})$. The proof environment includes axioms, properties and theorems already proved. An environment is initially provided, but the intention is to add new theorems. This means that we intend to prove the following properties in the sequent calculus style:

For any j in $\{1..q\}$, $\Gamma(\mathcal{D}) \vdash th_j : Q_j(S_1, \ldots S_n, C_1, \ldots, C_m)$

Theorems for the context are proved using the RODIN tool, but it is clear that the process for constructing the domain $\mathcal{D}$ is crucial to modelling the system, from consid-

Fig. 4 Context and machine

CONTEXT $\mathcal{D}$
EXTENDS $\mathcal{AD}$
SETS
$S_1, \ldots S_n$
CONSTANTS
$C_1, \ldots, C_m$
AXIOMS
$ax_1 : P_1(S_1, \ldots S_n, C_1, \ldots, C_m)$
$\ldots$
$ax_p : P_p(S_1, \ldots S_n, C_1, \ldots, C_m)$
THEOREMS
$th_1 : Q_1(S_1, \ldots S_n, C_1, \ldots, C_m)$
$\ldots$
$th_q : Q_q(S_1, \ldots S_n, C_1, \ldots, C_m)$

MACHINE $\mathcal{M}$
REFINES $\mathcal{AM}$
SEES $\mathcal{D}$
VARIABLES x
INVARIANTS
$inv_1 : I_1(x, S_1, \ldots S_n, C_1, \ldots, C_m)$
$\ldots$
$inv_r : I_r(x, S_1, \ldots S_n, C_1, \ldots, C_m)$
THEOREMS
$th_1 : SAFE_1(x, S_1, \ldots S_n, C_1, \ldots, C_m)$
$\ldots$
$th_s : SAFE_s(x, S_1, \ldots S_n, C_1, \ldots, C_m)$
EVENTS
EVENT initialisation
BEGIN
$x :\mid (P(x'))$
END
$\ldots$
EVENT e
ANY t
WHERE
$G(x, t)$
THEN
$x :\mid (P(x, x', t))$
END
$\ldots$
END

eration of the triptych of Bjoerner [24–27] and variations of this methodology.

2.4 Machines

A *machine* (see Fig. 4) is either basic or a refinement of a more abstract machine. A machine models a state via a list of variables x that are assumed to be modifiable by events listed in the machine. In this work we distinguish between *invariant* assertions, maintained by all events and *safety* properties which must be proved as theorems on the invariants. The invariant is a conjunction of logical statements called I_j. Proof obligations are given in the last section, and they are generated and checkable by the RODIN framework. The validation of the machine M leads to the validation of the safety and invariance properties.

We can obtain a variation of the triptych ($\Gamma(\mathcal{D}, M)$ is an associated environment for proof) as follows:

- For any j in $\{1..r\}$,
  $\Gamma(\mathcal{D}, M) \vdash BA(Init)(x') \Rightarrow I_j(x', S_1, \ldots S_n, C_1, \ldots, C_m)$
- For any j in $\{1..r\}$, for any event e of M,

$$\Gamma(\mathcal{D}, M) \vdash \left(\begin{array}{l} \left(\bigwedge_{j\in\{1..r\}} I_j(x, S_1, \ldots S_n, C_1, \ldots, C_m) \right) \wedge BA(e)(x, x') \\ \Rightarrow I_j(x', S_1, \ldots S_n, C_1, \ldots, C_m) \end{array} \right)$$

- For any k in $\{1..s\}$,

$$\Gamma(\mathcal{D}, M) \vdash \left(\begin{array}{l} \left(\bigwedge_{j\in\{1..r\}} I_j(x, S_1, \ldots S_n, C_1, \ldots, C_m) \right) \\ \Rightarrow SAFE_k(x, S_1, \ldots S_n, C_1, \ldots, C_m) \end{array} \right)$$

Using temporal operators for expressing the safety and invariant properties, we summarise the requirements expressed by the machine M as follows:

$$\mathcal{D}, M \longrightarrow \Box \left(\begin{array}{l} \left(\bigwedge_{j\in\{1..r\}} I_j(x, S_1, \ldots S_n, C_1, \ldots, C_m) \right) \\ \left(\bigwedge_{k\in\{1..s\}} SAFE_k(x, S_1, \ldots S_n, C_1, \ldots, C_m) \right) \end{array} \right)$$

2.5 System specification

We have shown that requirements $\mathcal{R}$ are first expressed using the *always* temporal operator. To specify total correctness properties, we should extend the scope of the requirements language by adding *eventuality* properties. Eventuality properties will be defined in the next section and will be specific to our methodology.

The definition of the specification of traces is defined as follows:

Definition 1 Let M be an EVENT B machine and D a context seen by M. Let x be the list of variables of M, let E be the set of events of M, and let $Init(x)$ be the initialisation event in M. The temporal framework of M over D is defined by the TLA specification denoted:
$\mathcal{S}pec(M) \widehat{=} BA(Init)(x) \wedge \Box[Next]_x \wedge$FAIR, where $Next \equiv \exists e \in E.BA(e)(x, x')$ and FAIR defines the fairness assumptions.

Following Lamport [51,52], the specification $\mathcal{S}pec(M)$ is valid for the set of infinite traces simulating M with respect to the events of M. $\mathcal{S}pec(M)$ is thus defined by the initial conditions, the next relation and fairness constraints. In practice we have to discover the weakest fairness assumptions, denoted FAIR(M), that allow us to derive the required liveness properties. These fairness assumptions emerge from the proof rules applied and are expressed in terms of the temporal operators of TLA, namely WF and SF . FAIR(M) is thus a combination of fairness operators over events of M. Liveness properties for M are, de facto, defined in TLA as follows: M satisfies $P \rightsquigarrow Q$, when $\Gamma(M) \vdash \mathcal{S}pec(M) \Longrightarrow (P \rightsquigarrow Q)$. When deriving proof of $\mathcal{S}pec(M) \Longrightarrow (P \rightsquigarrow Q)$, we apply the right introduction rule of the implication and then we eliminate the conjunctive connective in the left part of the $\vdash$ symbol. Thus $\Gamma(M)$ will be increased by fairness assumptions and we can use an alternative form for expressing the initial sequent: $\Gamma(M)$ is the proof context of M. An alternative approach to liveness properties is to use the wp-based approach for defining the liveness properties under weak fairness [32,55,56]. Next, we have to extend the scope of the Event-B refinement by providing conditions to maintain liveness properties.

2.6 Liveness-preserving refinement

Sections 2.3 and 2.4 defined the POs for the machines and contexts in the proof landscape of the development shown in Fig. 5. This figure gives the relationships amongst the different structures of contexts, machines, temporal properties. Internal consistency of contexts and machines is defined by these POs. These POs and the relationships SEES and EXTENDS are defined in the Event-B modelling language and implemented by the RODIN toolkit.

We say that machine $M(y)$ refines machine $AM(x)$ according to the Event-B refinement relationship REFINES in Fig. 5. This is expressed in the *gluing invariant* $J(x, y)$ which may include new constraints on y. Safety properties on y are also expressible as per Sect. 2.4. For each event $ae(x)$ in AM, there exists an event $ce(y)$ in M which refines and preserves its externally visible behaviour. The associated POs are:

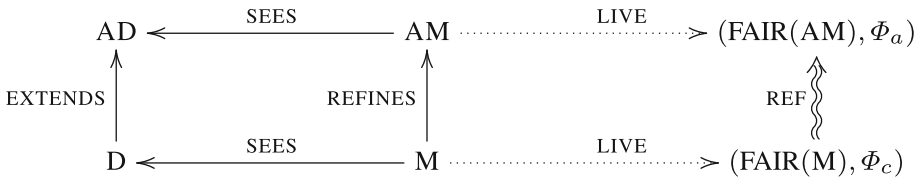


Fig. 5 Summary of the refinement methodology

- $BA(Init_c)(y') \Rightarrow \exists x'.BA(Init_a)(x) \wedge J(x', y')$
- $\text{INV}(x) \wedge \text{J}(x, y) \wedge BA(ce)(y, y') \Rightarrow \exists x'.(BA(ae)(x, x') \wedge \text{J}(x', y'))$

The methodology is sketched by the diagram of Fig. 5. The given models are AD and AM with fairness assumptions and liveness requirements FAIR(AM) and Φ_a. The refinement step is achieved by defining the list of liveness properties Φ_c so that Φ_a can be derived from Φ_c using inference rules for *leadsto* properties. Then one defines the new Event-B model M. This last point may be demanding, since one should also define carefully how fairness is preserved by the new events.

The relationship LIVE expresses the following properties:

- For any liveness property $P \rightsquigarrow Q$ of Φ_a: $\Gamma(AD, AM)$, $\mathcal{S}pec(AM) \vdash P \rightsquigarrow Q$.
- For any liveness property $P \rightsquigarrow Q$ of Φ_c: $\Gamma(D, M)$, $\mathcal{S}pec(M) \vdash P \rightsquigarrow Q$

Finally, the relationship REF is stating that any liveness property $P \rightsquigarrow Q$ of Φ_a is derivable from Φ_c and M. We can rewrite this condition as follows:

$$\forall P, Q.(P \rightsquigarrow Q \in \Phi_a) \implies (\Gamma(D, M), \mathcal{S}pec(M), \Phi_c \vdash (P \rightsquigarrow Q)) \quad (1)$$

The process of refinement REF is driven by the derivation of *abstract* liveness properties from concrete models and it is based on the use of inference rules for deriving liveness properties expressed as *leadsto* expressions. The meaning of REF is supported by the deduction relation of the liveness properties using the proof system of TLA. At this point, we have to indicate that the TLA [51] refinement is defined by the logical relationship $\mathcal{S}pec(M) \Rightarrow \mathcal{S}pec(AM)$: each concrete trace is an abstract trace up to stuttering steps. In fact, when considering the refinement process, the list of POs states a necessary condition for inferring the refinement relationship. It means that the objective is to preserve a list of safety and liveness properties and we face a major problem, when trying to use an inclusion of traces. This strong constraint, namely the trace inclusion up to stuttering, can be weakened and our p*roof-directed* refinement follows this simple idea.

However, our approach to the refinement of liveness properties is intended to mimic Event-B, where refinement of AM by M is checked by discharging a list of POs. The relationship REFINES in Fig. 5 denotes our reuse, *for free*, of POs generated by Event-B/RODIN modelling and refinement; this guarantees invariant and safety property preservation through refinement. We reduce liveness refinement checking to a list of liveness POs. In the context of liveness properties, Abadi and Lamport [1] introduce the notion of *refinement mapping* to relate two specifications. Their main result *proposition 1* concerns a necessary condition for inferring the *implementation* of a specification S_2 by a specification S_1. A specification S is a four-tuple (Σ, F, N, L) where (Σ, F, N) is a state machine over state space Σ, initial states $F \subseteq \Sigma$ and transition relation N. The notion of property induced by S is defined to be the set of S-behaviours closed under stuttering, denoted Σ and L is a Σ-property and expresses *liveness* properties which are required and which are restricting the set of behaviours. In our case, the state machine is an Event-B machine and L is expressed using *fairness* assumptions FAIR(M) which restrict traces. The *implements* relationship is defined as follows: a specification S_1 implements a specification S_2 if, and only if, the externally visible property induced by S_1 is a subset of the externally visible property induced by S_2. In our case, the *implements* relationship is called *refinement*. Consequently, when we refine, we define a refinement mapping $f : \Sigma_1 \rightarrow \Sigma_2$ based on the following verification conditions:

1. The refinement mapping preserves the externally visible state component: a consequence of the refinement of each event e of S_2 by an event of S_1.
2. Initial states of S_1 are mapped under f to initial states of S_2
3. For each event e_1 of S_1, e_1 refines some e_2 of S_2 or is a stuttering step in S_2, thus under f, $N_1 \Rightarrow N_2$
4. $f(L_1) \subseteq L_2$

Under the existence of a refinement mapping, the specification S_1 implements S_2, which means that the traces of S_1 are, up to stuttering, traces of S_2.

Next we express more precisely what we mean by the refinement REF of liveness-extended Event-B models as per

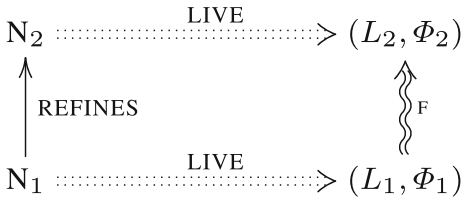


Fig. 6 Refinement methodology of Abadi and Lamport

Fig. 5. This extended meaning of refinement states that the traces of the refinement model are included in the set of traces of the abstract model according to the definition of Abadi and Lamport [1] (Fig. 6).

Considering the liveness part of Fig. 5 we recall that the meaning of the LIVE arrows in their respective proof environments is $\mathcal{S}pec(AM) \vdash \Phi_a$ and $\mathcal{S}pec(M) \vdash \Phi_c$. Thus Abadi and Lamport suggest the following approach (called A1):

1. Development of a concrete Event-B machine driven by concrete liveness properties from abstract AM and Φ_a: M and Φ_c
2. Verifying that AM satisfies abstract liveness properties: $\mathcal{S}pec(AM) \vdash \Phi_a$
3. Stating the concrete fairness assumptions FAIR(M)
4. Verifying that M satisfies concrete liveness properties: $\mathcal{S}pec(M) \vdash \Phi_c$
5. Verifying that FAIR$(M) \Rightarrow$ FAIR(AM)
6. Verifying that $\mathcal{S}pec(M) \Rightarrow \mathcal{S}pec(AM)$: we must find a refinement mapping. Thus we have for free that $\mathcal{S}pec(M) \Rightarrow \Phi_a$, i.e. refinement of liveness

From the theorem of Abadi and Lamport, we derive trace refinement $Spec(M) \Rightarrow Spec(AM)$. In practice, however, it may be difficult to prove these strong refinement conditions. We propose a second possible proof approach to relate the two models but we do not prove that the two sets of conditions are equivalent. In the second approach (called A2), we establish the following steps:

1. Development of a concrete Event-B machine driven by concrete liveness properties from abstract AM and Φ_a: M and Φ_c
2. Stating the concrete fairness assumptions FAIR(M)
3. Verifying that M satisfies concrete liveness properties: $\mathcal{S}pec(M) \vdash \Phi_c$
4. Verifying that $\mathcal{S}pec(M), \Phi_c \vdash \Phi_a$[3]. Thus we must prove refinement of liveness. But we get for free that Φ_a, i.e. we do not need to prove $\mathcal{S}pec(AM) \vdash \Phi_a$

When these conditions are checked, it means that we do not have to prove that the abstract liveness properties are derived from the abstract model, since they have already been derived in step 4. We are driven by the inference rules and we design the concrete Event-B machine to have the concrete liveness properties required to derive the abstract liveness properties ($\Phi_c \vdash \Phi_a$).

However, when it is possible and when we are able to discharge the refinement of fairness assumptions, we can simply apply the approach 1, and the approach 2 follows implicitly. We summarise our refinement verification as follows assuming that the machine M refines AM with respect to Event-B:

- Case 1 (A1): we prove that $\mathcal{S}pec(AM) \vdash \Phi_a$, FAIR$(M) \Rightarrow$ FAIR(AM), that $\mathcal{S}pec(M) \Rightarrow \mathcal{S}pec(AM)$ and that $\mathcal{S}pec(M) \vdash \Phi_c$. It follows that $\mathcal{S}pec(M) \vdash \Phi_a$.
- Case 2 (A2): we prove that $\mathcal{S}pec(M) \vdash \Phi_c$ and $\mathcal{S}pec(M), \Phi_c \vdash \Phi_a$. It follows that $\mathcal{S}pec(M)$ satisfies Φ_a.

3 Red and green lights

We present the lights [17], a simple population protocol modelled and refined in Event-B, in order to demonstrate the temporal style of reasoning about convergence. We emphasise the explicit temporal proof of convergence through the liveness properties that define and support such convergence. Each lower-level proof step—the hypotheses used in the temporal proof rules—is easily coded and proved in RODIN; we indicate any exceptions to this. Figure 7 gives a schematic structure for this partial refinement-based development satisfying temporal properties under defined fairness assumptions.

Network nodes l are coloured red or green (coded $l \in V \rightarrow$ COLOURS where COLOURS $= \{green, red\}$). Node interaction is defined for any two adjacent red nodes—i.e. connected by the interaction graph—when one node turns green. The protocol terminates when only one red remains.

Initially, the lights are modelled over a complete interaction graph. The first model PPM_0 specifies an abstract, nondeterministic "one-shot" convergence, simply proved with WF1. The first refinement PPM_1 describes pairwise interaction between red nodes—event iact—as an inductive process. The inductive *leadsto* property is proved by WF1. The refinement of the liveness property, by its decomposition into the underlying induction, is trivial by LATTICE. The second refinement PPM_2 introduces a dynamic interaction graph through angel and demon events. This breaks the always enabled interaction. Thus the induction must be reproved using SF1 and a lemma, assuming weak fairness of the angel, that re-enables interaction.

[3] We will write Φ_c REF Φ_a to abbreviate this.

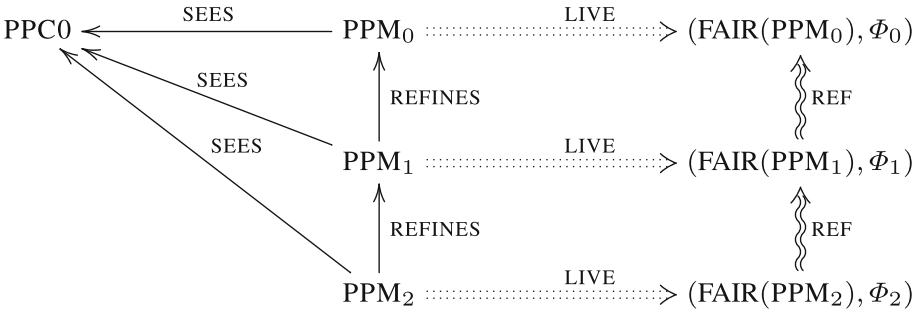


Fig. 7 Development scheme

3.1 Model PPM_0

In this first, most abstract Event-B model PPM_0, the graph is complete—every node is connected to every other. Nodes are initialised to an arbitrary initial configuration with at least one red node. As per Event-B convention, the first model specifies the required convergence property as a one-shot transition; conv_0 nondeterministically selects one red node and sets all others to green. Apart from initialisation the model has two events conv_0 and iact_0, each modifying one variable $l \in V \to \{red, green\}$. iact_0 is an Event-B artefact to ease refinement and will not affect the liveness proof.[4]

EVENT conv_0
ANY i
WHERE $i \in V \wedge l(i) = red$
THEN $l :| \; l' \in V \to \{red, green\} \wedge l'^{-1}[\{red\}] = \{i\}$
END

EVENT iact_0 *anticipated*
ANY l
THEN $l :| \; l' \in V \to \{red, green\} \wedge l' \rhd \{red\} \subseteq l \rhd \{red\}$
END

We illustrate the proof scheme with this initial trivial liveness proof. Here and in the refinements, bearing in mind the limitations of the Event-B provers over the arithmetic of the naturals, we express cardinalities in terms of fixed-size injections.

- $\mathrm{FAIR}(PPM_0)$ defines the fairness assumption over the event conv_0: $\mathrm{WF}_l(\mathsf{conv}_0)$
- $\Phi_0 \stackrel{def}{=} (l \in V \to \mathrm{COLOURS} \wedge l^{-1}[\{red\}] \neq \varnothing) \rightsquigarrow (\exists f.f \in 1 \mathinner{..} 1 \rightarrowtail\!\!\!\!\!\!\rightarrow l^{-1}[\{red\}])$
- We define the invariant for this model: $Inv_{PPM_0} \mathrel{\widehat{=}} l \in V \to \mathrm{COLOURS} \wedge l^{-1}[\{red\}] \neq \varnothing$
- Φ_0 is proved by Lamport's WF1 rule; we list the WF1 hypotheses derived from PPM_0:
  - For each event e (here, just one event $\mathsf{e} = \mathsf{conv}_0$) of PPM_0,

$$\left(\begin{pmatrix} Inv_{PPM_0} \\ \wedge \neg(\exists f.f \in 1 \mathinner{..} 1 \rightarrowtail\!\!\!\!\!\!\rightarrow l^{-1}[\{red\}]) \\ \Rightarrow \left(Inv'_{PPM_0} \vee \exists f.f \in 1 \mathinner{..} 1 \rightarrowtail\!\!\!\!\!\!\rightarrow l'^{-1}[\{red\}]\right) \end{pmatrix} \wedge \begin{pmatrix} \mathrm{BA}(\mathsf{e})(l, l') \\ \vee \; l' = l \end{pmatrix}\right)$$

  - $$\left(\begin{array}{l} \left(Inv_{PPM_0} \wedge \neg(\exists f.f \in 1 \mathinner{..} 1 \rightarrowtail\!\!\!\!\!\!\rightarrow l^{-1}[\{red\}])\right) \\ \quad \wedge \mathrm{BA}(\mathsf{conv}_0)(l, l') \\ \Rightarrow \exists f.f \in 1 \mathinner{..} 1 \rightarrowtail\!\!\!\!\!\!\rightarrow l'^{-1}[\{red\}] \end{array}\right)$$

  - $$\left(Inv_{PPM_0} \wedge \neg(\exists f.f \in 1 \mathinner{..} 1 \rightarrowtail\!\!\!\!\!\!\rightarrow l^{-1}[\{red\}])\right) \Rightarrow \textsc{Enabled}\ \langle \mathsf{conv}_0 \rangle_l$$

3.2 Model PPM_1

In refinement PPM_1 new event iact_1 pairwise switches one red node of an adjacent red pair to green. Event conv_1—guarded by convergence to a single red node—refines conv_0. It skips, simply observing that convergence has taken place. Event-B refinement allows such strengthening of guards, as long as the overall system guard is maintained; there are associated POs.

EVENT conv_1
REFINES conv_0
ANY i
WHERE
$i \in V \wedge l^{-1}[\{red\}] = \{i\}$
END

EVENT iact_1 *convergent*
ANY $i \; j$
WHERE
$i \in V \wedge j \in V \wedge i \neq j \wedge l(i) = red \wedge l(j) = red$
THEN $l(i) := green$
END

VARIANT $l \rhd \{red\}$

Convergence is proved straightforwardly using the WF1 fairness and LATTICE induction proof rules. In PPM_1, it is obvious that each interaction iact_1 reduces the problem and that $l \rhd \{red\}$ is a suitable set-valued inductive variant

[4] iact_0 "anticipates" the interaction in the subsequent refinement PPM_1 by allowing maximal nondeterministic change in l under the constraint of the invariant and the initial configuration cl. This provides a vehicle against which the behaviour of "anticipated" interaction event iact_1 on l can simulate defined behaviour in PPM_0. It removes the need for a duplicate variable for l in a data refinement.

expression[5]. Using TLA we can be more explicit about the inductive process of convergence than we can in Event-B:

- FAIR(PPM$_1$) defines the fairness assumptions over events conv$_1$ and iact$_1$:
  $\mathrm{WF}_l(\mathsf{iact}_1) \wedge \mathrm{WF}_l(\mathsf{conv}_1)$
- We define the invariant for this machine PPM$_1$:
  $Inv_{PPM_1} \mathrel{\widehat{=}} Inv_{PPM_0} \equiv l \in V \to \mathrm{COLOURS} \wedge l^{-1}[\{red\}] \neq \varnothing$
- We define the inductive assumptions:
  $\forall n.n \in 0..card(V) - 1 \Rightarrow R(n) \mathrel{\widehat{=}} \exists f.f \in 1 \,..\, (n+1) \rightarrowtail l^{-1}[\{red\}]$
- The new liveness property to check is defined:
  $\Phi_1 \stackrel{def}{=} \left[\forall n \in 0..card(V) - 1 \cdot Inv_{PPM_1} \wedge R(n+1) \rightsquigarrow R(n)\right]$
- Φ_1 REF Φ_0 is a simple induction proof by the LATTICE rule applied to the assumptions $R(n)$
- We now prove Φ_1 using the WF1 rule, assuming WF_l(iact$_1$), and deriving the hypotheses from the machine PPM$_1$ as before:
  - For each event e of PPM$_1$,
    $Inv_{PPM_1} \wedge R(n+1) \wedge (\mathrm{BA}(\mathsf{e})(l, l') \vee l' = l)$
    $\Rightarrow Inv'_{PPM_1} \wedge (R'(n+1) \vee R'(n))$
  - $Inv_{PPM_1} \wedge R(n+1) \wedge \mathrm{BA}(\mathsf{iact}_1)(l, l')$
    $\Rightarrow Inv'_{PPM_1} \wedge R'(n)$
  - $Inv_{PPM_1} \wedge R(n+1) \Rightarrow \textsc{Enabled}\ \langle \mathsf{iact}_1 \rangle_l$

We see that liveness properties provide the guidelines for the refinement REF. The weak fairness assumption on iact$_1$ is derived from the liveness proof WF1 of the induction step Φ_1. Note that we have proved convergence of iact$_1$ to $R(0)$, i.e. a single red node. Following Event-B convention, this convergence is observed by a termination event: conv$_1$. For simplicity here conv$_1$ skips; usually it would set a termination flag, say $end := TRUE$. A further trivial WF1 liveness proof—assuming WF_l(conv$_1$)—is required to prove that $Inv_{PPM_1} \wedge R(0) \rightsquigarrow end = TRUE$.

3.3 Model PPM$_2$

In refinement PPM$_2$ we add a new variable $c \in V \leftrightarrow V$ to model the dynamically connected graph, initialised arbitrarily. The iact$_2$ guard is refined, allowing only connected nodes to interact. The environment is modelled by two new events: a daemon event models, e.g. loss of radio connectivity or node failure by arbitrary reassignment of network connectivity and angel event models the contribution of the environment to liveness, e.g. through improvement in radio signal quality, or intervention of a support team, by adding a new link between two red nodes.

```
EVENT daemon
BEGIN
  c :∈ {r|r : ℙ(V × V)
        ∧ r ∩ id = ∅}
END
```

```
EVENT angel
ANY  i  j
WHERE
  i ∈ V ∧ j ∈ V ∧ i ≠ j
  l(i) = red ∧ l(j) = red
  i ↦ j ∉ c
THEN
  c := c ∪ {i ↦ j})
END
```

```
EVENT iact2
ANY  i  j
WHERE
  i ∈ V ∧ j ∈ V ∧ i ≠ j
  l(i) = red ∧ l(j) = red
  i ↦ j ∈ c
THEN  l(i) := green
END

VARIANT  l ▷ {red}
```

The simple angel-daemon model of the environment's dynamic disruption of the network is essentially nondeterministic; an implicit variant such as in PPM$_1$ is not available. The variant-based convergence proof required by Event-B effectively forces us to schedule the environment here explicitly, perhaps designing in some counter or time bound on which to base a variant. This is a too concrete view of scheduling, and TLA allows more flexible and abstract reasoning about scheduling and convergence. Note that iact$_2$ is no longer always enabled since two reds may not be connected at a given time. It may be infinitely often disabled and thus needs a strong fairness assumption. Subject to the following definitions, we will apply SF1 and LATTICE to prove liveness Φ_{20}—i.e. Φ_1 over refined traces—for this refinement. In this proof we will reveal a further liveness property Φ_{21} required as a lemma.

- FAIR(PPM$_2$) defines the fairness assumptions over the events:
  $\mathrm{SF}_{l,c}(\mathsf{iact}_2) \wedge \mathrm{WF}_{l,c}(\mathsf{conv}_2) \wedge \mathrm{WF}_{l,c}(\mathsf{angel}) \wedge \mathrm{WF}_{l,c}(\mathsf{daemon})$.
- Invariant Inv_{PPM_2}
  $\mathrel{\widehat{=}} \begin{pmatrix} l \in V \to \mathrm{COLOURS} \wedge l^{-1}[\{red\}] \neq \varnothing \\ \wedge\ c \in V \leftrightarrow V \wedge c \cap id = \varnothing \end{pmatrix}$
- The inductive liveness property must now be reproved as Φ_{20}, for infinitely often disabled iact$_2$. The need for lemma Φ_{21} emerges from the proof of Φ_{20}:
  - $\Phi_{20} \stackrel{def}{=} \left[\forall n \in 0..card(V) - 1 \cdot Inv_{PPM_2} \wedge R(n+1) \rightsquigarrow R(n)\right]$
  - $\Phi_{21} \stackrel{def}{=} \forall n \in 0..card(V) - 1 \cdot Inv_{PPM_2} \wedge R(n+1) \rightsquigarrow Enabled\langle \mathsf{iact}_2 \rangle_l$
- Φ_{20} is proved by rules SF1, REWRITING and lemma Φ_{21}. Φ_{21} is proved by WF1. $\{\Phi_{20}, \Phi_{21}\}$REF Φ_1 follows.

[5] Keyword *convergent* for iact$_1$ generates an inductive PO requiring this variant to be reduced by the event.

PROVE: $\Phi_{20} \mathrel{\widehat{=}} \forall n \in 0..card(V)-1 \cdot Inv_{PPM_2} \wedge R(n+1) \rightsquigarrow R(n)$
APPLY: Rule SF1.

- ► PROVE: For each event e of PPM_2, $Inv_{PPM_2} \wedge R(n+1) \wedge (\mathrm{BA}(\mathsf{e})(l,c,l',c') \vee (l'=l \wedge c'=c)) \Rightarrow Inv'_{PPM_2} \wedge (R'(n+1) \vee R'(n))$
- ► PROVE: $Inv_{PPM_2} \wedge R(n+1) \wedge \mathrm{BA}(\mathsf{iact}_2)(l,l') \Rightarrow Inv'_{PPM_2} \wedge R'(n)$
- ► PROVE: $\Box(Inv_{PPM_2} \wedge R(n+1)) \wedge \Box[N]_{l,c} \wedge \mathrm{WF}_{l,c}(\mathsf{angel}) \Rightarrow \Diamond \mathrm{ENABLED}\ \langle \mathsf{iact}_2 \rangle_l$

PROOF: lemma Φ_{21} to follow, REWRITING □

Note that the state variable subscripts for the action formulae constituting the above hypotheses indicate which state variables are in frame. For example, iact_2 only acts on l, whereas $\Box[N]$ acts on l, c. The first two hypotheses of SF1 are proved in first-order logic, similarly to the WF1 proof of PPM_1.

The third hypothesis is a formula in temporal logic not directly expressible in Event-B, establishing the eventual enablement of iact_2 under weak fairness of the angel. In PPM_2 the environment's dynamic effect on the network is modelled by the daemon and—helped by the support team—the angel. We assume the angel is always enabled, and thus a weak fairness assumption suffices to ensure it acts infinitely often. We require the following lemma:

- PROVE: $\Phi_{21} \mathrel{\widehat{=}} \forall n \in 0..card(V)-1 \cdot Inv_{PPM_2} \wedge R(n+1) \rightsquigarrow Enabled\langle \mathsf{iact}_2 \rangle_l$
  APPLY: Rule WF1
  - ► PROVE: For each event e of PPM_2, $Inv_{PPM_2} \wedge R(n+1) \wedge (\mathrm{BA}(\mathsf{e})(l,c,l',c') \vee (l'=l \wedge c'=c)) \Rightarrow Inv'_{PPM_2} \vee \mathrm{ENABLED}'\langle \mathsf{iact}_2 \rangle_l$
  - ► PROVE: $Inv_{PPM_2} \wedge R(n+1) \wedge \mathrm{BA}(\mathsf{angel})(c,c') \Rightarrow Inv'_{PPM_2} \wedge \mathrm{ENABLED}'\langle \mathsf{iact}_2 \rangle_l$
  - ► PROVE: $Inv_{PPM_2} \wedge R(n+1) \Rightarrow \mathrm{ENABLED}\langle \mathsf{angel} \rangle_c$

3.4 Lights: postscript

It is useful finally to add one more step in the direction of realism in this example. Whereas the daemon of environmental conditions or damage may reasonably be assumed to be always enabled, the angel may not: bad weather conditions for node–node radio transmission take time to clear, as does a maintenance team to replace batteries on nodes. It is thus more realistic to place a strong fairness requirement on a sometimes-enabled angel. We then find that the analogous proof to Φ_{21} of the above becomes a strong fairness proof—thus generating another, secondary PO Φ_{22} on the enablement of the angel:

$Inv \wedge R(n+1) \rightsquigarrow \mathrm{ENABLED}\ \langle \mathsf{angel} \rangle_c$

This process suggests a recursive first-order proof method—provided the recursion terminates with some initial, weakly fair triggering action.

In this example, we show some elements of our approach to proof. With the increased granularity of each refinement, in modifying and adding both variable types and events, we are decomposing and elaborating the liveness properties. Thus we show how the coarse-grained abstract liveness properties arise out of the finer-grained concrete ones. We also elaborate our fairness assumptions in the finer-grained setting of each refinement.

Note, in this simple example, that all constituent proof tasks contain no temporal operators and are thus all statements of FOL. The third hypothesis of the SF1 proof reduces to a WF1 proof. The proofs are all therefore expressible and provable in Event-B/RODIN; this we have done.

4 The dancers

A group of dancers [17] are each marked as either follower(f) or leader(l). The aim of this protocol is to establish whether there are more leaders, more followers or equal numbers of each. The protocol should converge to a configuration where if there are (i) initially more leaders than followers, then #(leaders–followers) leaders and no followers remain, (ii) initially more followers than leaders, then #(followers–leaders) followers and no leaders remain, (iii) initially equal numbers of followers and leaders, then none of either remain. The target configuration is reached by applying the following transition rules:

$$\mathsf{f} \leftrightarrow \mathsf{l} \Rightarrow 0 \leftrightarrow 0 \quad \mathsf{l} \leftrightarrow 0 \Rightarrow \mathsf{l} \leftrightarrow 1 \quad 0 \leftrightarrow 1 \Rightarrow 0 \leftrightarrow 0$$
$$\mathsf{f} \leftrightarrow 1 \Rightarrow \mathsf{f} \leftrightarrow 0$$

The protocol works by eliminating f-l pairs by the first rule. It is obvious that only this rule changes the sets of followers and leaders; the others manipulate only the ones and zeroes. Thus, the sets of followers and leaders are monotonically reduced pairwise until one is empty. We use the names F, L, O, U for the sets of followers, leaders, ones and zeroes, respectively. For convenience of proof we use the name X for a set that may be either all followers or all leaders, depending on context. We show that this protocol *eventually* leads to one of two *stable* (unchanging) configurations: either F, O or F, U.

As for the lights, the first model Dance_0 converges in one-shot, dealing separately with the two cases. The live-

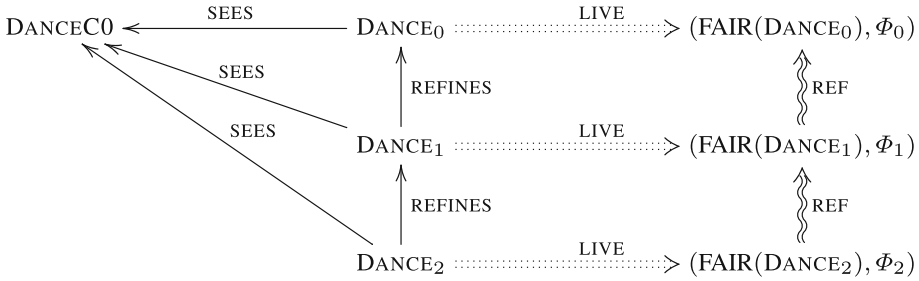


Fig. 8 Development scheme

ness proof is essential by WF1 as before. The first refined model Dance$_1$ introduces two intermediate configurations X, O, U for more followers and leaders, respectively. Thus convergence is in two steps for each case, proved by WF1 and transitivity. The second refinement Dance$_2$ introduces the pairwise interaction rules of the protocol. Inductive *leads-to* properties at this level are introduced to decompose convergence in two phases through X, O, U, finally to X, O or X, U. The inductive cases X, O, U though X, O are proved by WF1. However, the case through X, U is more complex and requires a new proof rule GF1, which we introduce. We will see how the proof of liveness properties at each level is effectively a stepwise construction of the proof $\mathcal{S}pec(M), \Phi_c \vdash \Phi_a$ as per our approach A2 of Sect. 2.6.

Figure 8 shows the development scheme for the Dancers. As before each lower-level proof step, the hypotheses used in the temporal proof rules are easily coded and proved in RODIN; we indicate any exceptions to this.

The set D of all dancers is initially partitioned into F_0 the initial set of followers, L_0 the initial set of leaders, O_0 the initial set of *zero* dancers and U_0 the initial set of *one* dancers. In the original problem [17], the sets U_0 and O_0 are empty but here we generalise the problem. Moreover, the original problem only stabilises to no leaders or no followers (or both), where the ones and zeroes can remain dynamic through enabled transition rules. We converge to a terminating state.

In temporal language the first property to verify is:

$$\Phi_0 : partition(D, L_0, F_0, U_0, O_0) \leadsto \exists \begin{pmatrix} X, \\ U, \\ O \end{pmatrix} . \begin{pmatrix} partition(D, X, U, O) \\ \wedge \begin{pmatrix} X \subseteq F_0 \wedge U = \varnothing \\ \vee \\ X \subseteq L_0 \wedge O = \varnothing \end{pmatrix} \\ \wedge\ O_0 \cup U_0 \subseteq O \cup U \end{pmatrix}$$

4.1 Model Dance$_0$: stating the convergence to one of two stable configurations

Our first model Dance$_0$ starts by defining abstract events which in one shot nondeterministically assign to the appropriate case: either no leader or no follower. Event Followers applies when there are at least as many followers as leaders: there is an injection i from L_0 into F_0. Event Leaders applies when the number of leaders is strictly greater than the number of followers: there is a nonsurjective injection i from F_0 into L_0. Each event is *simulating in a one shot way* the liveness property according to two possible scenarios.

```
EVENT Followers0
  ANY i
  WHERE
    F = F0 ∧ L = L0 ∧ O = O0 ∧ U = U0
    i ∈ L0 ⤖ F0
  THEN
    U, O, L, F :|
      ( partition(D, F', L', O', U')
      ∧ L' = ∅ ∧ F' = F \ i[L]
      O' = O ∪ i[L] ∪ L ∪ U ∧ U' = ∅
      ∧ O0 ∪ U0 ⊆ O' ∪ U' ))
  END
```

```
EVENT Leaders0
  ANY i
  WHERE
    L = L0 ∧ F = F0 ∧ O = O0 ∧ U = U0
    i ∈ F0 ⤖ L0 ∧ i[F0] ≠ L0
  THEN
    U, O, L, F :|
      ( partition(D, F', L', O', U')
      ∧ F' = ∅ ∧ L' = L \ i[F]
      U' = U ∪ i[F] ∪ F ∪ O ∧ O' = ∅
      ∧ O0 ∪ U0 ⊆ O' ∪ U' ))
  END
```

This initial model asserts the existence of an injection from one set of dancers into the other. The algorithmic process will progressively construct the final injection. We should also notice that the members of D are not distinguishable.

Relationship LIVE on the first line of Fig. 8 (i.e. $\mathcal{S}pec(\text{Dance}_0) \vdash \Phi_0$) is derived using the LATTICE rule, case analysis and the WF rule for this model. The invariant Inv_{Dance_0} of this model asserts $partition(D, L, F, U, O)$ as well as allowing reduction of the sizes of F, L and increases in O, U.

- FAIR(DANCE$_0$) defines the fairness assumptions over events Followers$_0$ and Leaders$_0$: WF $_{F,L,O,U}$(Followers$_0$) $\wedge$ WF $_{F,L,O,U}$(Leaders$_0$)
- $Inv_{\text{Dance}_0} \mathrel{\widehat{=}} partition(D, F, L, O, U) \wedge F \subseteq F0 \wedge L \subseteq L0 \wedge O0 \cup U0 \subseteq O \cup U$

– The liveness property to prove is Φ_0
– Proof is by case analysis:

PROVE: $partition(D, L_0, F_0, U_0, O_0) \quad \leadsto \quad \exists \begin{pmatrix} X, \\ U, \\ O \end{pmatrix} . \begin{pmatrix} partition(D, X, U, O) \\ \wedge \begin{pmatrix} X \subseteq F_0 \wedge U = \varnothing \\ \vee \\ X \subseteq L_0 \wedge O = \varnothing \end{pmatrix} \\ \wedge\ O_0 \cup U_0 \subseteq O \cup U \end{pmatrix}$

APPLY: LATTICE rule and case analysis

► PROVE: $\left[\begin{array}{l} partition(D, L_0, F_0, U_0, O_0) \\ \Rightarrow \\ \left(\begin{array}{l} partition(D, L_0, F_0, U_0, O_0) \wedge \exists i.i \in L0 \rightarrowtail F0 \\ \vee \\ partition(D, L_0, F_0, U_0, O_0) \wedge \exists i.i \in F0 \rightarrowtail L0 \wedge i[F0] \neq L0 \end{array} \right) \end{array} \right]$

PROOF: Either there are at least as many followers as leaders, or there are more leaders. □

► PROVE: $partition(D, L_0, F_0, U_0, O_0) \wedge \exists i.i \in L0 \rightarrowtail F0 \leadsto \exists X. \begin{pmatrix} partition(D, X, O) \\ \wedge\ X \subseteq F_0 \wedge U = \varnothing \end{pmatrix}$

APPLY: Case - at least as many followers as leaders: apply WF1 to $\mathsf{Followers}_0$:

► PROVE: For each event e of DANCE$_0$,

$$\left(\begin{array}{l} \left(\begin{array}{l} \left(\begin{array}{l} partition(D, L_0, F_0, U_0, O_0) \\ \wedge \exists i.i \in L0 \rightarrowtail F0 \end{array} \right) \\ \wedge \neg \exists X. \begin{pmatrix} partition(D, X, O) \\ \wedge\ X \subseteq F_0 \wedge U = \varnothing \end{pmatrix} \end{array} \right) \\ \wedge \begin{pmatrix} \mathrm{BA}(\mathsf{e})((D, L, F, U, O), (D, L, F, U, O)') \\ \vee\ (D, L, F, U, O)' = (D, L, F, U, O) \end{pmatrix} \\ \Rightarrow \\ \left(\begin{array}{l} \left(\begin{array}{l} partition(D, L_0, F_0, U_0, O_0) \\ \wedge \exists i.i \in L0 \rightarrowtail F0 \end{array} \right) \\ \vee \\ \exists X'. \begin{pmatrix} partition(D', X', O') \\ \wedge\ X' \subseteq F_0 \wedge U' = \varnothing \end{pmatrix} \end{array} \right) \end{array} \right)$$

► PROVE:

$$\left(\begin{array}{l} \left(\begin{array}{l} \left(\begin{array}{l} partition(D, L_0, F_0, U_0, O_0) \\ \wedge \exists i.i \in L0 \rightarrowtail F0 \end{array} \right) \\ \wedge \neg \exists X. \begin{pmatrix} partition(D, X, O) \\ \wedge\ X \subseteq F_0 \wedge U = \varnothing \end{pmatrix} \end{array} \right) \\ \wedge \mathrm{BA}(\mathsf{Followers}_0)((D, L, F, U, O), (D, L, F, U, O)') \\ \Rightarrow \\ \exists X'. \begin{pmatrix} partition(D', X', O') \\ \wedge\ X' \subseteq F_0 \wedge U' = \varnothing \end{pmatrix} \end{array} \right)$$

► PROVE:

$$\left(\begin{array}{l} \left(\begin{array}{l} \left(\begin{array}{l} partition(D, L_0, F_0, U_0, O_0) \\ \wedge \exists i.i \in L0 \rightarrowtail F0 \end{array} \right) \\ \wedge \neg \exists X. \begin{pmatrix} partition(D, X, O) \\ \wedge\ X \subseteq F_0 \wedge U = \varnothing \end{pmatrix} \end{array} \right) \\ \Rightarrow \ \text{ENABLED}\ \langle \mathsf{Followers}_0 \rangle_{(D,L,F,U,O)} \end{array} \right)$$

► PROVE: $partition(D, L_0, F_0, U_0, O_0) \wedge \exists i.i \in F0 \rightarrowtail L0 \wedge i[F0] \neq L0 \leadsto \exists X. \begin{pmatrix} partition(D, X, U) \\ \wedge\ X \subseteq L_0 \wedge O = \varnothing \end{pmatrix}$

APPLY: Case - more leaders, apply WF1 rule to $\mathsf{Leaders}_0$ straightforwardly as above

► PROVE: Assemble final result predicate:

$$\left(\begin{array}{l} \exists X. \begin{pmatrix} partition(D, X, O) \\ \wedge\ X \subseteq F_0 \wedge U = \varnothing \end{pmatrix} \\ \vee \\ \exists X. \begin{pmatrix} partition(D, X, U) \\ \wedge\ X \subseteq L_0 \wedge O = \varnothing \end{pmatrix} \end{array} \right) \Rightarrow \exists \begin{pmatrix} X, \\ U, \\ O \end{pmatrix} . \begin{pmatrix} partition(D, X, U, O) \\ \wedge \begin{pmatrix} X \subseteq F_0 \wedge U = \varnothing \\ \vee \\ X \subseteq L_0 \wedge O = \varnothing \end{pmatrix} \\ \wedge\ O_0 \cup U_0 \subseteq O \cup U \end{pmatrix}$$

PROOF: Derived from predicate calculus. □

The LATTICE rule is applied in a specialised way with case analysis, and we refer to the CONFLUENCE rule as in proof lattices by Owicki and Lamport [60]. We summarise the proof technique in Fig. 9.

We have proved the liveness property Φ_0 using fairness assumptions on very macroscopic steps, namely $\mathsf{Followers}_0$ and Leader_0. In refinement Dance_2 we will see the dancers progressing according to a pairwise dancing rule; this means that there are intermediate configurations. We adopt some notation for simplifying the expression of diagrams and formulas. The following configurations are of interest. Note that each represents a partition of the dancers into up to four sets:

– $D = F \oplus L \oplus O \oplus U$: at least one each of Follower, Leader, ZERO and ONE
– $D = F \oplus O \oplus U$: at least one each of Follower, ZERO and ONE, and no Leader
– $D = L \oplus O \oplus U$: at least one each of Leader, ZERO and ONE, and no Follower
– $D = L \oplus U$: at least one each of Leader and ONE, and no Follower and no ZERO
– $D = F \oplus O$: at least one each of Follower and ZERO, and no Leader and no ONE
– $D = ... \oplus T \oplus ...$ where $T = 0$ or 1 means that there is exactly one ZERO or ONE
– $D = L \oplus 0^i \oplus 1^j$: at least one Leader, and i ZEROs and j ONEs

Using this notation we rewrite Figs. 9 as 10. We regard reaching these configurations as liveness properties. Thus, interpreting Figs. 10, 11 as predicate diagrams [31], each arrow states a $\leadsto$ property, describing progress between the configurations. Model Dance_0 in Fig. 10 is then *refined* to Dance_1 in Fig. 11, which introduces intermediate configurations and decomposes the two possible runs (either followers or leaders).

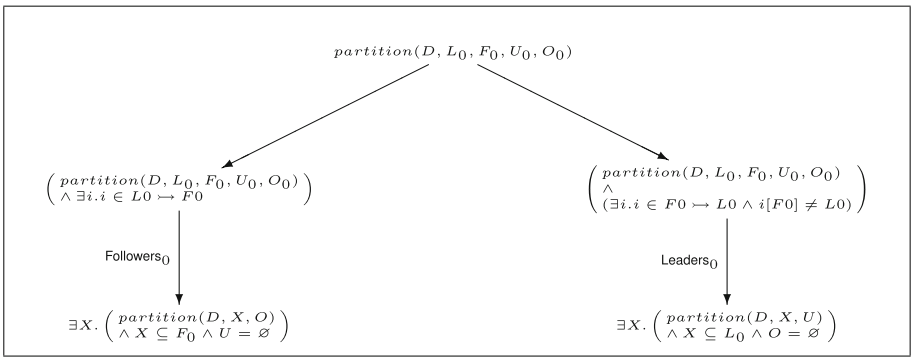


Fig. 9 Liveness proof diagram: Dance_0

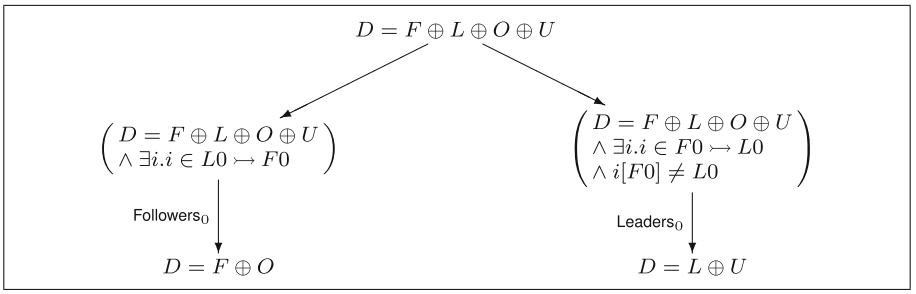


Fig. 10 Liveness proof diagram: Dance_0, schematic

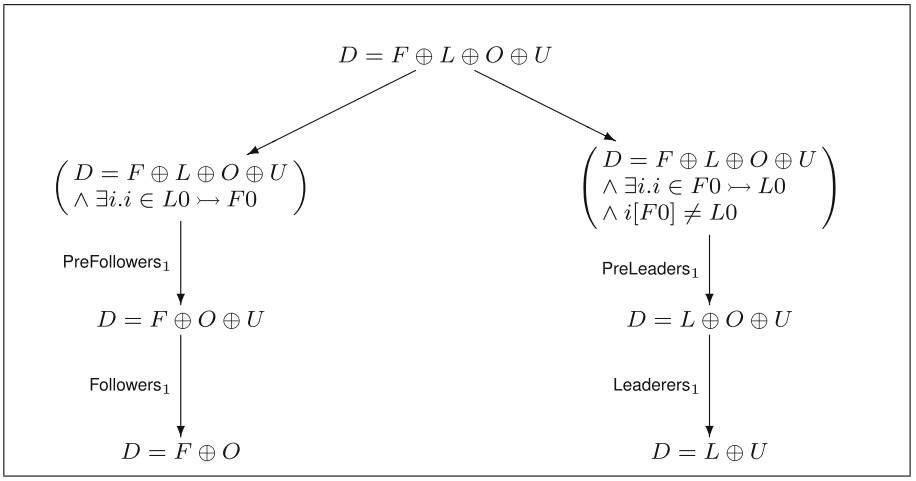


Fig. 11 Liveness proof diagram: Dance_1

4.2 Model Dance_1: decomposing the process into two phases

In the first refinement we rephrase and elaborate the liveness properties of Dance_0 in Fig. 10 to obtain a decomposition in Fig. 11, using the abstract notation introduced in the last subsection[6].

We describe the two target configurations $D = F \oplus O$ and $D = L \oplus U$ as *stable* in the sense of not changing further. Two new events are introduced in this step, namely $\mathsf{PreFollowers}_1$ and $\mathsf{PreLeaders}_1$ which each *prepare* the convergence by eliminating all (l,f) pairs defined by the injections. These events thus reach an interim configuration describing at least as many followers ($D = F \oplus O \oplus U$) or more leaders ($D = L \oplus O \oplus U$), respectively. These events abstract out all transitions of the three rules that change only ones and zeroes, until an interim configuration is reached. Note that our interim configurations are the final configurations of the protocol in [17]; we extend their result as per Fig. 11.

[6] For ease of reading, we gloss over the event artefact required to avoid redundant variables in data refinement, as per iact_0 in Sect. 3.1.

EVENT $\mathsf{PreFollowers}_1$
ANY i
WHERE
$grd1 : i \in L0 \rightarrowtail F0$
$grd2 : F = F0 \land L = L0$
$grd3 : O = O0 \land U = U0$
THEN
$act1 : F := F \setminus i[L]$
$act2 : U, O, L :|$
$(U' \subseteq D \land O' \subseteq D$
$\land U' \cap (F \setminus i[L]) = \varnothing \land O' \cap (F \setminus i[L]) = \varnothing$
$\land U' \cap O' = \varnothing \land O0 \cup U0 \subseteq O' \cup U' \land L' = \varnothing$
$\land D = L' \cup U' \cup O' \cup (F \setminus i[L])$
END

EVENT $\mathsf{PreLeaders}_1$
ANY i
WHERE
$grd1 : i \in F0 \rightarrowtail L0$
$grd2 : i[F0] \neq L0$
$grd3 : F = F0 \land L = L0$
$grd4 : O = O0 \land U = U0$
THEN
$act1 : L := L \setminus i[F]$
$act2 : U, O, F :|$
$(U' \subseteq D \land O' \subseteq D$
$\land U' \cap (L \setminus i[F]) = \varnothing \land O' \cap (L \setminus i[F]) = \varnothing$
$\land U' \cap O' = \varnothing \land O0 \cup U0 \subseteq O' \cup U' \land L' = \varnothing$
$\land D = F' \cup U' \cup O' \cup (L \setminus i[F])$
END

The two events $\mathsf{Followers}_1$ and $\mathsf{Leaders}_1$ refine the corresponding events of model Dance_0. We follow the same proof process as in that model to derive the following four liveness properties.

– FAIR(DANCE$_1$) defines the fairness assumptions over events $\mathsf{PreFollowers}_1$, $\mathsf{PreLeaders}_1$, $\mathsf{Followers}_1$, $\mathsf{Leaders}_1$:
$\mathrm{WF}_{F,L,O,U}(\mathsf{PreFollowers}_1) \land \mathrm{WF}_{F,L,O,U}(\mathsf{PreLeaders}_1) \land \mathrm{WF}_{F,L,O,U}(\mathsf{Followers}_1) \land \mathrm{WF}_{F,L,O,U}(\mathsf{Leaders}_1)$

$$Inv_{\mathsf{Dance}_1} \mathrel{\widehat{=}} Inv_{\mathsf{Dance}_0} \mathrel{\widehat{=}} partition(D, F, L, O, U) \land F \subseteq F0 \land L \subseteq L0 \land O0 \cup U0 \subseteq O \cup U$$

– $\Phi_1 \stackrel{def}{=} \left[\begin{array}{l} \left(\begin{array}{l} D = F \oplus L \oplus O \oplus U \\ \land\ \exists i.i \in L0 \rightarrowtail F0 \end{array} \right) \rightsquigarrow D = F \oplus O \oplus U \\ D = F \oplus O \oplus U \rightsquigarrow D = F \oplus O \\ \left(\begin{array}{l} D = F \oplus L \oplus O \oplus U \\ \land\ \exists i.i \in F0 \rightarrowtail L0 \\ \land\ i[F0] \neq L0 \end{array} \right) \rightsquigarrow D = L \oplus O \oplus U \\ D = L \oplus O \oplus U \rightsquigarrow D = L \oplus U \end{array} \right]$

– Φ_1 REF Φ_0

The derivation of relationship LIVE on the second line of Fig. 8 (i.e. $\mathcal{S}pec(\mathsf{Dance}_1) \vdash \Phi_1$) is simply done using the WF rule on each of the four events. We observe that Φ_1 REF Φ_0 follows by applying the proof steps of $\mathcal{S}pec(\mathsf{Dance}_0) \vdash \Phi_0$ of Sect. 4.1, as well as the TRANSITIVITY and CONFLUENCE rules. Thus the refinement of liveness proof elaborates the abstract liveness proof.

We have obtained a process which acts in two steps or phases, according to the case of either at least as many followers, or more leaders. In the next refinement we can decompose the events of the model Dance_1 into a set of events which model the algorithm of the original protocol itself.

4.3 Model Dance_2: identifying the algorithm

4.3.1 Prolegomena

We now refine the four events of Dance_1 into events which simulate the transition rules of the following population protocol:

$$\mathsf{f} \leftrightarrow \mathsf{l} \Rightarrow \mathsf{0} \leftrightarrow \mathsf{0} \quad \mathsf{l} \leftrightarrow \mathsf{0} \Rightarrow \mathsf{l} \leftrightarrow \mathsf{1} \quad \mathsf{0} \leftrightarrow \mathsf{1} \Rightarrow \mathsf{0} \leftrightarrow \mathsf{0} \quad \mathsf{f} \leftrightarrow \mathsf{1} \Rightarrow \mathsf{f} \leftrightarrow \mathsf{0}$$

Model Dance_2 introduces the algorithm for getting a configuration satisfying $D = F \oplus O \lor D = L \oplus U$. We introduce new variables $vf, vl, oldf, oldl, vu, vo, f, l$ with the following roles:

– vl, vf initially contain L_0, F_0, respectively; these variables accumulate the remaining leaders or followers. At convergence, L, F are respectively assigned to their final values
– $oldf, oldl$ are initialised empty; they accumulate the followers and leaders, respectively, as these are eliminated in pairs by the first transition rule
– vu, vo contain one-dancers and zero-dancers, respectively. At convergence, U, O are respectively assigned to their final values
– f, l each record the injection required for the refinement; these functions are constructed iteratively and only one is finally used, since either followers or leaders win. f maps $oldl$ to $oldf$ and l maps $oldf$ to $oldl$

Considering the transition rule $\mathsf{f} \leftrightarrow \mathsf{l} \Rightarrow \mathsf{0} \leftrightarrow \mathsf{0}$ implemented by event $\mathsf{Dancing}$: we see that when a dancer of vl or vf moves into vo, he/she will never return to vl or vf. The other rules do not change the state of participating L or F dancers. We can derive an inductive property based on $vl \cup vf$—with followers and leaders remaining—expressing the fact that the set $vl \cup vf$ is strictly decreasing by the rule $\mathsf{f} \leftrightarrow \mathsf{l} \Rightarrow \mathsf{0} \leftrightarrow \mathsf{0}$. The model Dance_2 is characterised by the invariant of Fig. 12.

The refinement is checked by the RODIN platform; we list the events of this model (The notation $BA(\mathsf{e})(h, h')$ denotes the before–after relation of the event e in the frame h.).

Fig. 12 Invariant for model Dance_2

$$
\begin{array}{l}
inv1 : l \in F0 \nrightarrow L0 \land f \in L0 \nrightarrow F0 \\
inv3 : f \neq \varnothing \Rightarrow f \in dom(f) \twoheadrightarrow ran(f) \\
inv4 : f \in L0 \rightarrowtail F0 \land l \in F0 \rightarrowtail L0 \\
inv6 : l \neq \varnothing \Rightarrow l \in dom(l) \twoheadrightarrow ran(l) \\
inv7 : vf \subseteq D \land vl \subseteq D \land vu \subseteq D \land vo \subseteq D \\
inv11 : vl \subseteq L0 \land vf \subseteq F0 \land vf \cap vl = \varnothing \\
inv14 : oldf \subseteq F0 \land oldl \subseteq L0 \\
inv16 : vl \cup oldl = L0 \\
inv17 : vf \cup oldf = F0 \\
inv18 : dom(l) = oldf \land ran(l) = oldl \land l \in oldf \rightarrowtail oldl \land l \in oldf \twoheadrightarrow oldl \\
inv19 : dom(f) = oldl \land ran(f) = oldf \land f \in oldl \rightarrowtail oldf \land f \in oldl \twoheadrightarrow oldf \\
inv20 : vo \cap vu = \varnothing \\
inv21 : vu \cup vo \cup vf \cup vl = D \\
inv22 : vf \cap oldf = \varnothing \land vl \cap oldl = \varnothing \\
inv24 : vo \cap vf = \varnothing \land vu \cap vf = \varnothing \\
inv26 : vo \cap vl = \varnothing \land vo \cap vf = \varnothing \\
inv28 : U0 \subseteq vo \cup vu \\
inv29 : O0 \subseteq vo \cup vu \\
inv30 : vf \cup oldf = F0 \\
inv31 : vl \cup oldl = L0 \\
inv32 : vf = F0 \setminus oldf \\
inv33 : vl = L0 \setminus oldl \\
inv34 : oldl = l[oldf] \\
inv35 : oldf = f[oldl] \\
inv36 : vu \cap vl = \varnothing \\
inv37 : vu \cap vf = \varnothing \\
inv38 : f = l^{-1} \land l = f^{-1} \\
inv40 : vl = \varnothing \land f \neq \varnothing \Rightarrow f \in L0 \rightarrowtail F0 \\
inv41 : vf = \varnothing \land l \neq \varnothing \Rightarrow l \in F0 \rightarrowtail L0 \\
inv42 : end \in BOOL \\
inv43 : end = FALSE \Rightarrow FF = F0 \land LL = L0 \\
inv44 : end = TRUE \land LL \neq \varnothing \Rightarrow FF = \varnothing \\
inv45 : end = TRUE \land FF \neq \varnothing \Rightarrow LL = \varnothing \\
inv46 : fol1 \in BOOL \\
inv47 : injfol \in L0 \nrightarrow F0 \\
inv48 : phase \in PHASES
\end{array}
$$

The invariant of Fig. 12 states that the two f and l are progressively built during the computing process. Moreover, it gives the relationship between these two functions: for instance, $f = l^{-1} \land l = f^{-1}$.

EVENT Dancing
ANY
x, y
WHERE
$grd1 : x \in vf \land y \in vl$
THEN
$act1 : vo := vo \cup \{x, y\}$
$act2 : vf := vf \setminus \{x\}$
$act3 : vl := vl \setminus \{y\}$
$act4 : oldf := oldf \cup \{x\}$
$act5 : oldl := oldl \cup \{y\}$
$act6 : f(y) := x$
$act7 : l(x) := y$
END

Event Dancing is guarded on the existence of both followers and leaders. In fact, the event modifies vo, vf, vl and is building both the injections f and l. It is still an abstract model and we are not yet sufficiently refined to merge in one unique concrete event.
The three next events do not modify vf and vl; they are modelling the three transition rules over vo and vu.

EVENT DancingFU
ANY
x, y
WHERE
$grd1 : x \in vf \land y \in vu$
THEN
$act1 : vo := vo \cup \{y\}$
$act2 : vu := vu \setminus \{y\}$
END

EVENT DancingLO
ANY
x, y
WHERE
$grd1 : x \in vl \land y \in vo$
THEN
$act1 : vo := vo \setminus \{y\}$
$act2 : vu := vu \cup \{y\}$
END

EVENT Dancing0U
ANY
x, y
WHERE
$grd1 : x \in vo \land y \in vu$
THEN
$act1 : vo := vo \cup \{y\}$
$act2 : vu := vu \setminus \{y\}$
END

Finally, the two events $\mathsf{PreFollowers}_2$ and $\mathsf{PreLeaders}_2$ are modelling the end of the construction of the injection: either f or l. We do not refine the events $\mathsf{Followers}_1$ and $\mathsf{Leaders}_1$ that will be refined in the last model into an event called Termination. In fact, these events are only observing the end of the process.

EVENT $\mathsf{PreLeaders}_2$
REFINES $\mathsf{PreLeaders}_1$
WHEN
$grd3 : vf = \varnothing \wedge vl \neq \varnothing$
$grd4 : end = FALSE$
WITNESSES
$i : i = l$
THEN
$act4 : U := vu$
$act5 : O := vo$
$act6 : F := vf$
$act7 : L := vl$
$act8 : end := TRUE$
END

EVENT $\mathsf{PreFollowers}_2$
REFINES $\mathsf{PreFollowers}_1$
WHEN
$grd4 : vl = \varnothing$
$grd5 : end = FALSE$
WITNESSES
$i : i = f$
THEN
$act4 : U := vu$
$act5 : O := vo$
$act6 : F := vf$
$act7 : L := vl$
$act8 : end := TRUE$
END

4.3.2 Liveness Properties

Now we consider the local liveness properties Φ_2 and how they refine Φ_1. This liveness refinement is proved as for the lights with the LATTICE rule. The new liveness properties are simply defined as follows:

– FAIR(DANCE$_2$) defines the three fairness assumptions required to prove each of the three Φ_2 properties, respectively:
  – WF$_h$(Dancing), where $h \mathrel{\widehat{=}} (vl, vf, vu, vo)$ is the frame of these assertions
  – WF$_h$(DancingFU)
  – $\mathsf{GFd}_h \stackrel{def}{=} \forall i, j.i, j \in 1..n \wedge i + j = card(O \cup U) \Rightarrow GF_h((D = L \oplus 0^i \oplus 1^j, \mathsf{DancingLO}, (D = L \oplus 0^{i-1} \oplus 1^{j+1}))$.
– The invariant is defined in Fig. 12. Define:

$DANCING(vl, vf, vu, vo)$
$\mathrel{\widehat{=}} partition(D, vl, vf, vu, vo)$, i.e. $D = L \oplus F \oplus O \oplus U$
$DANCINGF(vf, vu, vo)$
$\mathrel{\widehat{=}} partition(D, vf, vu, vo)$, i.e. $D = F \oplus O \oplus U$
$DANCINGL(vl, vu, vo)$
$\mathrel{\widehat{=}} partition(D, vl, vu, vo)$, i.e. $D = L \oplus O \oplus U$

– $\Phi_2 \stackrel{def}{=}$
$$\left[\begin{array}{l} \left(\begin{array}{l} DANCING(vl, vf, vu, vo) \\ \wedge vl \cup vf \neq \varnothing \end{array}\right) \\ \quad \rightsquigarrow \exists \begin{pmatrix} vl', \\ vf', \\ vu', \\ vo' \end{pmatrix} . \left(\begin{array}{l} DANCING(vl', vf', vu', vo') \\ \wedge vl' \cup vf' \subset vl \cup vf \end{array}\right) \\ \left(\begin{array}{l} DANCINGF(vf, vu, vo) \\ \wedge vu \neq \varnothing \end{array}\right) \\ \quad \rightsquigarrow \exists \begin{pmatrix} vf', \\ vu', \\ vo' \end{pmatrix} . \left(\begin{array}{l} DANCINGF(vf', vu', vo') \\ \wedge vu' \subset vu \end{array}\right) \end{array}\right]$$

– Proof that $\mathcal{S}pec(\mathsf{Dance}_2) \vdash \Phi_2$, i.e. relationship LIVE on the third line of Fig. 8, is simply derived as before by application of WF1 to each of Dancing and DancingFU:

PROVE:
$$\left(\begin{array}{l} DANCING(vl, vf, vu, vo) \\ \wedge vl \neq \varnothing \\ \wedge vf \neq \varnothing \end{array}\right) \rightsquigarrow \exists \begin{pmatrix} vl', \\ vf', \\ vu', \\ vo' \end{pmatrix} . \left(\begin{array}{l} DANCING(vl', vf', vu', vo') \\ \wedge vl' \cup vf' \subset vl \cup vf \end{array}\right)$$
APPLY: Rule WF1 applied to inductive action Dancing

► PROVE: For each event e of {Dancing, DancingOU, DancingFU, DancingLO},
$$\left(\begin{array}{l} DANCING(vl, vf, vu, vo) \\ \wedge vl \neq \varnothing \wedge vf \neq \varnothing \\ \wedge card(L_0) < card(F_0) \\ \wedge (BA(e)(h, h') \vee h' = h) \end{array}\right) \Rightarrow \begin{array}{l} DANCING(vl', vf', vu', vo') \\ \wedge vl' \cup vf' \subseteq vl \cup vf \end{array}$$

► PROVE:
$$\left(\begin{array}{l} DANCING(vl, vf, vu, vo) \\ \wedge vl \neq \varnothing \wedge vf \neq \varnothing \\ \wedge BA(\mathsf{Dancing})(h, h') \end{array}\right) \Rightarrow \begin{array}{l} DANCING(vl', vf', vu', vo') \\ \wedge\ vl' \cup vf' \subset vl \cup vf \end{array}$$

► PROVE:
$$\left(\begin{array}{l} DANCING(vl, vf, vu, vo) \\ \wedge vl \neq \varnothing \wedge vf \neq \varnothing \end{array}\right) \Rightarrow \text{ENABLED } \langle \mathsf{DANCING} \rangle_h$$

PROVE:
$$\left(\begin{array}{l} DANCINGF(vl, vf, vu, vo) \\ \wedge vu \neq \varnothing \end{array}\right) \rightsquigarrow \exists \begin{pmatrix} vl', \\ vf', \\ vu', \\ vo' \end{pmatrix} . \left(\begin{array}{l} DANCINGF(vl', vf', vu', vo') \\ \wedge\ vu' \subset vu \end{array}\right)$$
APPLY: Rule WF1 applied as above to inductive action DancingFU

– Φ_2 REF Φ_1

Φ_1 expresses the liveness properties at a very high level of abstraction via the four transitive leadsto properties to the interim and final configurations, for each case as per Fig. 11. Model Dance_2 introduces the protocol rules as iterations based initially on the elimination of pairs of dancers by event Dancing. Φ_2 REF Φ_1 is ensured by three applications of the LATTICE rule. This requires the identification of the *inductive* properties below on well-founded relations seen in Φ_2.

In order to prove the fourth property of Φ_1, i.e. $D = L \oplus O \oplus U \rightsquigarrow D = L \oplus U$, we might hope to add the following leadsto property to Φ_2:

$$\begin{pmatrix} DANCINGL(vl, vu, vo) \\ \wedge vl \neq \varnothing \end{pmatrix} \rightsquigarrow \exists \begin{pmatrix} vl', \\ vu', \\ vo' \end{pmatrix} . \begin{pmatrix} DANCINGL(vl', vu', vo') \\ \wedge vo' \subset vo \end{pmatrix}$$

This property does not hold because of the peculiar demonic character of the rule $0 \leftrightarrow 1 \Rightarrow 0 \leftrightarrow 0$, which by consuming 1's acts against the rule $1 \leftrightarrow 0 \Rightarrow 1 \leftrightarrow 1$ that we wish to converge. See Sect. 4.3.3 below for discussion on how we prove the convergence case to $D = L \oplus U$ in the "Appendix".

$$\text{PROVE:} \quad \left\{ \begin{pmatrix} DANCING(vl, vf, vu, vo) \\ \wedge vl \cup vf \neq \varnothing \end{pmatrix} \rightsquigarrow \exists \begin{pmatrix} vl', \\ vf', \\ vu', \\ vo' \end{pmatrix} . \begin{pmatrix} DANCING(vl', vf', vu', vo') \\ \wedge vl' \cup vf' \subset vl \cup vf \end{pmatrix} \right\} \vdash \begin{pmatrix} D = F \oplus L \oplus O \oplus U \\ \wedge \exists i.i \in L0 \rightarrowtail F0 \end{pmatrix} \rightsquigarrow D = F \oplus O \oplus U$$

PROOF: By temporal rule LATTICE for event Dancing. □

$$\text{PROVE:} \quad \left\{ \begin{pmatrix} DANCINGF(vf, vu, vo) \\ \wedge vu \neq \varnothing \end{pmatrix} \rightsquigarrow \exists \begin{pmatrix} vf', \\ vu', \\ vo' \end{pmatrix} . \begin{pmatrix} DANCINGF(vf', vu', vo') \\ \wedge vu' \subset vu \end{pmatrix} \right\} \vdash D = F \oplus O \oplus U \rightsquigarrow D = F \oplus O$$

PROOF: By temporal rule LATTICE for event DancingFU. □

$$\text{PROVE:} \quad \left\{ \begin{pmatrix} DANCING(vl, vf, vu, vo) \\ \wedge vl \cup vf \neq \varnothing \end{pmatrix} \rightsquigarrow \exists \begin{pmatrix} vl', \\ vf', \\ vu', \\ vo' \end{pmatrix} . \begin{pmatrix} DANCING(vl', vf', vu', vo') \\ \wedge vl' \cup vf' \subset vl \cup vf \end{pmatrix} \right\} \vdash \begin{pmatrix} D = F \oplus L \oplus O \oplus U \\ \wedge \exists i.i \in F0 \rightarrowtail L0 \wedge i[F0] \neq L0 \end{pmatrix} \rightsquigarrow D = L \oplus O \oplus U$$

PROOF: By temporal rule LATTICE for event Dancing. □

4.3.3 General fairness assumption for dancing

The proof of the property $D = L \oplus O \oplus U \rightsquigarrow D = L \oplus U$ is given in the "Appendix". The proof is intricate and requires a new, stronger fairness assumption than those seen so far. We define an appropriate fairness assumption called the *general fairness assumption* in PP. The Event-B model of the previous section expresses the population protocol rules and does not express any assumptions over executions or scheduling. Indeed, such statements are not possible in Event-B, which is a language of single-step state transitions. We now analyse fairness conditions to prove that the protocol reaches $D = L \oplus O \oplus U \rightsquigarrow D = L \oplus U$.

The configurations are of the form $O^i 1^j$ with $i + j = n$ and the actions are defining the following graph over those configurations:

$$O^n \leftrightarrows O^{n-1}1^1 \leftrightarrows O^{n-2}1^2 \leftrightarrows \cdots \leftrightarrows O^2 1^{n-2} \leftrightarrows O 1^{n-1} \rightarrow 1^n$$

The assumption called general fairness is very strong, since it allows to derive that the system will get out the possible loop. The proof is possible because of the finiteness of the set of possible configurations.

As before there are only two enabled events in this case: DancingLO, DancingOU. However, this proof is a more complex argument than case 1 and requires a richer fairness assumption because of the way DancingOU consumes the ONEs produced by DancingLO. We see looping transitions through the intermediate configurations and note that the configuration $D = L \oplus U$ is stable once reached, because DancingOU becomes disabled by the absence of ZEROs:

$$L \oplus 0^i \oplus 1^j \overset{DancingLO}{\longrightarrow} L \oplus 0^{i-1} \oplus 1^{j+1} \overset{DancingLO}{\longrightarrow} \cdots L \oplus 1^{i+j}$$
$$L \oplus 0^i \oplus 1^j \overset{DancingOU}{\longleftarrow} L \oplus 0^{i-1} \oplus 1^{j+1}$$

General fairness states that if a configuration C appears infinitely often in a sequence of configurations, and if $C \longrightarrow C'$, then C' should appear also infinitely often in the sequence.

The proof rule is based on the expression of the *general fairness* assumption which is a special strong fairness assumption:

$$\mathrm{GF}_f(B, E, C) \overset{def}{=} \Diamond\Box\neg Enabled\langle B \wedge E \wedge C' \rangle_f \vee \Box\Diamond\langle B \wedge E \wedge C' \rangle_f$$

For event E and configurations B, C, $\mathrm{GF}_f(B, E, C)$ means that, if E is infinitely often enabled at B and if E can transition to C, then C will occur infinitely often by execution of E. General fairness is incorporated into our general fairness rule GFd_h applied to the dancers:

$$\mathsf{GFd}_h \overset{def}{=} \forall i, j \cdot i, j \in 1..n \wedge i + j = card(O \cup U) = n \Rightarrow GF_h(D = L \oplus 0^i \oplus 1^j, \mathsf{DancingLO}, D = L \oplus 0^{i-1} \oplus 1^{j+1})$$

The target configuration is the configuration in which there is no more O element. The general fairness assumption means that each triple of configuration of the sequence above is infinitely often enabled, since the number of triples

is finite. Intuitively, under the general fairness, the target configuration is reached.

We propose a new rule GF1 for deriving liveness properties under the general fairness assumption. The rule is based on the WF1 and SF1 rules of Lamport [51]. It extends WF1 with another configuration: from B we may progress to B', take the inductive step to C' or reach another configuration A' which works against the inductive process. This "counter-inductive" step is itself counteracted by an assumption that, given GF_f, $A \rightsquigarrow B$. E should be enabled in B.

$$\mathrm{GF1}\quad \frac{\begin{array}{l} B \wedge [N]_f \Rightarrow (B' \vee C' \vee A') \\ B \wedge \langle N \wedge E\rangle_f \Rightarrow C' \\ \Box[N]_f \wedge \mathrm{GF}_f(B,E,C) \Rightarrow (A \rightsquigarrow B) \\ B \Rightarrow \mathit{Enabled}\langle E\rangle_f \end{array}}{\Box[N]_f \wedge \mathrm{GF}_f(B,E,C) \Rightarrow (B \rightsquigarrow C)}$$

The general fairness assumption is defined over (configuration, event, configuration) tuples, unlike the classical fairness assumptions made on *actions* or *events* in TLA. The classical WF/SF proof rules of TLA are not enough to prove the reachability of the case 2 configuration.

The soundness of this rule is proved by showing that traces (sequences of configurations) generated by N under the assumption of fairness GF_f are ensuring the $\rightsquigarrow$ property.

PROOF **From** assumptions (1–4) we will **infer**: $\Box[N]_f \wedge \mathrm{GF}_f \Rightarrow (B \rightsquigarrow C)$

Let a sequence of configurations for a given system be generated by N with state variables f. We assume that E is a special action of the system. Action E is executed under the general fairness assumption and it can be one of the actions amongst $a_0, \ldots, a_i, \ldots$:

$$D_0 \xrightarrow{a_0} D_1 \xrightarrow{a_1} D_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{i-1}} D_i \xrightarrow{a_i} D_{i+1} \xrightarrow{a_{i+1}} \cdots$$

We assume that B, C and A are three state predicates. We interpret these predicates over configurations by writing $B(D_i)$ as B_i and similarly $A(D_i)$ as A_i, $C(D_i)$ as C_i.

Let us assume that the sequence of configurations satisfies $\Box[N]_f \wedge \mathrm{GF}_f$ and that B holds for some configuration D_{i_1}: B_{i_1}. We will build a (sub-)sequence of B's :

$$B_{i_1} \ldots B_{i_2} \ldots B_{i_j} \ldots B_{i_{j+1}} \ldots$$

We assume that between no two Bs does a C occur, and argue by contradiction.

We know according to (2) that there is a possible transition by E to C and we know that the next state is either A, B or C by (1). Since between two Bs we assume there is no C, there must be either B or A, giving

$$B_{i_1}(B \vee A)$$

B will be followed by either B or A as above, and from (3) we know A will be followed in a finite number of steps by B, giving

$$B_{i_1}(B)^* A(\ldots) B_{i_2}$$

Inductively this extends to

$$B_{i_1}(B)^* A(\ldots) B_{i_2}(B)^* A(\ldots) B_{i_j}(B)^* A(\ldots) B_{i_{j+1}}(B)^* A(\ldots)$$

Thus we have built an infinite sub-sequence of Bs with configurations D in which the event E is enabled with a possible transition to C, but no C arises. This contradicts the general fairness assumption on (B, E, C). Hence C will eventually appear. □

4.4 Generating the population protocol from refinement

The last model is called Protocol. Its data refine away intermediate variables; the main events are:

```
EVENT Termination
REFINES PreLeaders2PreFollowers2
WHEN
  grd3 : vf = ∅ ∨ vl = ∅
  grd5 : end = FALSE
THEN
  act4 : U := vu
  act5 : O := vo
  act6 : F := vf
  act7 : L := vl
  act8 : end := TRUE
END
```

```
EVENT Dancing
REFINES Dancing
ANY
  x, y
WHERE
  grd1 : x ∈ vf
  grd2 : y ∈ vl
THEN
  act1 : vo := vo ∪ {x, y}
  act2 : vf := vf \ {x}
  act3 : vl := vl \ {y}
END
```

Event Termination models the global termination of the process; it is not an action of the protocol itself but only an observation by a global observer. The condition *end* is set to true at this step. It refines two events by merging them. Event Dancing is transformed into the rule for the population protocol: $\mathsf{f} \leftrightarrow \mathsf{l} \Rightarrow \mathsf{0} \leftrightarrow \mathsf{0}$.

Event DancingFU remains unchanged and is interpreted as the population protocol rule: $\mathsf{f} \leftrightarrow \mathsf{1} \Rightarrow \mathsf{f} \leftrightarrow \mathsf{0}$. Similarly, events DancingLO and DancingOU remain unchanged and are interpreted as rules $\mathsf{l} \leftrightarrow \mathsf{0} \Rightarrow \mathsf{l} \leftrightarrow \mathsf{1}$ and $\mathsf{0} \leftrightarrow \mathsf{1} \Rightarrow \mathsf{0} \leftrightarrow \mathsf{0}$ respectively.

5 Analysing leader election in population protocols

5.1 Introduction

Leader election is a significant mechanism for distributed systems. Algorithms based on a central coordinator exist for many problems. The existence, selection, reliability and notification of such a leader node to all other nodes is

an important area of study, and has been examined in a population protocol setting [14,23,30,37]. We consider the example of the *eventual leader detector* [37] population protocol, a version of the *failure detector*, which is a class of diagnostic devices for reporting failure information to nodes. The *eventual leader detector* is an oracle which reports periodically to every node *a guess about whether a leader exists in the network or not*. The *guess* need not be consistent; different guesses may be reported to different nodes. This is a *self-stabilising* algorithm, an execution of which "converges to a set of pre-defined stable configurations starting from any arbitrary configuration" [37]. A guarantee of *self-stabilisation* is given: in any trace, if from some point there is either always a leader, or always no leader, the oracle will eventually become reliable, i.e. report the fact accurately to each agent. This is stated more precisely [37]:

- If all but finitely many configurations of the trace lack a leader, then each process receives input `false` at all but finitely many steps
- If all but finitely many configurations of the trace contain one or more leaders, then each process receives input `true` at all but finitely many steps

Using the eventual leader detector, we model the self-stabilising leader election algorithm for a complete network graph, as defined by Fischer and Jiang [37]. We can regard this model as abstracting over communication unreliability in a lossy network: if each node expects periodic notification of the existence or identity of a leader, this will be subject to a timeout. In an unstable network, nodes and links may fail intermittently or permanently, alternative routing will be sought, and timeouts will be breached. In a stable network the timeouts will be honoured and the leader notifications (oracle inputs to nodes) will be reliable.

This formulation is interesting: in the unstable network, there is a high degree of nondeterminism in the status of nodes and their awareness of leader existence or nonexistence. Stabilisation for [37] takes place in any trace where there is either always a leader, or always no leader. The latter possibility is an artefact of their proof; the fairness assumptions we will make will ensure that there is eventually always a leader. Thus we interpret stabilisation as the point at which the oracle has become reliable *and* a leader exists; this realises the two self-stabilising conditions above. The oracle may well be reliable *before* stabilisation, i.e. before a leader exists. A leader may exist before the oracle is known to be reliable. The state of network nodes can change nondeterministically before stabilisation. We denote stabilisation by flag $stable = TRUE$ in models SEL_2 and beyond. Nondeterminism remains after stabilisation, in the dynamics of leader election.

Please note that there is some complexity in the two uses we make of nondeterminism in this example. We use it (i) to model instability in the network and (ii) for modelling convenience in avoiding duplicate variables in data refinement, similarly to the lights model PPM_0 in Sect. 3.1. This leads to more nondeterministic event nondet transitions in the proof diagrams than are necessary, and these are sometimes omitted from the diagrams to avoid clutter. These transitions do not affect the proof reasoning. In the last refinement model SEL_5 they are finally refined away.

We model the problem through four refinements; see Fig. 13. Machines $\mathrm{SEL}_1, \ldots, \mathrm{SEL}_5$ progressively define the leader election protocol under specific fairness constraints. $E_1, \ldots, E_5$ list the eventuality properties and safety properties. C-SEL is the context of the global development.

5.2 Specification of the leader election (SEL_1)

The first machine has two events. Event $\mathsf{election}_1$ models a one-shot election process; event nondet_1 models the nondeterministic activity of the environment and will be refined in further steps. The environment acts either positively or negatively with respect to the goal of *electing a leader*, thus giving a closed model.

Context C-SEL defines the node set C and their states taken from set $S \mathrel{\widehat{=}} \{L, N\}$. Machine SEL_1 has two variables: $done$ (initialised to FALSE) to control the election and s to describe the current state of the network. Here the initial configuration is completely nondeterministic and may have no leader at all.

EVENT $\mathsf{election}_1$
ANY
c, ns
WHERE
$grd0 : done = FALSE$
$grd3 : c \in C$
$grd4 : ns \in C \to S$
$grd5 : \forall d \cdot d \in C \wedge d \neq c \Rightarrow ns(d) = N$
$grd6 : ns(c) = L$
THEN
$act1 : s := ns$
$act2 : done := TRUE$
END

EVENT nondet_1
WHEN
$grd1 : done = FALSE$
THEN
$act1 : s :\in C \to S$
END

The two properties E_1 of this model are:

- liveness property: $\neg\, done \rightsquigarrow done$
- safety property (and invariant):
  $\Box(done = TRUE \Rightarrow \exists c \cdot (s(c) = L \wedge (\forall d \cdot d \in C \wedge d \neq c \Rightarrow s(d) = N))$
  This states that $done = TRUE$ is the protocol termination flag: the leader is elected.

The safety property is derived from the Event-B machine and the liveness property is simply derived by WF1, assuming WF($\mathsf{election}_1$). The next model will refine the current model by introducing the notion of stability: the election is possible when the network is stable. Before stabilisation event nondet_1 may lead to a configuration without leaders or an unstable configuration. The model SEL_1 defines the service that the system should ensure and it is a very abstract expression with a very simple fairness assumption introduced by WF1 rule.

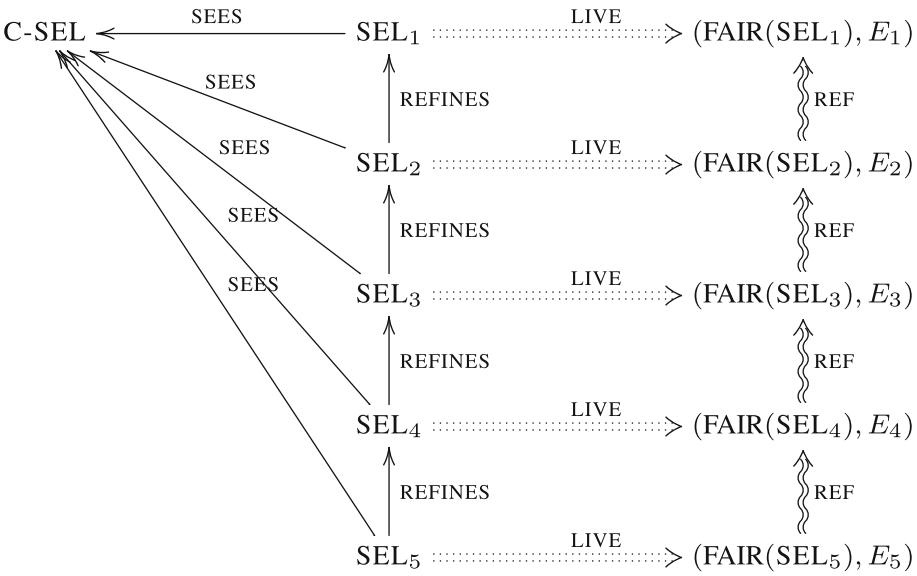


Fig. 13 Development scheme for leader election

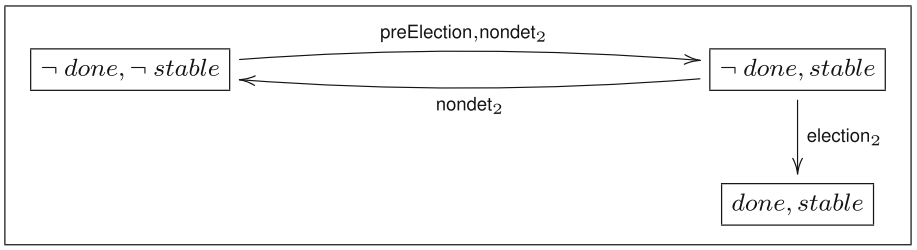


Fig. 14 SEL_2—stability

5.3 Stability and instability of the network (SEL_2)

The first refinement SEL_2 introduces the fact that the system may be in an *unstable* state. In our interpretation, the network is in a *stable* state at the time after stabilisation, when the oracle has become reliable; this is reflected in the invariant property in the second refinement SEL_3. Fig. 14 gives the liveness proof diagram and includes implicit nondet_2 self-loop transitions on each of the two upper states.

Events are given in the following. The liveness properties are:

- $\neg\ done \wedge \neg\ stable \leadsto \neg\ done \wedge stable$: ensured by event preelection under weak fairness
- $\neg\ done \wedge stable \leadsto done \wedge stable$: ensured by event $\mathsf{election}_2$ under strong fairness

These properties are combined as $E2$ below, and used to infer the property $E1 \stackrel{def}{=} \neg\ done \leadsto done$ of the abstract model, i.e. that $E2$ REF $E1$. We use the inference rules for liveness properties to decompose the system. The refinement machine SEL_2 is defined by case analysis: the system is either stable or not. A new event preElection is added which modifies the new *stable* flag and indicates that this is the stabilisation phase. In this abstraction we conflate stabilisation with the existence of a leader; prestabilisation—creation of a leader before stabilisation—will appear in a later refinement. We assume that when stability is obtained, we *keep* the system in a stable configuration by means of *stable*. Although refined event nondet_2 can destabilise again in this model, this behaviour will be refined away in the final model SEL_5. The Event-B machines are used for analysing the election process and allow us to express assumptions.

EVENT $\mathsf{preElection}_2$ REFINES nondet_1
WHEN
$grd2 : done = FALSE$
THEN
$act2 : s, stable :| \begin{pmatrix} s' \in C \to S \\ \wedge\ s'^{-1}[\{L\}] \neq \varnothing \\ \wedge\ stable' = TRUE \end{pmatrix}$
END
EVENT nondet_2 REFINES nondet_1
WHEN
$grd1 : done = FALSE$
THEN
$act1 : s :\in C \to S$
$act2 : stable :\in BOOL$
END

EVENT election$_2$ REFINES election$_1$
ANY
c, ns
WHERE
$grd0 : done = FALSE$
$grd3 : c \in C$
$grd4 : ns \in C \rightarrow S$
$grd5 : \forall d \cdot d \in C \wedge d \neq c \Rightarrow ns(d) = N$
$grd6 : ns(c) = L$
$grd21 : stable = TRUE$
$grd22 : s^{-1}[\{L\}] \neq \varnothing$
$grd23 : c \in s^{-1}[\{L\}]$
THEN
$act1 : s := ns$
$act2 : done := TRUE$
END

The model SEL_2 is still an abstraction of the final protocol and we are preparing the introduction of local information. The oracle is introduced in the next refinement since it helps nodes to get global information.

- FAIR(SEL_2) defines the fairness assumptions over the event preElection$_2$:
  WF $_{s,stable}$(preElection$_2$) $\wedge$ SF $_{s,stable}$(election$_2$)
- $E_2 \stackrel{def}{=} \{\neg\, done \wedge \neg\, stable \rightsquigarrow \neg\, done \wedge stable,$
  $\begin{pmatrix} \neg\, done \wedge \neg\, stable \\ \vee \\ \neg\, done \wedge stable \end{pmatrix} \rightsquigarrow done \wedge stable\}.$
- We prove that $\neg\, done \wedge \neg\, stable \rightsquigarrow \neg\, done \wedge stable$ by WF1 as before.
- We prove that $\begin{pmatrix} \neg\, done \wedge \neg\, stable \\ \vee \\ \neg\, done \wedge stable \end{pmatrix} \rightsquigarrow done \wedge stable$ by applying the PP-SF1 rule of Sect. 2.1 using the following hypotheses (using x as shorthand for $s, stable$). The hypotheses are straightforwardly derived in RODIN from the Event-B models as before.

$$H0\ \neg\, done \equiv (\neg\, done \wedge \neg\, stable) \vee (\neg\, done \wedge stable)$$
$$H1\ \neg\, done \wedge \neg\, stable \wedge [N]_x \Rightarrow (\neg\, done' \wedge \neg\, stable') \vee (\neg\, done' \wedge stable')$$
$$H2\ \neg\, done \wedge \neg\, stable \wedge \langle N \wedge \text{preElection}_2 \rangle_x \Rightarrow \neg\, done' \wedge stable'$$
$$H3a\ \neg\, done \wedge \neg\, stable \Rightarrow Enabled\langle \text{preElection}_2 \rangle_x$$
$$H3b\ \neg\, done \wedge stable \Rightarrow Enabled\langle \text{election}_2 \rangle_x$$
$$H4\ \neg\, done \wedge [N]_x \Rightarrow (\neg\, done' \vee done')$$
$$H5\ \neg\, done \wedge \langle N \wedge \text{election}_2 \rangle_x \Rightarrow done' \qquad \square$$

In this model the system remains very unstable and can go back to unstable states because of event nondet$_2$. Event election$_2$ is infinitely often enabled and is eventually executed. The next refinement introduces the *oracle* which is the means by which nodes get eventual reliable knowledge about leader status. We model the instability of the oracle being unreliable before stabilisation.

5.4 Introducing the oracle (SEL_3)

The new variable *oracle* informs the nodes whether there is at least one leader or not. Moreover, in liveness reasoning we use a new shorthand predicate called *ldr* meaning that *ldr* holds, when there is at least one node which is a leader, i.e. $s^{-1}[\{L\}] \neq \varnothing$. We model oracle unreliability prior to stabilisation. For simplicity we assume a single global oracle value (rather than local ones) at any time. The invariant relates *oracle*, *stable* and s by requiring the *oracle* to report reliably when the system is stable, the presence or absence of a leader:

INVARIANTS
$inv31 : oracle \in BOOL$
$inv32 : stable = TRUE \wedge s^{-1}[\{L\}] = \varnothing \Rightarrow oracle = FALSE$
$inv33 : stable = TRUE \wedge s^{-1}[\{L\}] \neq \varnothing \Rightarrow oracle = TRUE$

The event nondet$_3$ still models the nondeterministic environment and is subject to fairness assumptions. It adds implicit transitions which may make the system less stable. nondet$_3$ is a refined event and integrates the new variable *oracle*, preserving the invariant relating *stable*, *ldr* and *oracle*.

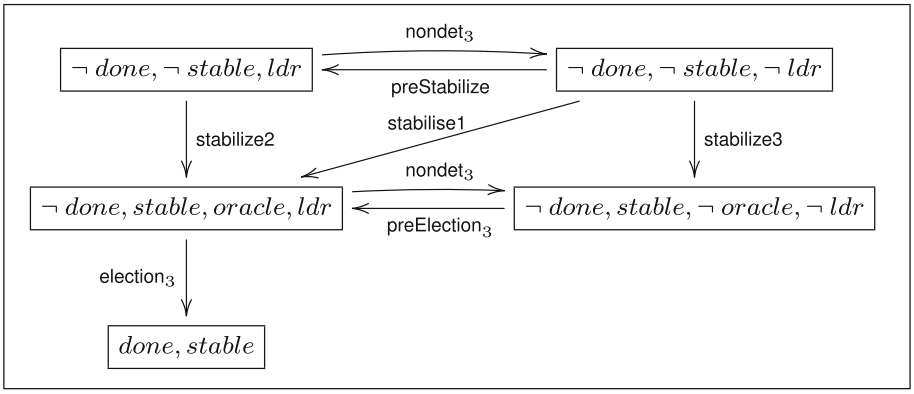
EVENT nondet$_3$ REFINES nondet$_2$
WHEN
$grd1 : done = FALSE$
THEN
$act4 : s, stable, oracle :| \begin{pmatrix} s' \in C \rightarrow S \wedge stable' \in BOOL \\ \wedge (stable' = TRUE \wedge s'^{-1}[\{L\}] = \varnothing \Rightarrow oracle' = FALSE) \\ \wedge (stable' = TRUE \wedge s'^{-1}[\{L\}] \neq \varnothing \Rightarrow oracle' = TRUE) \end{pmatrix}$
END

The two main fairness assumptions are borrowed from the assumptions suggested by Fischer and Jiang [37] and are expressed in a temporal way:

- If all but finitely many configurations of the trace lack a leader, then each node receives oracle `false` at all but finitely many steps: $\Diamond\Box\neg\, ldr \Rightarrow \Diamond\Box(oracle = FALSE)$
- If all but finitely many configurations of the trace contain one or more leaders, then each node receives oracle `true` at all but finitely many steps:
  $\Diamond\Box ldr \Rightarrow \Diamond\Box(oracle = TRUE)$

Event election$_3$ is unchanged and still models the one-shot election. Next we analyse the fairness requirements for ensuring that the system will eventually reach the state which is stable with oracle. A first solution (see Fig. 15) is to assign strong fairness to each event amongst stabilize1, stabilize2, stabilize3, preStabilize and preElection$_3$. We

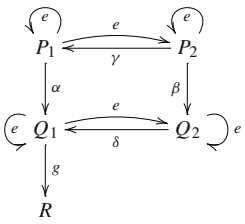
Fig. 15 SEL_3: oracle



must prove that each of the three top right $\neg\ done$ configurations of Fig. 15 leads to $\neg\ done, stable, oracle, ldr$. In the diagram, we indicate the possible moves between predicates, and we have mentioned two transitions for the nondet$_3$ event. In fact this event transitions implicitly from any predicate to any other predicate—as explained in Sect. 5.1—subject to invariant preservation. This is implicit to avoid clutter of the diagram. The nondet$_3$ event captures the guesses of the oracle about the existence of a leader, before stabilisation.

We set up the fairness and liveness conditions for this model to fit the proof rule PP-SF2 of Sect. 2.

Proof Pattern PP-SF2



We list the fairness assumptions and liveness properties. Each SEL_3 event below is annotated with the corresponding transition name in PP-SF2:

$$\text{FAIR}(SEL_3) \stackrel{def}{=} \text{WF}_x(\text{preStabilize} - \gamma) \wedge \text{SF}_x(\text{stabilize2} - \alpha) \wedge \text{WF}_x(\text{stabilize3} - \beta) \wedge \text{WF}_x(\text{preElection} - \delta) \wedge \text{WF}_x(\text{stabilize1}) \wedge \text{SF}_x(\text{election}_3 - g)$$

Each line of FAIR(SEL_3) corresponds to each liveness property $E_{31} - E_{33}$ in turn (replacing the $\wedge$ operator with commas for brevity) below by defining:

$$P_1 \stackrel{def}{=} \neg done, \neg stable, ldr$$
$$P_2 \stackrel{def}{=} \neg done, \neg stable, \neg ldr$$
$$Q_1 \stackrel{def}{=} \neg done, stable, oracle, ldr$$
$$Q_2 \stackrel{def}{=} \neg done,\ stable, \neg oracle, \neg ldr$$
$$R \stackrel{def}{=} done, stable$$

$$E_{31} \stackrel{def}{=} \{P_2 \leadsto\ (P_1 \vee Q_2)\}$$
$$E_{32} \stackrel{def}{=} \{Q_2 \leadsto\ Q_1\}$$
$$E_{33} \stackrel{def}{=} \{(P_1 \vee P_2 \vee Q_2) \leadsto\ Q_1\}$$
$$E_{34} \stackrel{def}{=} \{(P_1 \vee P_2 \vee Q_2 \vee Q_1) \leadsto\ R\}$$

By way of informal proof it suffices to observe that PP-SF2 is basically the superposition of two PP-SF1 proofs, the hypotheses being instantiated exactly in Sect. 5.3. We have proved the correctness of this construction in Sect. 2.1 and we understand the role of the string fairness in this example.

EVENT stabilize1
REFINES preElection$_2$
WHEN
$grd2 : done = FALSE$
$grd30 : stable = FALSE$
$grd31 : s^{-1}[\{L\}] = \varnothing$
THEN
$$act2 : s, stable :| \begin{pmatrix} s' \in C \to S \\ \wedge s'^{-1}[\{L\}] \neq \varnothing \\ \wedge stable' = TRUE \end{pmatrix}$$
$act30 : oracle := TRUE$
END

EVENT stabilize2
REFINES preElection$_2$
WHEN
$grd2 : done = FALSE$
$grd30 : stable = FALSE$
$grd31 : s^{-1}[\{L\}] \neq \varnothing$
THEN
$$act2 : s, stable :| \begin{pmatrix} s' \in C \to S \\ \wedge s'^{-1}[\{L\}] \neq \varnothing \\ \wedge stable' = TRUE \end{pmatrix}$$
$act30 : oracle := TRUE$
END

EVENT stabilize3
REFINES nondet$_2$
WHEN
$grd30 : done = FALSE$
$grd31 : stable = FALSE$
$grd32 : s^{-1}[\{L\}] = \varnothing$
THEN
$act2 :$
$$s, stable, oracle :| \begin{pmatrix} s' \in C \to S \\ \wedge s'^{-1}[\{L\}] = \varnothing \\ \wedge stable' = TRUE \\ \wedge oracle' = FALSE \end{pmatrix}$$
END

EVENT preStabilize
REFINES nondet_2
WHEN
$grd30 : done = FALSE$
$grd31 : stable = FALSE$
$grd32 : s^{-1}[\{L\}] = \varnothing$
THEN
$$act2s, stable, oracle : | \begin{pmatrix} s' \in C \to S \\ \wedge s'^{-1}[\{L\}] \neq \varnothing \\ \wedge stable' = FALSE \\ \wedge oracle' \in BOOL \end{pmatrix}$$
END
EVENT preElection_3
REFINES preElection_2 WHEN
$grd2 : done = FALSE$
$grd30 : stable = TRUE$
$grd31 : s^{-1}[\{L\}] = \varnothing$
$grd32 : oracle = FALSE$
THEN
$$act2 : s, stable : | \begin{pmatrix} s' \in C \to S \\ \wedge s'^{-1}[\{L\}] \neq \varnothing \\ \wedge stable' = TRUE \end{pmatrix}$$
$act30 : oracle := TRUE$
END

The current model is still a very abstract description of the system and we still need to introduce the protocol rules of computation. The next two refinements refine the event nondet by these rules of computation. Note that the event nondet is always observable but for simplicity, is not completely included in Fig. 15. It may require strong fairness.

5.5 Adding protocol rules 2 and 3 (SEL_4)

The algorithm of Fischer and Jiang [37] is described by the three following rules. A configuration is defined locally by the state of the node (L or N) and by the oracle value (T or F). $\star$ denotes a don't-care value:

- `rule 1`: $((L,\star),(L,\star)) \longrightarrow ((L),(N))$: when two leaders interact, then one of the two leaders becomes a non-leader
- `rule 2`: $((N,F),(N,\star)) \longrightarrow ((L),(N))$: when two non-leaders interact in a configuration where one sees a False oracle, then that non-leader becomes a leader
- `rule 3`: $((N,T),(N,\star)) \longrightarrow ((N),(N))$: when two non-leaders interact in a configuration where one sees a True oracle, then no change occurs

INVARIANTS
$inv41 : l \subseteq C$
$inv42 : n \subseteq C$
$inv43 : l \cap n = \varnothing$
$inv44 : l \cup n = C$
$inv45 : l = s^{-1}[\{L\}]$
$inv46 : n = s^{-1}[\{N\}]$

Two new variables are introduced in this refinement, i.e. n and l which state that a node i is either N ($i \in n$) or L ($i \in l$). They are useful for introducing local events and rules of computation [37]. We introduce rule 2, which reduces the variant n, the number of non-leaders. We also introduce rule 3, which skips.

The events of the new model can be classified into two groups:

- The first group is a set of events which model the *reaction* of the network when the environment changes: either stabilising the system (events stabilize1–3) or otherwise (event nondet_4)
- The second group is the set of events which model the protocol itself: rules 2 and 3 in their prestabilisation (rule2PreElection, rule2PreStabilize) as well as stabilised (rule2, rule3) forms

EVENT nondet_4 REFINES nondet_3
WHEN
$grd1 : done = FALSE$
THEN
$act40 : s, stable, oracle, l, n : |$
$$\begin{pmatrix} s' \in C \to S \\ \wedge stable' \in BOOL \\ \wedge (stable' = TRUE \wedge s'^{-1}[\{L\}] = \varnothing \Rightarrow oracle' = FALSE) \\ \wedge (stable' = TRUE \wedge s'^{-1}[\{L\}] \neq \varnothing \Rightarrow oracle' = TRUE) \\ \wedge l' \subseteq C \wedge n' \subseteq C \\ \wedge l' \cap n' = \varnothing \ \wedge \ l' \cup n' = C \\ \wedge l' = s'^{-1}[\{L\}] \ \wedge \ n' = s'^{-1}[\{N\}] \end{pmatrix}$$
END
EVENT stabilize1 REFINES stabilize1
WHEN
$grd1 : done = FALSE$
$grd2 : stable = FALSE$
$grd3 : s^{-1}[\{L\}] = \varnothing$
THEN
$$act1 : s, l, n : | \begin{pmatrix} s' \in C \to S \\ \wedge l' \subseteq C \wedge n' \subseteq C \\ \wedge l' \cap n' = \varnothing \ \wedge \ l' \cup n' = C \\ \wedge l' = s'^{-1}[\{L\}] \ \wedge n' = s'^{-1}[\{N\}] \\ \wedge s'^{-1}[\{L\}] \neq \varnothing \end{pmatrix}$$
$act2 : stable := TRUE$
$act3 : oracle := TRUE$
END

EVENT stabilize2 REFINES stabilize2
WHEN
$grd1 : done = FALSE$
$grd2 : stable = FALSE$
$grd3 : s^{-1}[\{L\}] \neq \varnothing$
THEN
$$act1 : s, l, n : | \begin{pmatrix} s' \in C \to S \\ \wedge l' \subseteq C \wedge n' \subseteq C \\ \wedge l' \cap n' = \varnothing \\ \wedge l' \cup n' = C \\ \wedge l' = s'^{-1}[\{L\}] \\ \wedge n' = s'^{-1}[\{N\}] \\ \wedge s'^{-1}[\{L\}] \neq \varnothing \end{pmatrix}$$
$act2 : stable := TRUE$
$act3 : oracle := TRUE$
END
EVENT stabilize3 REFINES stabilize3
WHEN
$grd30 : done = FALSE$
$grd31 : stable = FALSE$
$grd32 : s^{-1}[\{L\}] = \varnothing$
THEN
$act1 : stable := TRUE$
$act2 : oracle := FALSE$
END

EVENT rule2 REFINES nondet$_3$
ANY
i, j
WHERE
$grd1 : stable = FALSE$
$grd2 : s^{-1}[\{L\}] \neq \varnothing$
$grd3 : done = FALSE$
$grd4 : oracle = FALSE$
$grd5 : i \in n \wedge j \in n$
$grd7 : i \neq j$
THEN
$act1 : l := l \cup \{i\}$
$act2 : n := n \setminus \{i\}$
$act3 : s(i) := L$
$act4 : oracle :\in BOOL$
END

EVENT rule2preElec REFINES preElection
ANY
i, j
WHERE
$grd1 : stable = TRUE$
$grd2 : s^{-1}[\{L\}] = \varnothing$
$grd3 : done = FALSE$
$grd4 : oracle = FALSE$
$grd5 : i \in n \wedge j \in n$
$grd6 : i \neq j$
THEN
$act1 : l := l \cup \{i\}$
$act2 : n := n \setminus \{i\}$
$act3 : s(i) := L$
$act4 : oracle := TRUE$
END

EVENT rule2preStab REFINES preStabilize
ANY
i, j
WHERE
$grd1 : stable = FALSE$
$grd2 : s^{-1}[\{L\}] = \varnothing$
$grd3 : done = FALSE$
$grd4 : oracle = FALSE$
$grd5 : i \in n \wedge j \in n$
$grd7 : i \neq j$
THEN
$act1 : l := l \cup \{i\}$
$act2 : n := n \setminus \{i\}$
$act3 : s(i) := L$
$act4 : oracle :\in BOOL$
END
EVENT rule3
ANY
i, j
WHERE
$grd1 : done = FALSE$
$grd2 : oracle = TRUE$
$grd5 :: i \in n \wedge j \in n$
$grd5 : i \neq j$
THEN
$skip$
END

The remaining rule to introduce in the next refinement is rule 1. We keep nondet in our current model; next we can refine it to identify completely what are the rules of the protocol and what are the assumptions over the environment. The reader will notice that rule 3 is not very useful but it was part of the set of rules by Fischer and Jiang [37]. For this refinement, we keep the same fairness assumptions and we simply modify the names of the new events. For instance, rule 2 is under strong fairness (Fig. 16).

5.6 Deriving rules (SEL$_5$)

The final refinement introduces the third and final rule of the protocol, i.e. rule 1, as a refinement of nondet$_4$. The fairness assumption over this new event rule1 is simply WF (rule1). We retain the nondeterminism of the pre-stable system phase in this model, but at last ensure that the state $\neg\ done, stable, oracle, ldr$ represents a stable network and oracle, with a leader, that now allows election to proceed by the rules. This is done by giving an alternative refinement nondet$_5$ of nondet$_4$, which adds the guard $\neg\ stable \vee \neg\ oracle \vee \neg\ ldr$. The final refinement of election, election$_5$, simply observes that a single leader remains, and is thus elected (Fig. 17).

EVENT rule1 REFINES nondet
ANY
i, j
WHERE
$grd41 : done = FALSE$
$grd0 : i \in C$
$grd2 : j \in C$
$grd3 : i \neq j$
$grd4 : i \in l$
$grd5 : j \in l$
THEN
$act1 : l := l \setminus \{j\}$
$act2 : n := n \cup \{j\}$
$act3 : s(j) := N$
END

EVENT election$_5$ REFINES election$_4$
ANY
c, ns
WHERE
$grd40 : n = \{c\}$
$grd41 : l = C \setminus \{c\}$
$grd50 : done = FALSE$
$grd51 : oracle = TRUE$
$grd56 : s(c) = L$
THEN
$act2 : done := TRUE$
END

6 Conclusion

We have proposed a method integrating temporal and first-order logic, proof and tools for modelling and verification of liveness as well as safety properties, emphasising fairness-based reasoning. We thus integrate and exploit the best of two complementary technologies, Event-B/RODIN [4,6] and TLA [51]. We have performed Event-B developments for three example population protocols and fully discharged the usual first-order POs in the RODIN toolkit. We cannot directly prove—or even specify, for that matter—liveness and convergence properties for these protocols in the first-order formal language Event-B. Thus we have shown how we interpret the Event-B development in TLA, and then deploy standard and new proof rules in TLA to prove

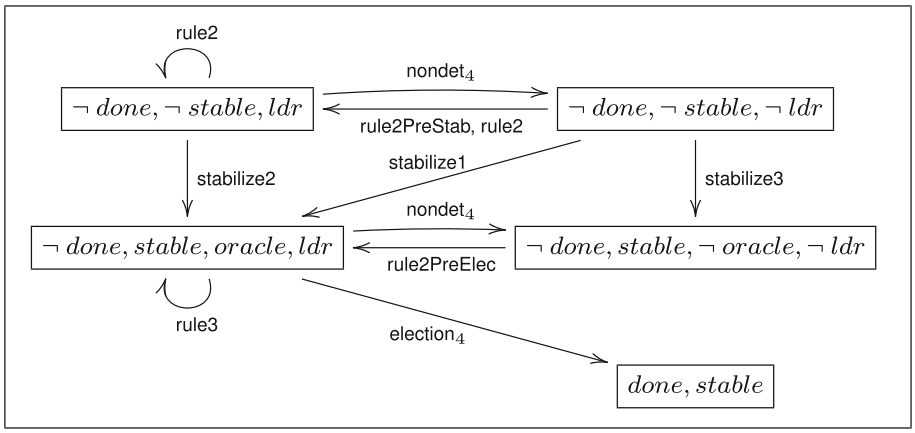


Fig. 16 SEL$_4$: introducing rules 2, 3

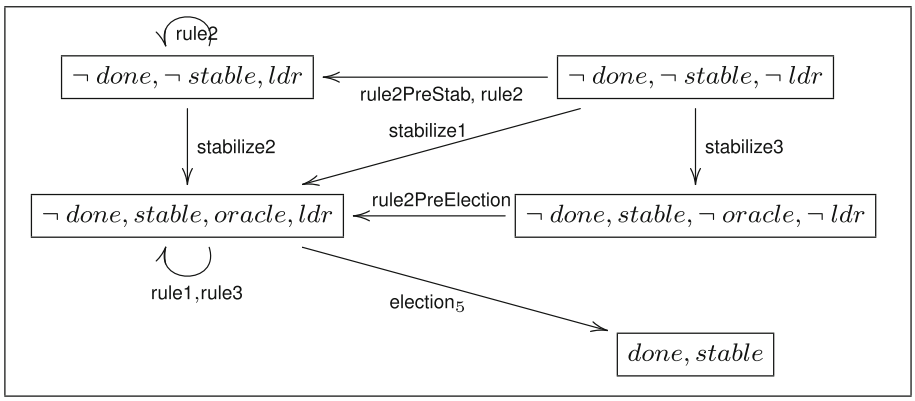


Fig. 17 SEL$_5$: introducing rule 1

liveness and its refinement. We believe the examples give sufficient patterns of reasoning for researchers to attempt the approach on their own examples; see comments below about plans for tool-supported proof.

We have observed repeating and reusable patterns in the liveness and liveness refinement proofs and have exploited this in defining new rules. We observe the close relationship between the liveness proofs for a model, and the LTS abstraction of that model, such as we see sketched in the leaders in Sect. 5. The LTS gives a diagrammatic abstraction of the algorithm, and we note related approaches. Butler's atomicity decomposition [28,36] is also a diagrammatic representation of the behaviour of the models in an Event-B development. Inspired by Jackson's JSD notation [46], these diagrams lay out sequences, selections and iterations of events at successive refinement levels. They are an aid to the modeller and are not related to proof. On the other hand, the fair objects of [40] proposed object-oriented encapsulation of, and reasoning about, assemblies of objects with fairness and liveness properties.

As future work we anticipate the possibility of extending our proof rules over certain classes of LTS models such as those in the leaders in Sect. 5. In particular our leaders model is a standard, simple one of being eventually always stabilised. A more challenging and realistic scenario would be an intermittently stabilised system where stable periods have some guaranteed lower time bound. This would open up quality-of-service reasoning for self-stabilising systems.

The set of temporal proof rules is semantically complete in the following sense. When a *leadsto* property is characterised using a trace semantics, it can be expressed equivalently using a predicate transformer semantics [38,55,61]. We can characterise the set of reachable states leading to a given assertion Q under fairness assumptions. This WP is defined using fixed-point characterisations and one can derive the minimal set of necessary rules. We have added rules that are often used when proving a liveness property, like confluence and transitivity.

We have mentioned that Event-B is not concerned with fairness or scheduling specification. Heuristics are employed with flags and counters to control the enablement and sequencing of events. This can be seen as constraining the choices of the (unspecified) scheduler. Event-B implicitly makes a fairness assumption on each event: if always, or infinitely often enabled, an event is assumed to fire infinitely often. To some extent, this is enforced in refinement by the VARIANT mechanism: every new event in a refinement step must reduce a natural- or set-valued variant expression. Thus the new event is eventually disabled until a refined event adjusts the variables constituting the variant expression. The

variant also thus provides an implicit LATTICE induction rule.

We believe an advantage of our approach is that making fairness and liveness first-class citizens in the method will make the design of scheduling more systematic and perhaps, through identification of patterns, automatable. We have seen that existing practice in state-based formal methods is to use heuristics and experience to manipulate event enablement in order to meet scheduling and liveness requirements. This is expensive in skills terms and does not make for easily verifiable solutions. A comment of J.-R. Abrial on the lights example is illustrative: "the ONLY important thing lies in the relationship between angel and iact. Once the angel has established a new connection between two (red) nodes then one must be sure that the controller (that is iact) has enough time to treat this new connection by removing one node. It seems to me that it is the key of this algorithm. In fact, it is a very frequent situation: the environment should not behave too quickly in comparison to the controller" [5]. This standard approach [29] to modelling control problems involves disabling a sensor event after reading environment data until the control action is completed. This engineering judgement—that the partial, approximate view the controller has of the environment is sufficiently accurate and safe—is left hidden in the development.

Our approach makes explicit the fairness assumptions and assertions at all refinement levels, so that in principle concrete scheduling design can be undertaken at a suitable level and verified to satisfy device fairness requirements. We believe this is particularly important for contemporary distributed systems such as MANET/WSN, a brave new world of low-energy and energy-harvesting computation [49]. Schedulability of processes/processors becomes more difficult, and explicit reasoning and design of scheduling to achieve liveness goals under fairness assumptions are required.

An immediate task is the automation of the paper-based temporal proofs by application of the emerging TLA proof tool TLAPS [34]. The bigger question is to what extent we can subcontract the first-order steps in a temporal proof to RODIN, and what trade-off cost that imposes on freedom of Event-B modelling. In particular, we will consider whether the more elaborate GF reasoning may be collapsible to FO, and tractable to RODIN.

Having demonstrated the utility of Event-B modelling with TLA reasoning for simple algorithms, next steps are to tackle (i) the extended population protocol models of Sect. 1 and even more challenging (ii) a real WSN/MANET algorithm. Two interesting application candidates are data aggregation [63] and localisation [66]. Aggregation is concerned with reducing data traffic either by averaging sensor data on a regional basis or by simply packing readings into larger messages. This includes notions of routing from data source to sink. In localisation, each node must dynamically identify neighbours to whom it is connected.

Appendix: Proof—$D = L \oplus O \oplus U \leadsto D = L \oplus U$—by GF1

Proving liveness properties requires to integrate fairness assumptions in the reasoning. We are presenting a proof of the following properties:

For any i and j satisfying $i + j = n$:

$$Spec \vdash (D = L \oplus 0^i \oplus 1^j) \leadsto (D = L \oplus 1^n).$$

The expression L means that there is at least one leader. We assume that the specification of the system is defined by $Spec \stackrel{def}{=} Init \wedge \Box[N]_h \wedge \mathsf{GFd}_h$. h is the list of state variables. N is the disjunction $\mathsf{DancingFU} \vee \mathsf{DancingLO} \vee \mathsf{DancingOU} \vee \mathsf{Dancing}$. n is the cardinality of $O \cup U$ where O and U are the initial sets of 0s and 1s.

Recall that GFd_h is the fairness assumption defined by:

$$\begin{aligned}\mathsf{GFd}_h \stackrel{def}{=}\ & \forall i, j \cdot i, j \in 1..n \wedge i + j = card(O \cup U) = n \\ & \Rightarrow GF_h(D = L \oplus 0^i \oplus 1^j, \mathsf{DancingLO}, \\ & \quad D = L \oplus 0^{i-1} \oplus 1^{j+1})\end{aligned}$$

It is clear that, without this assumption, the execution may loop in configurations avoiding the configuration without 0s. We consider that this property is stating the general fairness constraint of the population protocols.

PROOF: The proof is decomposed into steps.

1. $Spec \vdash (D = L \oplus 0^n) \leadsto (D = L \oplus 0^{n-1} \oplus 1)$
   PROOF:
   1.1. $Spec \vdash (D = L \oplus 0^n) \wedge [N]_h \Rightarrow (D = L \oplus 0^n) \vee (D = L \oplus 0^{n-1} \oplus 1)$
   PROOF:
   DancingFU and Dancing are not enabled, since F is empty. DancingLO may add a ONE and delete one ZERO. DancingOU is not enabled, since there is no 1. Only DancingLO is observed in the current configuration. □
   1.2. $Spec \vdash (D = L \oplus 0^n) \wedge \langle N \wedge \mathsf{DancingLO} \rangle_h \Rightarrow (D = L \oplus 0^{n-1} \oplus 1)$
   PROOF: DancingLO adds a ONE and deletes one ZERO. □
   1.3. $\Box[N]_h \wedge \mathsf{GFd}_h \Rightarrow ((D = L \oplus 0^n) \leadsto (D = L \oplus 0^n))$
   PROOF: Tautology □
   1.4. Q.E.D.
   PROOF: By GF1 rule, 1.1, 1.2 and 1.3, we conclude $Spec \vdash (D = L \oplus 0^n) \leadsto (D = L \oplus 0^{n-1} \oplus 1)$ □

2. ASSUME: $\forall a \in \{0..i\}.Spec \Rightarrow (D = L \oplus 0^{n-a}1^a) \rightsquigarrow (D = L \oplus 0^{n-a-1} \oplus 1^{a+1})$
   PROVE: $Spec \vdash (D = L \oplus 0^{n-(i+1)}1^{i+1}) \rightsquigarrow (D = L \oplus 0^{n-(i+2)} \oplus 1^{i+2})$
   PROOF: The proof applies the GF1 rule with the following predicates:
   - A is $\exists a.a \in \{0..i\} \wedge (D = L \oplus 0^{n-a}1^a)$
   - B is $(D = L \oplus 0^{n-(i+1)}1^{i+1})$
   - C is $(D = L \oplus 0^{n-(i+2)}1^{i+2})$

   2.1. $Spec \vdash (D = L \oplus 0^{n-(i+1)}1^{i+1}) \wedge [N]_h \Rightarrow (D = L \oplus 0^{n-(i+1)}1^{i+1}) \vee (D = L \oplus 0^{n-(i+2)} \oplus 1^{i+2}) \vee \big(\exists a.a \in \{0..i\} \wedge (D = L \oplus 0^{n-a}1^a)\big)$
   PROOF: DancingFU and Dancing are not enabled, since F is empty. DancingLO may add a ONE and delete one ZERO and DancingOU may add one ZERO and delete one ONE. □

   2.2. $Spec \vdash (D = L \oplus 0^{n-(i+1)}1^{i+1}) \wedge \langle N \wedge \mathsf{DancingLO}\rangle_h \Rightarrow (D = L \oplus 0^{n-(i+2)} \oplus 1^{i+2}$
   PROOF: DancingLO adds a ONE and deletes one ZERO. □

   2.3. $\Box[N]_h \wedge \mathrm{GFd}_h \Rightarrow (\exists a.a \in \{0..i\}(D = L \oplus 0^{n-a}1^a)) \rightsquigarrow D = L \oplus 0^{n-(i+1)}1^{i+1}))$
   PROOF:

   2.3.1. $\Box[N]_h \wedge \mathrm{GFd}_h \Rightarrow (D = L \oplus 0^{n-i}) \oplus 1^i \rightsquigarrow (D = L \oplus 0^{n-(i+1)}1^{i+1}))$
   PROOF: By assumption, we can choose $a = i$ and we obtain $Spec \vdash (D = L \oplus 0^{n-i}1^i) \rightsquigarrow (D = L \oplus 0^{n-(i+1)} \oplus 1^{i+1})$. □

   2.3.2. $\Box[N]_h \wedge \mathrm{GFd}_h \Rightarrow (D = L \oplus 0^{n-(i-1)}) \rightsquigarrow (D = L \oplus 0^{n-(i+1)}1^{i+1}))$
   PROOF: By assumption, we can choose $a = i - 1$ and we derive that $Spec \vdash (D = L \oplus 0^{n-(i-1)}1^{i-1}) \rightsquigarrow (D = L \oplus 0^{n-i}) \oplus 1^i)$ and, by step 2.3.1, $\Box[N]_h \wedge \mathrm{GFd}_h \Rightarrow (D = L \oplus 0^{n-(i-1)} \oplus 1^{i-1}) \rightsquigarrow (D = L \oplus 0^{n-(i+1)}1^{i+1}))$. The two properties are combined by the transitivity rule to obtain: $Spec \vdash (D = L \oplus 0^{n-(i-1)}1^{i-1}) \rightsquigarrow (D = L \oplus 0^{n-(i+1)}1^{i+1}))$. □

   2.3.3. $\forall a.a \in \{0..i\}.\Box[N]_h \wedge \mathrm{GFd}_h \Rightarrow (D = L \oplus 0^{n-a}1^a)) \rightsquigarrow D = L \oplus 0^{n-(i+1)}1^{i+1}))$
   PROOF: By steps 2.3.1, 2.3.2, the property for each a holds. □

   2.3.4. Q.E.D.
   PROOF: The $\rightsquigarrow$ operator is satisfying the rule of confluence: if $P_1 \rightsquigarrow Q$, $P_2 \rightsquigarrow Q$, …, $P_k \rightsquigarrow Q$, then $(\exists l.l \in 1..k \wedge P_l) \rightsquigarrow Q$, By step 2.3.3 and the confluence rule of $\rightsquigarrow$, we obtain that $\Box[N]_h \wedge \mathrm{GFd}_h \Rightarrow (\exists a.a \in \{0..i\}(D = L \oplus 0^{n-a}1^a)) \rightsquigarrow D = L \oplus 0^{n-(i+1)}1^{i+1}))$ □

   2.4. $Spec \Rightarrow GF((D = L \oplus 0^{n-(i+1)} \oplus 1^{i+1}), \mathsf{DancingLO}, (D = L \oplus 0^{n-(i+2)} \oplus 1^{i+2}))$
   PROOF: By definition of the specification of the model and especially GFd_h. □

   2.5. Q.E.D.
   PROOF: By GF1 rule with 2.1, 2.2,2.3,2.4, we conclude $Spec \vdash (D = L \oplus 0^{n-(i+1)}1^{i+1}) \rightsquigarrow (D = L \oplus 0^{n-(i+2)} \oplus 1^{i+2})$. □
   □

3. $\forall a \in 0..n-1.Spec \Rightarrow (D = L \oplus 0^{n-a}1^a) \rightsquigarrow (D = L \oplus 0^{n-(a+1)} \oplus 1^{a+1})$
   PROOF: By the two steps 1 and 2, we derive the property. □

4. $\forall a \in 0..n-1.Spec \Rightarrow (D = L \oplus 0^{n-a}1^a) \rightsquigarrow (D = L \oplus 1^n)$
   PROOF: By the step 3, and by the transitivity of $\rightsquigarrow$ properties, one derives that
   $Spec \vdash (D = L \oplus 0^{n-a}1^a) \rightsquigarrow (D = L \oplus 1^n)$ □

5. Q.E.D.
   PROOF: By 3, 4 and by the transitivity of the $\rightsquigarrow$, we derive that $\forall a \in 0..n-1.Spec \Rightarrow (D = L \oplus 0^{n-a} \oplus 1^a) \rightsquigarrow (D = L \oplus 0^{n-(a+1)} \oplus 1^{a+1})$. If we consider the property $(D = L \oplus 0^i \oplus 1^j$ for two values i and j satisfying $i + j = n$, one can apply for $a = j$, the previous property and obtain that $Spec \vdash (D = L \oplus 0^i \oplus 1^j) \rightsquigarrow (D = L \oplus 0^{i-1}) \oplus 1^{j+1})$ and if we reapply for $a = j + 1$, we obtain that $Spec \vdash (D = L \oplus 0^{i-1} \oplus 1^{j+1}) \rightsquigarrow (D = L \oplus 0^{i-2}) \oplus 1^{j+2})$ and after a finite number k of applications, we derive $Spec \vdash (D = L \oplus 0^{i-1} \oplus 1^{j+k-1}) \rightsquigarrow (D = L \oplus 0^{i-k}) \oplus 1^{j+k})$ such that $j + k = n$. By the transitivity rule of *leadsto*, we obtain that $Spec \vdash (D = L \oplus 0^i \oplus 1^j) \rightsquigarrow (x1^n)$. □

References

1. Abadi, M., Lamport, L.: The existence of refinement mappings. Theor. Comput. Sci. **82**(2), 253–284 (1991). doi:10.1016/0304-3975(91)90224-P
2. Abadi, M., Merz, S.: On TLA as a logic. In: Broy, M. (ed.) NATO ASI DPD, pp. 235–271 (1996)
3. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (1996)
4. Abrial, J.R.: Modeling in Event-B—System and Software Engineering. Cambridge University Press, Cambridge (2010)
5. Abrial, J.R.: Private communication (2013)
6. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in Event-B. Int. J. Softw. Tools Technol. Transf. (STTT) **12**(6), 447–466 (2010)
7. Abrial, J.R., Butler, M., Hallerstede, S., Voisin, L.: An open extensible tool environment for Event-B. In: Liu, Z., He, J. (eds.) Proceedings of ICFEM 2006, LNCS, vol. 4260. Macau (2006)
8. Abrial, J.R., Mussat, L.: Introducing dynamic constraints in B. In: Bert, D. (ed.) 2nd International B Conference, LNCS, vol. 1393, pp. 83–128. Springer, Montpellier (1998)

9. Abrial, J.R., Su, W., Zhu, H.: Formalizing hybrid systems with Event-B. In: ABZ, pp. 178–193 (2012)
10. Alpern, B., Schneider, F.B.: Defining liveness. Inf. Process. Lett. **21**(4), 181–185 (1985)
11. Alpern, B., Schneider, F.B.: Recognizing safety and liveness. Distrib. Comput. **2**(3), 117–126 (1987)
12. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distrib. Comput. **18**(4), 235–253 (2006)
13. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distrib. Comput. **20**(4), 279–304 (2007)
14. Angluin, D., Aspnes, J., Fischer, M.J., Jiang, H.: Self-stabilizing population protocols. TAAS **3**(4),103–117 (2008)
15. Apt, K.R., Olderog, E.R.: Proof rules and transformations dealing with fairness. Sci. Comput. Program. **3**(1), 65–100 (1983)
16. Araki, K., Galloway, A., Taguchi, K. (eds.): Integrated Formal Methods, Proceedings of the 1st International Conference on Integrated Formal Methods, IFM 99, New York, UK, 28–29 June 1999. Springer (1999)
17. Aspnes, J., Ruppert, E.: An introduction to population protocols. Bull. EATCS **93**, 98–117 (2007)
18. Back, R.J., Kurki-Suonio, R.: Distributed cooperation with action systems. ACM Trans. Program. Lang. Syst. **10**(4), 513–554 (1988)
19. Back, R.J., Kurki-Suonio, R.: Decentralization of process nets with centralized control. Distrib. Comput. **3**(2), 73–87 (1989)
20. Banach, R.: Pliant modalities in hybrid Event-B. In: iu, Z., Woodcock, J., Zhu, H. (eds.) Theories of Programming and Formal Methods, Lecture Notes in Computer Science, vol. 8051, pp. 37–53. Springer (2013)
21. Banach, R., Butler, M.: Cruise control in hybrid Event-B. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) ICTAC, Lecture Notes in Computer Science, vol. 8049, pp. 76–93. Springer (2013)
22. Banach, R., Zhu, H., Su, W., Wu, X.: Continuous behaviour in Event-B: A sketch. In: ABZ, pp. 349–352 (2012)
23. Beauquier, J., Blanchard, P., Burman, J.: Self-stabilizing leader election in population protocols over arbitrary communication graphs. In: Baldoni, R., Nisse, N., van Steen, M. (eds.) Principles of Distributed Systems—17th International Conference, OPODIS 2013, Nice, France, December 16–18, 2013. Proceedings, *Lecture Notes in Computer Science*, vol. 8304, pp. 38–52. Springer (2013)
24. Bjorner, D.: Software Engineering 1 Abstraction and Modelling. Springer, Texts in Theoretical Computer Science. An EATCS Series (2006). ISBN 978-3-540-21149-5
25. Bjorner, D.: Software Engineering 2 Specification of Systems and Languages. Springer, Texts in Theoretical Computer Science. An EATCS Series (2006). ISBN 978-3-540-21150-1
26. Bjorner, D.: Software Engineering 3 Domains, Requirements, and Software Design. Springer, Texts in Theoretical Computer Science. An EATCS Series (2006). ISBN 978-3-540-21151-8
27. Bjørner, D., Henson, M.C. (eds.): Logics of Specification Languages. EATCS Textbook in Computer Science. Springer, New York (2007)
28. Butler, M.: Decomposition structures for Event-B. In: Leuschel, M., Wehrheim, H. (eds.) Integrated Formal Methods, 7th International Conference, IFM 2009, Düsseldorf, Germany, February 16–19, 2009. Proceedings, Lecture Notes in Computer Science, vol. 5423, pp. 20–38. Springer (2009)
29. Butler, M.: Towards a cookbook for modelling and refinement of control problems. ECS, University of Southampton, Technical Report (2009)
30. Cai, S., Izumi, T., Wada, K.: How to prove impossibility under global fairness: on space complexity of self-stabilizing leader election on a population protocol model. Theory Comput. Syst. **50**(3), 433–445 (2012)
31. Cansell, D., Méry, D., Merz, S.: Predicate diagrams for the verification of reactive systems. In: Grieskamp, W., Santen, T., Stoddart, B. (eds.) IFM, Lecture Notes in Computer Science, vol. 1945, pp. 380–397. Springer (2000)
32. Chandy, K., Misra, J.: Parallel Program Design: A Foundation. Addison-Wesley, Reading (1988)
33. Chatzigiannakis, I., Michail, O., Spirakis, P.: Experimental verification and performance study of extremely large sized population protocols. In: Technical Report FRONTS-TR-2009-3 (2009). http://fronts.cti.gr/aigaion/?page=publication&kind=single&ID=61
34. Cousineau, D., Doligez, D., Lamport, L., Merz, S., Ricketts, D., Vanzetto, H.: TLA + proofs. In: Giannakopoulou, D., Méry, D. (eds.) FM, Lecture Notes in Computer Science, vol. 7436, pp. 147–154. Springer (2012)
35. Dijkstra, R.M.: Computation calculus bridging a formalization gap. Sci. Comput. Program. **37**(1–3), 3–36 (2000)
36. Fathabadi, A.S., Butler, M., Rezazadeh, A.: A systematic approach to atomicity decomposition in event-b. In: Eleftherakis, G., Hinchey, M., Holcombe, M. (eds.) Software Engineering and Formal Methods—10th International Conference, SEFM 2012, Thessaloniki, Greece, October 1–5, 2012. Proceedings, Lecture Notes in Computer Science, vol. 7504, pp. 78–93. Springer (2012). doi:10.1007/978-3-642-33826-7
37. Fischer, M.J., Jiang, H.: Self-stabilizing leader election in networks of finite-state anonymous agents. In: Shvartsman, A.A. (ed.) Principles of Distributed Systems, 10th International Conference, OPODIS 2006, Bordeaux, France, December 12–15, 2006, Proceedings, Lecture Notes in Computer Science, vol. 4305, pp. 395–409. Springer (2006)
38. Francez, N.: Fairness. Springer, New York (1986)
39. Gibson, J.P., Méry, D.: A unifying model for specification and design. In: Proceedings of the Workshop on Proof Theory of Concurrent Object-Oriented Programming (1996)
40. Gibson, P., Méry, D.: Fair objects. In: In OT98 COTSR, pp. 245–254 (1997)
41. Groslambert, J.: Verification of LTL on B event systems. In: Julliand, J., Kouchnarenko, O. (eds.) B 2007: Formal Specification and Development in B, 7th International Conference of B Users, Besançon, France, January 17–19, 2007, Proceedings, Lecture Notes in Computer Science, vol. 4355, pp. 109–124. Springer (2007)
42. Hallerstede, S.: On the purpose of Event-B proof obligations. In: E. Börger, M.J. Butler, J.P. Bowen, P. Boca (eds.) Abstract State Machines, B and Z, First International Conference, ABZ 2008, London, UK, September 16–18, 2008. Proceedings, Lecture Notes in Computer Science, vol. 5238, pp. 125–138. Springer (2008)
43. Hoang, T.S., Abrial, J.R.: Reasoning about liveness properties in Event-B. In: Qin, S., Qiu, Z. (eds.) ICFEM, Lecture Notes in Computer Science, vol. 6991, pp. 456–471. Springer (2011)
44. Hoare, C.: Communicating Sequential Processes. Prentice-Hall International, Upper Saddle River (1985)
45. Hudon, S., Hoang, T.S.: Systems design guided by progress concerns. In: IFM, pp. 16–30 (2013)
46. Jackson, M.: System Development. Prentice-Hall, Englewood Cliffs (1983)
47. Järvinen, H.M., Kurki-Suonio, R.: Disco specification language: marriage of actions and objects. In: ICDCS, pp. 142–151. IEEE Computer Society (1991)
48. Jones, C.B.: Specification and design of (parallel) programs. In: IFIP Congress, pp. 321–332 (1983)
49. Kansal, A., Hsu, J., Zahedi, S., Srivastava, M.B.: Power management in energy harvesting sensor networks. ACM Trans. Embed. Comput. Syst. (2007). doi:10.1145/1274858.1274870
50. Lamport, L.: A simple approach to specifying concurrent systems. Commun. ACM **32**(1), 32–45 (1989)

51. Lamport, L.: The temporal logic of actions. ACM Trans. Program. Lang. Syst. **16**(3), 872–923 (1994)
52. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley, Boston (2002)
53. Lamport, L., Matthews, J., Tuttle, M.R., Yu, Y.: Specifying and verifying systems with TLA+. In: Muller, G., Jul, E. (eds.) ACM SIGOPS European Workshop, pp. 45–48. ACM (2002)
54. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems—Safety. Springer, New York (1995)
55. Méry, D.: A proof system to derive eventually properties under justice hypothesis. In: Gruska, J., Rovan, v, Wiedermann, J. (eds.) Mathematical Foundations of Computer Science 1986, Bratislava, Czechoslovakia, August 25–29, 1996, Proceedings, Lecture Notes in Computer Science, vol. 233, pp. 536–544. Springer (1986). doi:10.1007/BFb0016280
56. Méry, D.: Requirements for a temporal B : Assigning Temporal Meaning to Abstract Machines ... and to Abstract Systems. In: Galloway, A., Taguchi, K. (eds.) IFM'99 Integrated Formal Methods 1999, Workshop on Computing Science, New York (1999)
57. Méry, D.: Refinement-based guidelines for algorithmic systems. Int. J. Softw. Inf. **3**(2–3), 197–239 (2009)
58. Méry, D., Poppleton, M.: Formal modelling and verification of population protocols. In: Integrated Formal Methods, 10th International Conference, IFM 2013, Turku, Finland, June 10–14, 2013. Proceedings, pp. 208–222 (2013). doi:10.1007/978-3-642-38613-8_15
59. Olderog, E.R., Apt, K.R.: Fairness in parallel programs: the transformational approach. ACM Trans. Program. Lang. Syst. **10**(3), 420–455 (1988)
60. Owicki, S.S., Lamport, L.: Proving liveness properties of concurrent programs. ACM Trans. Program. Lang. Syst. **4**(3), 455–495 (1982)
61. Park, D.: A predicate transformer for weak fair iteration. In: Proceedings of 6th IBM Symposium on Mathematical Foundations in Computer Science, pp. 257–275. IBM, Hakone (1981)
62. Picco, G.: Software engineering and wireless sensor networks: happy marriage or consensual divorce. In: FoSER 2010 (2010)
63. Rajagopalan, R., Varshney, P.K.: Data-aggregation techniques in sensor networks: a survey. IEEE Commun. Surv. Tutor. **8**(1–4), 48–63 (2006)
64. Schneider, S.A., Treharne, H., Wehrheim, H., Williams, D.M.: Managing LTL properties in event-b refinement. In: Albert, E., Sekerinski, E. (eds.) Integrated Formal Methods—11th International Conference, IFM 2014, Bertinoro, Italy, September 9–11, 2014, Proceedings, Lecture Notes in Computer Science, vol. 8739, pp. 221–237. Springer (2014). doi:10.1007/978-3-319-10181-1
65. Spirakis, P.: Population protocols and related models. In: Theoretical Aspects of Distributed Computing in Sensor Networks Monographs in Theoretical Computer Science. An EATCS Series 2011, pp. 109–159. Springer (2011)
66. Stavvides, A., Srivastava, M., Girod, L., Estrin, D.: Wireless Sensor Networks. Springer, New York (2004)
67. Yick, J., Mukherjee, B., Ghosal, D.: Wireless sensor network survey. Comput. Netw. **52**(12), 2292–2330 (2008). doi:10.1016/j.comnet.2008.04.002

Dominique Méry is professor of computing science at Université de Lorraine since the first of September 1993 and he is teaching in the School of Computing Science, Telecom Nancy. He is Head of the Research Group (Formal Methods and Applications) at the LORIA Laboratory and Head of the PhD School IAEM Lorraine. He us a former Member of l' Institut Universitaire de France [1995–2000]; he got a grant for Scientific Excellence Grad A (2009–2013 and 2013–2017)); he is member of IFIP WG 1.3 Foundations of System Specification. His scientific interests are formal methods and applications to real applications: correct-by-construction approach, refinement, modelling.

Michael Poppleton received his B.Sc. in Mathematics and Mathematical Statistics from the University of the Witwatersrand in Johannesburg, South Africa in 1977. He then pursued a 13-year career in the UK in system and data communications development and testing, including time with Apricot Computers and on the Inland Revenue Computerization of PAYE project. In 1991 he completed an M.Sc. in Software Engineering from Aston University, Birmingham, and in 2001 he completed his part-time Ph.D under supervision of Richard Banach at the University of Manchester. His topic "Formal Methods for Continuous Systems" proposed the notion of retrenchment, a liberalization of the refinement method central to state-based Formal Methods. Mostly at the University of Southampton, he then investigated application and development of retrenchment. Joining the RODIN and DEPLOY EU IST projects with Prof. Michael Butler, he used the Event-B language to investigate composition, decomposition, feature-based formal development, and integration with UML. He then developed an interest in formal and co-simulation modelling of Wireless Sensor Networks. Inspired by this application, with Prof. Dominique Méry of LORIA, Nancy he has more recently considered the integration of temporal logic with the first-order Event-B for integrated safety/liveness reasoning in distributed systems.