MoTeL: A Modal Temporal Logic Proofmode in Rocq

Jonas Kastberg Hinrichsen
Aalborg University
Copenhagen, Denmark
jkhi@cs.aau.dk

Elli Anastasiadi
Aalborg University
Aalborg, Denmark
ellia@cs.aau.dk

Abstract

Temporal Logic is a fundamental design mechanism and formal verification tool of temporal properties, such as eventual delivery and termination. The metatheory of various temporal logics have been mechanised in proof assistants, but limited focus have been put on the interactive verification of examples, resulting in verbose and time-consuming proofs. To address this concern, we develop MoTeL; an embedded proofmode in Rocq for tractable mechanisation of temporal properties. To do so we first define a general-purpose higher-order temporal logic tailored for expressing linear time temporal properties, subsuming conventional linear temporal logic and the linear time μ-calculus. We then develop an embedded proofmode for the logic in Rocq, through an instantiation of the MoSel modal separation logic proofmode infrastructure. The key insight is that temporal logic is based on temporal modalities that share the same modal properties as modalities already supported by MoSeL. As a result, we can reuse empirically validated modality-based reasoning principles. We motivate MoTeL by mechanising examples on fair progress, parity automata, and Petri nets.

1 Introduction

Formal verification of concurrent and distributed systems is imperative for establishing trust in digital infrastructure. A popular approach to formal verification in general is mechanisation via proof assistants [38, 39, 41], that validate proofs against a small trusted computing base (TCB). Proof assistants are highly expressive general-purpose tools, typically supporting higher-order logic, infinite state-spaces, and verification of undecidable properties. As a result, mechanisation is increasingly becoming a verification community standard.

The primary caveat of proof assistants is that proofs are done interactively, where even minute details must be explicitly accounted for. This is especially true for concurrent and distributed systems, where properties are often *temporal*; asserting not only whether a property is true but also *when* it is true. Verification of temporal properties is often based on traces of execution, that must be manually managed throughout an interactive proof. Modal temporal logic ease this burden, by abstracting over the notion of time, letting logical modalities determine when something holds true, e.g. that a proposition P holds true at the *next step* $\circ P$, *always* $\Box P$, or *eventually* $\Diamond P$. These modalities come with modal laws, facilitating well-behaved reasoning principles.

Existing proof assistant embeddings of modal temporal logic [10, 13, 31, 40, 44, 46] do not leverage these reasoning principles for interactive verification of examples. They generally focus on either (1) mechanising the metatheory, or (2) exporting concrete examples to external solvers. Both of these are unsatisfactory for the general-purpose verification of concrete examples. Approach (1) treats temporal logic as an embedded logic, which does not readily integrate with the verification infrastructure of the proof assistant, as we will elaborate on momentarily. Approach (2) extends the TCB to any unverified translation and external tool, and is less expressive than proof assistants, often limited to finite or structurally recursive state spaces. We elaborate further on prior developments in §6.

This paper addresses the facilitation of modality-based interactive verification for temporal properties, through the development of an embedded proofmode for modal temporal logic in the Rocq proof assistant.

The challenges of working with temporal logic in a proof assistant. Proof assistants provide proofmode infrastructure for working with the proof state at high level. In the Rocq proof assistant, a statement such as `forall (P Q : Prop), (P -> Q) /\ P -> Q`, will yield the proof state:

```
P, Q : Prop
HPQ: (P -> Q)
HP : P
============================
Q
```

Notably, all terms and hypotheses (above the line) are named, and can be individually manipulated.

Embedded temporal logics typically define the propositions of the logic as meta-logic propositions indexed by some world (denoting time) $\mathtt{tProp} \triangleq W \to \mathtt{Prop}$. The entailment of the logic internalise the world, e.g. via universal quantification the world under consideration $P \vdash Q \triangleq \forall w.\, P\, w \to Q\, w$. In Rocq, a statement such as `forall (P Q : W -> Prop), (P -> Q) /\ P |- Q` will thus result in:

```
P, Q : W -> Prop
============================
(P -> Q) /\ P |- Q
```

Here, we cannot introduce the temporal logic hypotheses into the named Rocq context. A user must thus manually commute and associate hypotheses to match the shape of applied lemmas. This manual effort is exacerbated when we

have to consider modalities that change the underlying world. An alternative is to unfold the temporal logic entailment:

```
P, Q : W -> Prop
w : W
HPQ : (P -> Q) w
HP : P w
============================
Q w
```

We can now use Rocq's proofmode, but have lost the benefits of working with temporal logic, and must manually thread the current world between hypothesis whenever it changes.

Embedded proofmode as a solution. Krebbers et al. [25, 26] observed the above problem for Iris [21]—a separation logic in Rocq—and developed MoSeL an embedded proofmode for separation logic in Rocq. MoSeL gives names to separation logic level hypotheses, in a separate separation logic proof context, while preserving the abstraction:

```
P, Q : W -> Prop
============================
HPQ : (P -> Q)
HP : P
----------------------------*
Q
```

MoSeL additionally has tactics for modality-based reasoning, automatically keeping the proof context consistent when the world changes. MoSeL has been used in mechanisations of more than 150 papers in the past 10 years [37], indicative of its helpfulness for interactive verification.

To make interactive verification of temporal properties more tractable, this paper develops MoTeL, an embedded modal temporal logic proofmode in Rocq, for a general-purpose higher-order linear time μ-calculus logic. MoTeL is an instantiation of MoSeL, simultaneously evidencing MoSeL's general applicability for developing new embedded proofmodes for modal logics other than separation logic.

Overview of MoTeL. Our temporal logic is parameterised by possibly-finite traces based on an arbitrary relations, between unrestricted state-spaces indexed by unrestricted labels. The logic is based on the temporal operators *now* ($\downarrow \phi$), and *next* ($\circ P$), corresponding to head and tail functions over the trace. Unlike traditional linear temporal logic (LTL), $\downarrow \phi$ assert unrestricted Rocq predicates over the state-label pair, e.g. permitting the definition of enabledness of labels, by referring to the relation over which the trace itself is defined. The logic supports conventional propositional logic connectives, and—by Rocq convention—higher-order quantification. It additionally support Knaster-Tarski monotonic least and greatest fixpoints. To express temporal properties, the logic uses derived connectives in terms of quantifiers and fixed points. The LTL *always* ($\Box$), *until* ($\cup$), and *eventually* ($\Diamond$) connectives can be defined as:

$$\Box P \triangleq \forall n.\ \circ^n P$$
$$P \cup Q \triangleq \mu X.\ Q \vee (P \wedge \circ\ X)$$
$$\Diamond P \triangleq \mathsf{True} \cup P$$

By Rocq convention our logic is constructive, with an optional classical extension. We cover the logic further in §2.

To give an idea of how MoTeL works, consider the following temporal assertion (with named hypotheses for clarity):

$$HPQ : \Box(P \to \circ \Diamond Q), HQR : \Box(Q \to \circ \Diamond R), HP : \Diamond P \vdash \circ \circ \Diamond R$$

It effectively captures transitivity across various temporal modalities. The typical temporal logic approach here is to

1. eliminate HP : $\Diamond P$ to get HP: P,
2. apply HPQ to HP to get HQ : $\circ \Diamond Q$,
3. introduce $\circ$ to get HQ : $\Diamond Q$,
4. eliminate HQ : $\Diamond Q$ to get HQ: Q,
5. apply HQR to HQ to get HR : $\circ \Diamond R$

while preserving the global hypotheses and goal at every step. An interactive proof of the above assertion using a temporal logic in Rocq looks like the following:

```
Lemma introduction_example_no_pm P Q R :
  □ (P ⟶ ○ ◊ Q) ∧ □ (Q ⟶ ○ ◊ R) ∧ ◊ P ⊢ ○ ○ ◊ R.
Proof.
  (* 1. eliminate HP : ◊ P *)
  rewrite -{2}(ltl_eventually_idemp R);
  rewrite -(ltl_eventually_next_comm (◊ R));
  rewrite -(ltl_eventually_next_comm (○ ◊ R));
  rewrite (ltl_always_idemp (P ⟶ ○ ◊ Q));
  rewrite (ltl_always_idemp (Q ⟶ ○ ◊ R));
  rewrite ltl_always_eventually_and_l;
  rewrite ltl_always_eventually_and_l;
  apply ltl_eventually_mono.
  (* 2. apply HPQ to HP *)
  rewrite bi.and_comm -bi.and_assoc
    (ltl_always_elim (P ⟶ ○ ◊ Q)) bi.impl_elim_r.
  (* 3. introduce ○ to get HQ *)
  rewrite ltl_always_next ltl_next_and;
    apply ltl_next_mono.
  (* 4. eliminate HQ : ◊ Q *)
  rewrite -{2}(ltl_eventually_idemp R);
  rewrite -(ltl_eventually_next_comm (◊ R));
  rewrite ltl_always_eventually_and_l;
  apply ltl_eventually_mono.
  (* 5. apply HQR to HQ *)
  by rewrite bi.impl_elim_l.
Qed.
```

The verbosity of the proof comes from, (1) managing the proof context (e.g. commuting conjunctions), and (2) preserving proof context/goal across worlds.

MoTeL handles both of these in a similar fashion to MoSeL; (1) using a proof context for the embedded logic, and (2) providing tactics for working with modalities. This lets us write a proof similar to the above as follows:

```
Lemma introduction_example (P Q R : tProp) :
  □ (P ⟶ ○ ◊ Q) ∧ □ (Q ⟶ ○ ◊ R) ∧ ◊ P ⊢ ○ ○ ◊ R.
Proof.
```

```
  (* 0. introduce hypotheses *)
  iIntros "(#HPQ&#HQR&HP)".
  (* 1. eliminate HP : ◊ P *)
  iMod "HP".
  (* 2. apply HPQ to HP *)
  iDestruct ("HPQ" with "HP") as "HQ".
  (* 3. introduce ∘ to get HQ *)
  iModIntro.
  (* 4. eliminate HQ : ◊ Q *)
  iMod "HQ".
  (* 5. apply HQR to HQ *)
  by iApply "HQR".
Qed.
```

Besides the proof being shorter, a benefit of a custom proofmode is that the proof becomes more *declarative*. Each step of the proof corresponds to one tactic, that describes what we want to achieve, rather than how we want to achieve it. We will elaborate on how the proofmode works in §3.

To motivate MoTeL we mechanised a suite of examples, including but not limited to reachability under fair scheduling, and fair delivery in an unreliable network. We additionally used our expressive approach to prove properties about Petri nets and parity automata.

Contributions. The contributions made by the paper can be summarised as follows

- Mechanisation of a linear time μ-calculus in Rocq §2
- An embedded proofmode for the logic in Rocq, with support for proof context and modalities §3
- A suite of case studies, including the Stenning Protocol, parity automata, and Petri nets §4
- Demonstrating the versatility of MoSeL through our non-separation logic instantiation §5.

All of our results have been mechanised in Rocq.

2 The Modal Temporal Logic

We now cover the semantics and rules of our temporal logic. In particular, we cover the definition of our atypical possibly-infinite traces §2.1, the top-level logical entailment §2.2 propositional connectives §2.3, the temporal connectives §2.4, support for least and greatest fixpoints §2.5, and projections of point-level propositions §2.6. We additionally show how to derive model-level axioms from an input model §2.7, and extract model-level properties from a proof in the logic §2.8. We conclude with an example of how the logic can be instantiated and used to prove a reachability property for a model of fair scheduling §2.9.

2.1 Possibly-Finite Trace Definition

To jointly support both non-terminating and terminating systems, our logic considers possibly-finite traces, with an explicit absorbing post-terminated state $\bot$.[1] We define our traces coinductively as follows:

$$\mathsf{PreTrace\ S\ L} ::= \mathsf{trsingl}\ s \mid \mathsf{trcons}\ s\ l\ tr$$

$$\mathsf{Trace\ S\ L} \triangleq (\mathsf{PreTrace\ S\ L})^?$$

Where, $A^?$ ::= **some** a | **none**. We use the syntax $\langle s \rangle \triangleq$ **some**(trsingl s), $s \xrightarrow{l} tr \triangleq$ **some**(trcons $s\ l\ tr$), and $\bot \triangleq$ **none** for the inhabitants of Trace. We define Trace in two steps to rule out the invalid construction $s \xrightarrow{l} \bot$.

We define head (hd) and tail (tl) functions for the trace.

$$\mathsf{hd} : \mathsf{Trace\ S\ L} \to (\mathsf{S} * \mathsf{L}^?)^? \qquad \mathsf{tl} : \mathsf{Trace\ S\ L} \to \mathsf{Trace\ S\ L}$$

$$\mathsf{hd}\ \bot \triangleq \textbf{none} \qquad \mathsf{tl}\ \bot \triangleq \bot$$

$$\mathsf{hd}\ \langle s \rangle \triangleq \textbf{some}(s, \textbf{none}) \qquad \mathsf{tl}\ \langle s \rangle \triangleq \bot$$

$$\mathsf{hd}\ (s \xrightarrow{l} tr) \triangleq \textbf{some}(s, \textbf{some}(l)) \qquad \mathsf{tl}\ (s \xrightarrow{l} tr) \triangleq \textbf{some}\ tr$$

In the logic, we restrict ourselves to maximal traces w.r.t. a stepping relation R, via the dependent pair:

$$\mathsf{WFTrace\ S\ L\ R} \triangleq \{tr : \mathsf{Trace\ S\ L};\ wf_tr : \mathsf{well_formed\ R}\ tr\}$$

Here well_formed is coinductively defined as:

$$\frac{}{\mathsf{well_formed\ R}\ \bot} \qquad \frac{\neg\ \mathsf{reducible}\ s}{\mathsf{well_formed\ R}\ \langle s \rangle}$$

$$\frac{\mathsf{hd}\ (\textbf{some}\ tr) = \textbf{some}\ (s', _) \quad \mathsf{R}\ s\ l\ s' \quad \mathsf{well_formed\ R}\ (\textbf{some}\ tr')}{\mathsf{well_formed\ R}\ (s \xrightarrow{l} tr)}$$

Where $\mathsf{reducible}\ s \triangleq \exists l, s'.\ \mathsf{R}\ s\ l\ s'$. We omit the type and relation arguments whenever they are implied from context.

2.2 Entailment and Adequacy Theorem

Our temporal propositions are predicates over well-formed traces $\mathsf{tProp\ S\ L\ R} \triangleq \mathsf{WFTrace\ S\ L\ R} \to \mathsf{Prop}$. We define entailment in the usual way for linear-time temporal logic:

$$P \vdash Q \triangleq \forall tr.\ P\ tr \Rightarrow Q\ tr$$

This yields the following adequacy theorem by definition.

Theorem 2.1. *(Adequacy) If $P \vdash Q$ holds in the temporal logic, then $\forall tr.\ P\ tr \Rightarrow Q\ tr$ holds in the meta logic.*

Meta-level properties about traces can thus be derived from proven assertions in the logic.

2.3 Propositional Connectives

We define all the conventional propositional logic connectives via point-wise lifting on the traces:

$$P \wedge Q \triangleq \lambda tr.\ P tr \wedge Q tr \qquad \forall x.\ P \triangleq \lambda tr.\ \forall x.\ P tr$$

$$P \vee Q \triangleq \lambda tr.\ P tr \vee Q tr \qquad \exists x.\ P \triangleq \lambda tr.\ \exists x.\ P tr$$

$$P \to Q \triangleq \lambda tr.\ P tr \to Q tr \qquad p \triangleq \lambda_.\ p$$

For quantifiers, we reuse the Rocq meta-logic quantifiers, meaning our quantifiers are impredicative and higher-order, for example allowing quantification over tProp itself. The

[1] The absorbing state is a built-in approach for handling the common requirement that the examined model (e.g., Kripke structure) has always at least one enabled transition per state [7, Chapter 3]. Further clarification on the use of the absorbing state can be found in the appendix §A.

standard propositional logic rules follow from the point-wise definitions. Meta-level propositions p are implicitly coerced the logic, and transport freely $(\vdash p) \Leftrightarrow p$

2.4 Temporal Connectives

We define core temporal logical connectives Now ($\downarrow \phi$) and Next ($\circ P$) via their corresponding trace operations:

$$\downarrow \phi \triangleq \lambda tr.\ \phi\ (\mathsf{hd}\ tr)$$
$$\circ P \triangleq \lambda tr.\ P\ (\mathsf{tl}\ tr)$$

Our $\downarrow \phi$ assertion supersede conventional LTL atomic propositions, as they are unrestricted Rocq predicates on the state-label-pair returned by hd. This lets us define expressive assertions, such as strong fairness, as demonstrated in §2.9. The definitions yield a conventional set of axioms, seen in Fig. 1. We can derive the conventional globally modality ($\Box P$) in terms of universal quantification and $\circ$:

$$\Box P \triangleq \forall n.\ \circ^n P$$

We additionally derive the conventional modalities until ($P \cup Q$) and eventually ($\Diamond P$) via internal fixpoints as explained below. We support the conventional laws of introduction, elimination, induction, and co-induction of each modality as expected, derived using the axioms of $\circ$. A more complete overview of the logic can be found in the appendix §B

2.5 Least and Greatest Fixpoints

Our logic supports internal least and greatest fixpoints, for any functor F, where $\forall P, Q.\ \Box(P \to Q) \to (F\,P) \to (F\,Q)$. This is the standard monotonicity requirement also present in fixed-point calculi. With these we can give definitions of Until ($P \cup Q$) and Eventually ($\Diamond P$).

$$P \cup Q \triangleq \mu X.\ Q \vee (P \wedge \circ X)$$
$$\Diamond P \triangleq \mathsf{True} \cup P$$

The definition of Until satisfies the criteria as:

$$\Box(X \to Y) \to Q \vee (P \wedge \circ X) \to Q \vee (P \wedge \circ Y)$$

The fixpoints come with fixpoint theorems:

$$\text{(least)} \quad \Box(F\,(P \wedge \mu F) \to P) \vdash (\mu F \to P)$$
$$\text{(greatest)} \quad \Box(P \to F\,(P \vee \nu F)) \vdash (P \to \nu F)$$

The canonical fixpoint theorem of Until follows directly from the least fixpoint theorem:

$$\Box((Q \vee (P \wedge \circ(P \cup Q) \wedge \circ R)) \to R) \vdash ((P \cup Q) \to R)$$

Similar to the μ-calculus, our fixpoints let us define temporal connectives beyond Until and Eventually, such as a proposition holding every other step:

$$\widehat{\Box}\,P \triangleq \nu X.\ P \wedge \circ\circ X$$

The definition of $\widehat{\Box}$ satisfies the criteria as:

$$\Box(X \to Y) \to (P \wedge \circ\circ X) \to (P \wedge \circ\circ Y)$$

It has the following fixpoint theorem:

$$\Box(P \to \circ\circ(P \vee \widehat{\Box}\,Q)) \vdash P \to \widehat{\Box}\,Q$$

Note that such use of fixpoints does not increase our expressive power beyond classical monadic second-order logic (MSO) over traces [5, 14]. Even so, fixpoints allows a small core of basic modalities, whilst using MSO instead requires heavier reliance on quantifiers.

2.6 Projections

Our state-label-space is more complex than a simple finite set, and often looks like a tuple of typed variables. Our Now ($\downarrow \phi$) construction assert ϕ over the whole state-label-space, but we often want to only assert predicates about *parts* of a state or label. To more succinctly express such assertions we add projection variants of the $\downarrow \phi$ assertion:

$$\downarrow^f \phi \triangleq \downarrow (\mathsf{from_option}\ \phi\ \mathsf{False} \circ \mathsf{map}\ f)$$
$$\downarrow^f_s \phi \triangleq \downarrow^{(\mathsf{map}\ f \circ \mathsf{map\ fst})} \phi$$
$$\downarrow^f_l \phi \triangleq \downarrow^{(\mathsf{map}\ f \circ \mathsf{flat_map\ snd})} \phi$$

More precisely, $\downarrow^f \phi$ asserts that ϕ holds for projection $f : (\mathsf{S} * \mathsf{L}^?)^? \to A^?$ of the current head, ruling out incomplete projections **none** via False. Subsequently, $\downarrow^f_s \phi$ and $\downarrow^f_l \phi$ projects the state (resp. label) of the current head. Finally, we often omit f to denote the identity projection (id), and write x in place of ϕ, to denote equality with x ($\mathsf{eq}\ x$). We can then write e.g. $\downarrow_s 42$ to denote that the current state is 42.

2.7 Model-Specific Axioms

To prove trace properties about specific models we must first prove the axioms of a given model, often in terms of the individual steps a model can take, such as $\downarrow_s i \vdash \downarrow_s i + 1$. For this, we have the following lemmas that reflect the stepping relation R into the logic:

$$\textsc{step}\ \frac{\mathsf{reducible}\ s}{\downarrow_s s \vdash \exists l, s'.\ R\,s\,l\,s' \wedge \downarrow_l l \wedge \circ\downarrow_s s'}$$

$$\textsc{step-label}\ \frac{}{\downarrow_s s \wedge \downarrow_l l \vdash \exists s'.\ R\,s\,l\,s' \wedge \circ\downarrow_s s'}$$

As previously, reducible s ($\triangleq \exists l, s'.\ R\,s\,l\,s'$) is the obligation to show that s is not a terminated state (and can thus reduce).

2.8 Extraction

Applying the adequacy theorem Theorem 2.1 yield trace predicates at the meta-level, expressed in terms of temporal logic connectives. To extract a true meta-level property, we use the following equivalences:

$$\Box P \equiv \forall n.\ \circ^n P$$
$$\Diamond P \equiv \exists n.\ \circ^n P$$
$$\circ P\ tr \equiv P\ (\mathsf{tl}\ tr)$$
$$\downarrow^f P\ tr \equiv P\ (f\ (\mathsf{hd}\ tr))$$
$$\downarrow^f_s P\ tr \equiv \exists s.\ \mathsf{hd}\ tr = \mathbf{some}\ (s, _) \wedge P\ (f\ s)$$
$$\downarrow^f_l P\ tr \equiv \exists l.\ \mathsf{hd}\ tr = \mathbf{some}\ (_, \mathbf{some}\ l) \wedge P\ (f\ l)$$

Syntax:

$$\begin{array}{lll} P, Q \in \mathsf{tProp} & ::= \mathsf{True} \mid \mathsf{False} \mid P \wedge Q \mid P \vee Q & (\text{Propositional logic}) \\ & \mid \forall x.\, P \mid \exists x.\, P \mid x = y & (\text{Higher-order logic with equality}) \\ & \mid \downarrow \phi \mid \circ P & (\text{Temporal logic}) \end{array}$$

Now (↓) Axioms

$$\frac{\forall osl.\, \phi\ osl \Rrightarrow \psi\ osl}{\downarrow \phi \vdash \downarrow \psi}\ \downarrow\text{-MONO} \qquad \forall x.\, \downarrow \phi \dashv\vdash \downarrow(\widehat{\forall}\, x.\, \phi)\ \downarrow\!-\forall \qquad \exists x.\, \downarrow \phi \dashv\vdash \downarrow(\widehat{\exists}\, x.\, \phi)\ \downarrow\!-\exists$$

$$\neg \downarrow \phi \vdash \downarrow \widehat{\neg}\, \phi\ \downarrow\text{-NOT} \qquad \downarrow \phi \vdash \exists osl.\, \phi\ osl\ \downarrow\text{-PURE}$$

Next (○) Axioms:

$$\frac{\vdash P}{\vdash \circ P}\ \circ\text{-TAUT} \qquad \circ(P \to Q) \vdash \circ P \to \circ Q\ \circ\text{-MONO} \qquad \circ \neg P \dashv\vdash \neg \circ P\ \circ\text{-NOT}$$

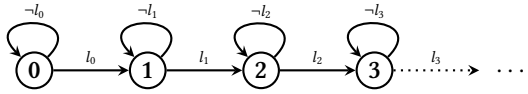
$$\circ \exists x.\, P \vdash \exists x.\, \circ P\ \circ\!-\exists\text{-COMM-R} \qquad \forall x.\, \circ P \vdash \circ \forall x.\, P\ \circ\!-\forall\text{-COMM-R} \qquad \circ p \dashv\vdash p\ \circ\text{-PURE}$$

Figure 1. Foundation of the Temporal Logic. We use $\widehat{_}$ as pointwise lifting.

Note that we omit $P \cup Q$, as (1) it requires complex iteration, and (2) is rarely used in top-level theorems. A property such as $\downarrow_s i \vdash \Diamond \downarrow_s (i+1)$ will thus yield the meta-level assertion $\forall tr.\, \mathsf{hd}\ tr = \mathbf{some}(i, _) \Rightarrow \exists n.\, \mathsf{hd}\ (\mathsf{tl}^n\ tr) = \mathbf{some}(i+1, _)$.

2.9 Example: Progress Under Fair Scheduling

Consider a model of progress under fair scheduling between some set of threads. The states of the model are natural numbers signifying progress, while labels determine the scheduled thread. Each state i is associated with a single thread that can make progress denoted l_i.



We write $\neg l$ for any label $l' \neq l$. The model is formally represented as follows:

$$\mathsf{S} := \mathbb{N} \qquad \mathsf{L} : \mathsf{Type} \qquad f : \mathsf{S} \to \mathsf{L}$$
$$\mathsf{R} := \mathsf{R}\ i\ (f\ i)\ (i+1) \mid \mathsf{R}\ i\ (\neg f\ i)\ i$$

We write $l_i := f\ i$. We can reflect the relation into the logic via STEP-LABEL, to get the model axioms:

$$\downarrow_s i \wedge \downarrow_l l_i \vdash \downarrow_s (i+1) \qquad \downarrow_s i \wedge \downarrow_l \neg l_i \vdash \downarrow_s i$$

The property that we want to prove is:

$$\downarrow_s 0 \vdash \Diamond \downarrow_s n$$

That is, we eventually reach any natural number n. The property only holds under some assumption of *fairness*, as a possible trace is an infinite sequence of $\neg l_0$:

$$0 \xrightarrow{\neg l_0} 0 \xrightarrow{\neg l_0} 0 \xrightarrow{\neg l_0} \ldots$$

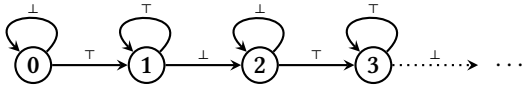
We define the conventional notion of strong fairness as:

$$\begin{aligned} \mathsf{enabled_meta}\ l\ s &\triangleq \exists s'.\, R\ s\ l\ s' \\ \mathsf{enabled}\ l &\triangleq \downarrow_s (\mathsf{enabled_meta}\ l) \\ \mathsf{fair} &\triangleq \forall l.\, \Box(\Box \Diamond \mathsf{enabled}\ l \to \Diamond \downarrow_l l) \end{aligned}$$

A label which is enabled (can reduce in the current state) infinitely often ($\Box \Diamond$), will eventually be scheduled. As the labels of this model are always enabled, fairness simplifies to $\forall l.\ \Box \Diamond \downarrow_l l$.

The proof follows by first proving that a single increment eventually happens via induction on the fairness assumption, for the progressing label. We can then use this fact alongside induction on $i := n - j$, where $\downarrow_s s_j$ to prove the property.

We can instantiate the general model to verify reachability of a concrete model which counts how many times a coin alternatingly result in heads and tails [2].



We define the label and projection function as follows:

$$\mathsf{L} := \mathbb{B} \qquad f := \mathsf{is_even}$$

This immediately yields the reachability progress as a corollary of reachability for the general model. We can similar define instantiations for a model with one participant:

$$\mathsf{L} := \mathsf{unit} \qquad f := id$$

and the infinite model with countably infinite participants:

$$\mathsf{L} := \mathbb{N} \qquad f := id$$

2.10 Classical Extension

By Rocq convention our logic is constructive, but includes an optional classical extension. The extension axiomatise excluded middle in the meta-logic, which yields excluded middle in the temporal logic $\vdash P \vee \neg P$. Consequently, this yields conventional classical axioms of temporal logic:

$$\Box P \equiv \neg \Diamond \neg P \qquad \Diamond P \equiv \neg \Box \neg P$$

[2] Inspired by [42], where the coin corresponds to a scheduler between two concurrent threads that take turns performing work.

The classical variant is necessary for liveness proofs in which progress must be proven by contradiction; the steps necessary for progress can only be proven assuming progress never happens. We use this pattern in our Stenning example covered in §4.

3 The MoTeL Proofmode

We now cover how MoTeL (1) manages the proof context of the temporal logic by treating as a map over named hypotheses, and (2) provides tactics for the introduction and elimination of modalities, while automatically keeping the proofmode consistent. We first give an overview how proofs are carried out in MoTeL, focusing on proof context management (§3.1) and working with modalities (§3.2). We conclude by covering the mechanisation of the fair scheduler example of §2.9 (§3.3). Note that we additionally inherit any uncovered core functionality of MoSeL, described in [25] and more presently [35], such as embedded logic-level induction.

3.1 Managing the Proof Context

Let us first recall how the Rocq proofmode works, by considering the following proof:

```
Lemma modus_ponens :
  forall (P Q : Prop), (P -> Q) /\ P -> Q.
Proof.
  intros P Q [HPQ HP]. apply HPQ in HP as HQ. apply HQ.
Qed.
```

Rocq starts in the proof state:

```
  ============================
  forall (P Q : Prop), (P -> Q) /\ P -> Q
```

Here, the line separates the (initially empty) proof context and the goal. The first tactic `intros P Q [HPQ HP]` introduces the universally quantified propositions and the implication antecedent, naming them in the process. The `[HPQ HP]` pattern uses conjunction elimination to immediately turn the introduced conjunct into two separate hypotheses. The resulting proof state is:

```
  P, Q : Prop
  HPQ: (P -> Q)
  HP : P
  ============================
  Q
```

Corresponding to $P \to Q, P \vdash Q$, modulo names. The tactic `apply HPQ in HP as HQ` then applies `HPQ` in `HP` and renames the result to `HQ`, using the rule of implication elimination:

```
  P, Q : Prop
  HPQ: (P -> Q)
  HQ : Q
  ============================
  Q
```

Finally, `apply HQ` applies `HQ` to the goal, finishing the proof.

A corresponding proof in MoTeL is as follows:

```
Lemma mosel_modus_ponens :
  ⊢ ∀ (P Q : tProp), (P → Q) ∧ P → Q.
Proof.
  iIntros (P Q) "[HPQ HP]".
  iDestruct ("HPQ" with "HP") as "HQ".
  iApply "HQ".
Qed.
```

The first key distinction is that we use $\vdash$ to denote that we are working in our temporal logic, and quantify over temporal logic propositions tProp. We use the symbols $\forall$, $\to$, and $\wedge$, following MoSeL convention. Our tactics are prefixed with an 'i' (legacy notation referring to "Iris", inherited from MoSeL), and use strings as identifiers necessary for integrating with Rocq's extensible parser.

The `iIntros (P Q) "[HPQ HP]"` tactic has two distinct parts; we first introduce the proposition terms as meta-level (Rocq) terms (as they are not indexed by a trace), and then the antecedent as a temporal logic hypothesis, resulting in:

```
  P, Q : tProp
  ============================
  "HPQ": (P -> Q)
  "HP" : P
  ----------------------------*
  Q
```

The occurrences between the two lines are then the proof context of the embedded logic, which are each indexed by the current trace under consideration. The `iDestruct ("HPQ " with "HP") as "HQ"` tactic, similar to Rocq, applies the implication `"HPQ"` to `"HP"`, naming it `"HQ"`, resulting in:

```
  P, Q : tProp
  ============================
  "HQ" : Q
  ----------------------------*
  Q
```

Note that `"HPQ"` is gone. This is an artifact of MoSeL, which treats propositions linearly, and thus consumed when used. To preserve facts needed multiple times, one can first use the rule $P \vdash P \wedge P$. We can finish the proof via `iApply "HQ"`.

3.2 Managing Temporal Modalities

A key feature of MoTeL is how it handles temporal modalities, such as globally and eventually. This is supported through MoSeL's tactics for modality introduction `iModIntro` and elimination `iMod`.

Modality introduction. The desiderata of the `iModIntro` tactic is to determine the top-level modality M of the goal, and introduce it, stepping into the world(s) where it must hold, while preserving the strongest proof state possible. For a modality to be compatible with introduction it must satisfy the following laws:

$$M\,P \wedge M\,Q \vdash M\,(P \wedge Q) \qquad \frac{P \vdash Q}{M\,P \vdash M\,Q} \qquad \mathsf{True} \vdash M\,\mathsf{True}$$

This is the case for our globally (□) and next (○) modalities. The intuition is that when used in a proof context:

$$MP_1 \land \ldots \land MP_n \vdash MQ$$

`iModIntro` will first use the law of distributivity over conjunction to combine all the hypotheses, resulting in:

$$M(P_1 \land \ldots \land P_n) \vdash MQ$$

after which the monotonicity law can be used to step into the world at M, yielding:

$$P_1 \land \ldots \land P_n \vdash Q$$

while preserving the hypothesis names. The final law is necessary for adding a modality to the proof context whenever it is empty. As all hypotheses must be under M, any existing hypothesis not under M must either be updated or discarded. MoSeL handles this automatically, as we will cover below. In the following proof state:

```
P, Q : tProp
============================
"HPQ": ○ (P -> Q)
"HP" : ○ P
"HQ" : Q
----------------------------*
○ Q
```

using `iModIntro` will result in

```
P, Q : tProp
============================
"HPQ": (P -> Q)
"HP" : P
----------------------------*
Q
```

The `"HQ" : Q` hypothesis is discarded, as it is not of the form ○ _. The remaining hypotheses have their ○ removed, but are otherwise preserved.

The proofmode can additionally carry out this procedure while considering propositions that can be turned into propositions of the form ○ _. Consider the following example:

```
P, Q : tProp
============================
"HPQ": □ (P -> Q)
"HP" : ○ P
"HQ" : Q
----------------------------*
Q
```

As we have $\Box P \vdash \bigcirc \Box P$, the proofmode will automatically first insert the requisite ○ when using `iModIntro`, before removing it again, resulting in:

```
P, Q : tProp
============================
"HPQ": □ (P -> Q)
"HP" : P
----------------------------*
Q
```

In some cases, there are multiple ways in which a hypothesis can be transformed to fit the introduced modality. Consider:

```
P, Q : tProp
============================
"HPQ": □ ○ (P -> Q)
"HP" : ○ P
"HQ" : Q
----------------------------*
○ Q
```

Here, we could proceed as before, and add ○ to `"HPQ"`, but this results in `□ ○ (P -> Q)`, rendering an unprovable state. The stronger option is to use $\Box \bigcirc P \dashv\vdash \bigcirc \Box P$, resulting in:

```
P, Q : tProp
============================
"HPQ": □ (P -> Q)
"HP" : P
----------------------------*
Q
```

MoSeL lets us instrument the proofmode with a priority order of transformations, in this case first trying to exhume and eliminate a ○, before simply adding it. As a part of our proofmode instantiation, this instrumentation is given for the conventional transformations of the core temporal modalities. Further extension is readily available, which crucially allows instrumenting the proofmode with behaviour on how to treat custom fixpoints and model-specific.

Remark 3.1 (The persistent (global) context)**.** To better structure proofs, MoSeL includes a separate proof context for persistent (here global) facts, where all facts are implicitly under a □. The modality tactics work similarly for the separate context, for example the above looks like the following:

```
P, Q : tProp
============================
"HPQ": ○ (P -> Q)
----------------------------□
"HP" : ○ P
"HQ" : Q
----------------------------*
○ Q
```

Which `iModIntro` turns into:

```
P, Q : tProp
============================
"HPQ": (P -> Q)
----------------------------□
"HP" : P
----------------------------*
Q
```

Modality elimination. Similar to `iModIntro` the desiderata of modality elimination via the `iMod` tactic is to eliminate a modality of a specified hypothesis of the form $M\,P$, stepping to the world where P holds, while preserving the strongest proof state possible. For a modality to be compatible with

elimination, it must satisfy the following law:

$$N(P \to Q) \vdash MP \to MQ$$

Where N is another modality satisfying the laws required by `iModIntro`. This is the case for our eventually ($\Diamond$) and until ($\cup$) modalities, where $N := \Box$. So, for a proof context:

$$NR_1 \land \ldots \land NR_n \land MP \vdash MQ$$

`iMod` first use the above law with the declared hypothesis:

$$NR_1 \land \ldots \land NR_n \vdash N(P \to Q)$$

after which N is introduced similar to `iModIntro`.

$$R_1 \land \ldots \land R_n \land P \vdash Q$$

This effectively steps to the world at M, while discarding any hypotheses not preserved under N. Similar to `iModIntro` the tactic preserve hypothesis names, and may first update/discard the goal and hypotheses. Consider the following:

```
P, Q : tProp
============================
"HPQ" : (P -> ◇ Q)
----------------------------□
"HPQ'": ◇ (P -> Q)
"HP"  : ◇ P
----------------------------*
◇ Q
```

Here, `iMod "HP"` will "step" to the place in which `"HP"` holds. Naively applying monotonicity would result in an unprovable goal Q. However, the tactic first uses $\Diamond P \dashv\vdash \Diamond\Diamond P$ to update the goal, preserving the $\Diamond$. This ultimately results in:

```
P, Q : tProp
============================
"HPQ": (P -> ◇ Q)
----------------------------□
"HP" : P
----------------------------*
◇ Q
```

The global hypothesis `"HPQ"` remains as it is preserved through introduction of $\Box$. Conversely, `"HPQ'"` is lost as the position in the trace where it holds might have been before the position we stepped to.

3.3 Mechanising the Fair Progress Example in MoTeL

To verify the fair progress example from §2.9, we first define the model in terms of its state, label, and transition relation in Rocq.

```
Definition state : Set := nat.
Context (label : Set).
Context (f : nat → label).
Inductive steps : state → label → state → Prop :=
| my_step_succ i : steps i (f i) (i+1)
| my_step_fail i j : f i ≠ j → steps i j i.
```

We then prove the model-level axioms, e.g.:

```
Lemma step_succ i :
  ↓s i ∧ ↓l (f i) ⊢ ○ ↓s (i+1).
Proof.
  iIntros "Hsl".
  iDestruct (step-label with "Hsl") as (s' Hsteps) "Hs".
  by inversion Hsteps; simplify_eq.
Qed.
```

We first use STEP-LABEL to arrive at:

```
label : Set
f : nat → label
i : nat
s' : state
Hsteps : steps i (f i) s'
============================
"Hs" : ○ ↓s s'
----------------------------------------*
○ ↓s (i + 1)
```

Using case analysis on the transition relation, we can deduce that the trace must step in accordance with `step_succ`, concluding the proof.

We assume fairness using a Rocq-level axiom `Axiom fair : ∀ i, ⊢ ◇ ↓l i`. We first prove the following lemma via induction on the fairness assumption (elided for brevity sake):

```
Lemma eventually_step i :
  ↓s i ⊢ ◇ (↓s i ∧ ↓l (f i)).
```

The lemma concludes that we eventually step over the productive label `f i`, in the preserved state `↓s s`. We then prove the following progress lemma:

```
Lemma eventually_incr i :
  ↓s s ⊢ ◇ ↓s (s+1).
Proof.
  iIntros "Hs".
  iMod (eventually_step with "Hs") as "Hsl".
  iDestruct (step_succ with "Hsl") as "Hs".
  iApply ltl_next_eventually.
  iModIntro. iModIntro. iApply "Hs".
Qed.
```

The proof first use `eventually_step` to conclude `◇ (↓s s ∧ ↓l (f s))`, and immediately eliminate the eventually modality $\Diamond$ from the result via `iMod`. We then use `step_succ` to arrive at `○ ↓s (s+1)`. We conclude the proof by applying $\bigcirc \Diamond P \vdash \Diamond P$ to the goal, and introducing $\bigcirc$ and $\Diamond$. The theorem `eventually_n n : ↓s 0 ⊢ ◇ ↓s n` is carried out via induction on the difference between the current state i and n.

Deriving properties about model instantiation is a simple matter of applying the theorem with a valid model:

```
Lemma eventually_n_coin n :
  ↓s 0 ⊢@{tProp nat bool (steps bool even)} ◇ ↓s n.
Proof. apply eventually_n. Qed.
```

4 Case Studies

We now give an overview of examples mechanised via MoTeL. Table 1 gives an overview of the lines of code of each example, compared to prior examples where applicable.

4.1 Trillium Examples

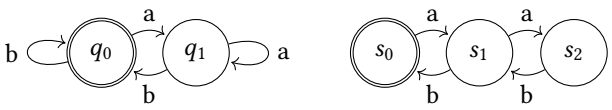
Timany et al. [42] developed the Trillium refinement logic and used it to prove liveness-preserving refinements between a program and a high-level model. To conclude liveness of a program, one must prove liveness of the high-level model, which they did manually in Rocq. Trillium is a quintessential use case for MoTeL, as the goal is to have a fully mechanised proof of liveness (i.e. not exported). We have carried out proofs of (variants of) the liveness examples of [42]. One example "even-odd" is a variation of the fair coin example from §2.9. Another "yes-no" example is similar, having two threads alternate between counting down from an initial value n until 0, proving termination. For brevity sake, we elide further details on the yes-no example, which can be found in our Rocq mechanisation [2].

4.2 Stenning's Reliable Transfer Protocol

Hinrichsen et al. [17, 18] extend Trillium [42] to verify liveness-preserving refinement of distributed systems in unreliable networks. Their key example is a higher-order remote procedure call that is refined by a model akin to Stenning's reliable transfer protocol [33]. The key features of the model is that it is labeled by an actor-action pair, that respectively denote which node took a step, and whether they sent or received. The former allows expressing a strong fairness property between the actors, while the latter is used to express network fairness; that messages sent infinitely often are received by participants who try to receive infinitely often. The liveness property we prove is that every message in a sequence to be transferred is eventually observed to be delivered. Further details are included in the appendix §C.

4.3 Parity Automata

Parity Automata (PA) [14] are one of the most fundamental tools used for providing algorithms for model checking temporal logics, and are expressively equivalent with all standard non-deterministic infinite automata. The acceptance condition of an PA is expressible in our logic (for any given Kripke structure - including infinite ones). Thus our approach inherits the ability to formally prove that any structure has a property of a vast plethora of logics whose model checking reduces to checking acceptance by an infinite automaton (see e.g., logics with team semantics [15], the μ-calculus [14], hyperlogics [4, 11], and multi-agent systems [23].) As a motivating example, we instantiate MoTeL with the following cartesian product over an automata $\mathcal{A}$ and a model $\mathcal{M}$:



We prove the following property, for the parity assignment: $\Omega\, q_0 := 1$ and $\Omega\, q_1 := 2$:

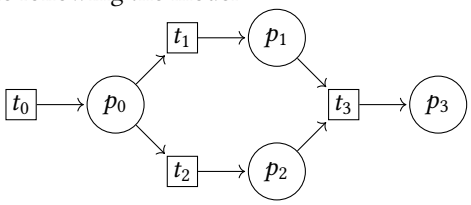
$$\downarrow_s (q_0, s_0) \vdash (\exists n.\ \mathsf{is_even}\ n) \wedge (\Box \Diamond \downarrow_s^{\Omega \cdot \mathsf{fst}} n) \wedge (\forall m.\ \Box \Diamond \downarrow_s^{\Omega \cdot \mathsf{fst}} m \rightarrow m \leq n)$$

This property (right of the $\vdash$) captures exactly the parity automata acceptance condition, requesting that the maximal parity encountered infinitely often over the runs of the automaton for a LTSs needs to be even. The specific automaton $\mathcal{A}$ we have designed captures the μ-calculus (and LTL) formula: "no trace of the input LTS ends with infinitely many consecutive b actions", but since we describe the acceptance condition for *any* automaton, the specific property is not the main focus. Since PA are the primary tool for answering the model checking problem of a plethora of temporal logics ([11, 14, 24]), being able to express this property in our logic immediately entails that our logic expressively contains the above (w.r.t. model checking over finite and infinite LTSs).

4.4 Petri Nets

Petri nets [29] is a popular framework for modeling infinite-state systems. In the standard Petri net paradigm, one has a set of places, a set of transitions, and arcs going from place to transitions, and transition to place. In a configuration each state is assigned a natural number of "marks". A transition is considered enabled if the places of all incoming arcs have at least one mark. In every step of the system one enabled transition is fired, subtracting one mark from all inbound places, and adding one mark to all outbound places.

As a motivating example, we instantiate our logic with the following the model:



The trace state is the configuration assignment, and the trace labels are fired transitions. The stepping relation is the transition firing condition explained above. The property we want to prove is reachability of p_3:

$$\downarrow_s \{p_0 := 0; p_1 := 0; p_2 := 0; p_3 := 0\} \vdash \Diamond \downarrow_s (\lambda s, (\exists x.\ s[p_3] = \mathbf{some}\ x \wedge x > 0)).$$

That is, given an initial state where all places have 0 marks, we will eventually reach a state where p_3 has a mark.

The property only holds under fairness, as the two traces that always only fire either t_1 or t_2 never enable t_3. We can

Table 1. Overview of the MoTeL Rocq mechanisation lines of code (LOC)

Examples			Proofmode	
Component	Prior LOC	MoTeL LOC	Component	MoTeL LOC
Fair scheduler (Trillium Even/Odd)	237	116	Trace definition	108
Trillium yes/no	189	74	Base logic + Proofmode	1852
Stenning protocol	949	663	Fixpoints	632
Parity automata		300	Adequacy + Extraction	122
Petri net		432	Classical extension	125
Total		**1585**	**Total**	**2839**

express the fairness condition as follows:

$$\begin{aligned}\mathsf{meta_petri_enabled}\, t\, s &\triangleq \forall p.\ (p,t) \in \mathsf{input_arcs} \rightarrow \\ &\quad \exists x.\ s[p] = \mathbf{some}\, x \wedge x > 0 \\ \mathsf{petri_enabled}\, t &\triangleq \downarrow_s (\mathsf{meta_petri_enabled}\, t) \\ \mathsf{petri_fair} &\triangleq \forall t.\ \Box \Diamond\, \mathsf{petri_enabled}\, t \rightarrow \Diamond \downarrow_l t\end{aligned}$$

5 Developing MoTeL

In this section we cover how we instantiate the MoSeL proofmode interface for our temporal logic. An overview of the amount of Rocq lines of code it took to develop the proofmode can be seen in Table 1. Instantiating MoSeL for a logic is done in two steps. (1) instantiate the MoSeL interface with definitions and rules of the core primitives and modalities (§5.1), and (2) extend the MoSeL typeclass infrastructure to support introduction and elimination of custom modalities (§5.2).

5.1 MoSeL Interface

MoSeL is designed for step-indexed bunched implication (SBI) logics [28]. To instantiate MoSeL, one must provide an enriched SBI interface: definitions and laws for the standard propositional connectives ($\wedge$,$\vee$,$\rightarrow$,$\forall$,$\exists$), bunched implication connectives (Emp,$*$,$-\!\!*$), the later modality ($\triangleright$) and a persistent modality ($\boxdot$). For our logic, the propositional connectives are defined via pointwise lifting, as covered in §2.3, for which the conventional propositional laws follow as expected. While the bunched implication and step-indexing connectives are required, previous instantiations of MoSeL [34] demonstrate how they can be side-stepped by defining them to coincide with their propositional counterparts for logics with no step index or resources. We follow this approach, and give the following definitions:

$$\mathsf{Emp} \triangleq \mathsf{True} \qquad P * Q \triangleq P \wedge Q$$
$$\triangleright P \triangleq P \qquad P -\!\!* Q \triangleq P \rightarrow Q$$

The MoSeL desiderata for the persistent modality is to internalise the notion of (separation logic) propositions holding without resources, and thereby being duplicable. The modality is inherently connected with the persistent proofmode context (covered in §3), where all facts are considered under the persistent modality. Additionally, fixpoints theorems are defined in terms of the persistent modality of choice. For a resource-less logic such as ours, all propositions are duplicable, and so the original MoSeL intent of the persistent modality serve no purpose. However, a key insight is that we can re-utilize the persistent modality features by defining the persistent modality as our global modality $\boxdot P \triangleq \Box P$. For a logic without resources, the laws required by the persistent modality are as follows:

$$\frac{P \vdash Q}{\boxdot P \vdash \boxdot Q} \qquad \mathsf{True} \vdash \boxdot \mathsf{True}$$

$$\boxdot P \wedge \boxdot Q \vdash \boxdot(P \wedge Q) \qquad \boxdot P \vdash \boxdot \boxdot P$$

These laws follow from the conventional axiomatic foundation of the globally modality that our logic includes.

5.2 Modality Extension

As covered in §3.2, MoSeL supports modality introduction and elimination via the tactics `iModIntro` and `iMod`, for any compatible modality. Once instantiated, MoSeL supports `iModIntro` for the persistent (thus globally) modality for free, as it's prerequisite laws satisfy the `iModIntro` requirements. For other modalities, such as next ($\circ$), eventually ($\Diamond$), and until ($\cup$), one needs to manually extend it's tactic infrastructure. This is done via Rocq's typeclass system, which the MoSeL tactics are built on top of. Notably, one has to provide typeclass instances of two MoSeL typeclasses `FromModal` and `ElimModal`, for `iModIntro` and `iMod`, respectively. We now cover how we extended our MoSeL instantiation to support our temporal modalities. For brevity sake, we elide details about the distinction between the persistent context and the spatial context.

iModIntro and the Next Modality. When `iModIntro` is used, it will look for an instance of the MoSeL typeclass `FromModal`, which operationalise the requirements covered above. The typeclass simultaneously lets us extend `iModIntro` with support for new modalities, such as the next modality, by providing Rocq typeclass instances for new modalities.

It's (simplified, for brevity) Rocq definition at time of writing is as follows:

```
Class FromModal (M : modality tProp) (P Q : tProp) :=
  from_modal : M Q ⊢ P.
```

It dictates that a goal `P` can be turned into a goal `M Q`. Notably, `M` must be a `modality` Rocq construction. This carries two functionalities. (1) the modality (`M : tProp -> tProp`) satisfy the necessary modality laws. (2) the modality is associated with an action that transform/discard the hypotheses of the proofmode context, so that all remaining hypotheses are under `M`.

To facilitate the modality transformation of propositions into/from propositions with M at the top-level, it is MoSeL convention to use modality-specific typeclasses, e.g. for the next modality:

```
Class IntoNext (P Q : tProp) := into_next : P ⊢ ○ Q
Class FromNext (P Q : tProp) := from_next : ○ P ⊢ Q
```

When a tactic is invoked it will search for an applicable instance of these typeclasses, effectively constructing an extensible proof search mechanism. All modalities should have instances reflecting their reflexive laws as their base case. For example, the next modality has:

```
Instance into_next_next P : IntoNext (○ P) P.
Instance from_next_next P : FromNext P (○ P).
```

Which both translate to, `○ P ⊣⊢ ○ P`. The action of the modality construction of a given modality is defined in terms of the corresponding typeclasses. We elide further details on the modality construction, and instead refer to [25].

The `FromModal` instance for the next modality is then:

```
Instance from_modal_next (P Q : tProp) :
  FromNext Q P → FromModal modality_next P Q | 1.
```

Given the above typeclasses, `iModIntro` will work on goals of the form `○ _`, and discard any hypothesis not of the form `○ _`. This is even the case for hypotheses such as `□ P`, which could first be turned into `○ □ P`, at which point it would match the above typeclass. To extend the proofmode to automatically make such transformations, we add the following instance:

```
Instance into_next_always P : IntoNext (□ P) (□ P) | 1.
```

The proofmode will now preserve the global hypothesis by first adding a next modality before immediately eliminating it along with the modality in the goal. Now consider the case where there are multiple ways of transforming a hypothesis, such as `□ ○ P`. In this case, we can simply add a typeclass instance with a higher priority:

```
Instance into_next_always_comm p P :
  IntoNext (□ ○ P) (□ P) | 0.
```

The proofmode will now first find the latter typeclass instance, turning the hypothesis `□ ○ P` into `○ □ P`, ending up with the stronger hypothesis `□ P`, instead of `□ ○ P`. Finally, the above typeclass can be improved slightly further, as it now only considers hypotheses where next is nested right after a globally modality. This means that a hypothesis such as `□ □ ○ P`, would hit the first typeclass, instead of the latter. This can be alleviated by using the following improved typeclass instead:

```
Instance into_next_always_comm p P Q :
  IntoNext p P Q →
  IntoNext p (□ P) (□ Q) | 0.
```

This will recursively exhume a next modality from the inner proposition, if possible, and otherwise backtrack and find the initial typeclass instance.

MoTeL includes typeclass instances for most expected patterns. Even then, given the ability to derive new custom modalities in the logic, it is imperative that a user can additionally extend the proofmode with new instances for how such custom modalities will work with the existing modalities such as the next modality.

ElimModal and the Eventually/Until Modalities. When `iMod` is used, it will look for an instance of the MoSeL typeclass `ElimModal`, which operationalise the requirements for modality elimination covered above. The (simplified, for brevity sake) definition is as follows:

```
Class ElimModal
  (M : modality tProp) (P P' Q Q' : tProp) :=
    elim_modal : M (P' → Q') ⊢ P → Q.
```

The typeclass captures the requisite law for modality elimination, while facilitating updating the hypothesis/goal under consideration simultaneously. To make sense of this, consider the following instance for eventually (◇).

```
Instance elim_modal_eventually P P' Q Q' :
  IntoEventually P P' →
  IntoEventually Q Q' →
  FromEventually Q' Q →
  ElimModal modality_persistently P P' Q Q | 2.
```

That is, in any goal `Q`, where `Q ⊣⊢ ◇ Q'`, we can eliminate a hypothesis `P`, where `P ⊢ ◇ P'`, into `P'`, while preserving the goal. The rest of the proof context is updated according to the introduction of □, which only preserves the persistent context. This codifies the behaviour seen in §3, where all other non-persistent hypotheses are lost.

The instance for until (∪) is:

```
Instance elim_modal_until :
  IntoUntil P Q1 P2 →
  IntoUntil Q Q1 Q2 →
  FromUntil Q1 Q2 Q →
  ElimModal modality_persistently P P2 Q Q | 1.
```

Here, the goal must be logically equivalent to `Q1 ∪ Q2`, while the eliminated hypothesis must be equivalent to `Q1 ∪ P2`. Note that the first argument `Q1` must coincide, as stepping the goal ahead requires proving that `Q1` holds until that point, which we get from the eliminated hypothesis. This typeclass

supersedes the above, as they coincide whenever whenever `Q1` is `True`, which is the case for eventually.

6 Related Work

The original deductive system for LTL, which is entirely included in our logic as shown in Figure Fig. 1 was presented by Gabbay et al. [12], while the complete axiomatisation for the μ-calculus was later presented by Kozen [24]. Since then, deductive systems for standard and extended temporal logics emerge continuously [16, 22, 45].

Our logic extends the standard modal μ-calculus by allowing quantification over infinitary state spaces, expressed as higher order Rocq types (as opposed to first-order extensions such as [6]), and subsequently is interpreted over infinite state models, allowing for a program modelling similar to that of the TLA$^+$ [27] and SPIN [19] tools for specifying programs, specifications, and proofs. Over such systems (assuming a reasonable, finite representation of a program) model-checking is not always decidable [1], and thus our proofmode offers an interactive way of reasoning about properties of systems that lie outside the reach of standard model checkers. An alternative way that quantification has been used as an addition to temporal logic is that of hyperlogics. However, the quantification semantics we equip is incompatible with those of hyperlogics [9], which are a relatively recent development in the area of specification logics [3, 8] for capturing information flow, privacy, and service level requirements. Even though hyperLTL, the most standard hyperlogic, is equipped with a deductive system [48], the mechanisation of this kind of semantics requires an augmented notion of proof context.

Moreover, our logic equips more expressive predicates over the notion of a transition from a state, thus allowing us to express properties such as fairness [43]. Even so, our logic remains linear time, and does not inhibit the full expressivity of branching-time logic; for one we cannot reason about single path properties.

Ioannidis et al. [20] addressed the tractable mechanisation of temporal properties via temporal logic from the perspective of composition. They instrument a computation tree-based logic with Hoare logic-style reasoning, permitting composition of sequential proofs, in style with program logic. This provides a high-level reasoning style suitable for mechanisation, but has the same problem as other embedded logics; it cannot use the (Rocq) proof assistant named hypotheses unless they break abstraction.

Coupet-Grimal [10] mechanise the metatheory of LTL in Rocq. They use an embedded logic, with no focus on automation or reuse for proofs of concrete examples. They have no notion of point predicates, and instead include a primitive definition of label-enabledness to define fairness conditions. Their mechanisation is similar to the official Rocq LTL development [40]. Wiegley [46] gives a similar mechanisation, but with a foundation based on [45].

Wu and Goré [47] and Goré and Peigné [13] formalise decidable tableu procedures in Lean and Rocq respectively, for automatically mechanised proofs of temporal propositions in LTL. To achieve decidability, their logic use propositional variables as opposed to the unrestricted state-space propositions of our logic, and well-structured loops.

Vin et al. [44] develops a Lean mechanisation of a higher-order version of LTL with unrestricted state-space predicates, similar to our logic. They provide some automation through simplification sets, but generally approach proofs by unfolding the embedded logic, exposing the underlying trace.

The Isabelle LTL project [31] formalise the metatheory of LTL, which has primarily been used for further metatheoretic properties, e.g. translation into ω-automata [30] and normalisation procedures [32]. The verification of concrete examples rely on normalisation through rewrite lemmas and equivalence relations.

7 Future Work

We identify several opportunities for future work.

Branching time logic. Branching time logic (also called computation tree logic (CTL)) is more expressive than LTL, being able to express path existance properties. Similar to LTL, CTL has temporal modalities with modal laws similar to those required by MoSel. This suggest that an instantation similar to MoTeL may be possible. Even so, it is not obvious how the proofmode should handle the dynamic between modalities pertaining to all or single paths of the trace.

Temporal separation logic. The instantiation of MoSeL, built for separation logic, with temporal logic begs the question of how one could define the separation logic connectives if resources were added to the logic. The idea of modular reasoning about temporal properties is compelling, and worthy of future study. Even so, it is unclear how composition would be defined or even utilized.

Structural temporal logic. The structural temporal logic developed by Ioannidis et al. [20] targets compositional reasoning about temporal properties of programs, by instrumenting a computation tree-based logic with Hoare logic-style reasoning. Their extension over CTL is akin to how separation logic has been used for program verification. As MoSeL already integrates foundational separation logic with Hoare logic-style reasoning, an instantiation for structural temporal logic may be possible.

General proofmode interface. MoTeL is fixed to our definition of traces, their operators hd/tl, and the logical correspondents $\downarrow/\circ$. This requires users to translate verified properties to their own systems manually. A generalisation

of the proofmode interface, would allow instantiation with any world provided it satisfies the axioms in Fig. 1. Such a generalisation could simultaneously be a first step towards supporting other temporal logics, e.g. branching time logic.

Hyperlogics. The model checking problem of several standard hyperlogics (e.g. [11]), comes at an exponential cost, making interactive verification a compelling alternative. Compelling future work is investigating interactive verification via hyperlogics via an approach similar to MoTeL.

MoTeL in Lean. MoSeL has been fully ported to the Lean proof assistant, as a part of the Iris-Lean project [36], and used as the foundation for the Iris Proof Mode in Lean. The proofmode interface is based on the same modal law as MoSeL in Rocq. This suggests that the same can be done to port MoTeL in Lean.

Data-Availability Statement

The Rocq development for this paper can be found in [2].

References

[1] Parosh Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. 1996. General decidability theorems for infinite-state systems. *Proceedings - Symposium on Logic in Computer Science*. doi:10.1109/LICS.1996.561359

[2] Anonymous Authors. 2026. Rocq Mechanisation of this paper (found in supplementary material).

[3] Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Niklas Metzger. 2023. Second-Order Hyperproperties. In *Computer Aided Verification: 35th International Conference, CAV 2023, Paris, France, July 17–22, 2023, Proceedings, Part II*. doi:10.1007/978-3-031-37703-7_15

[4] Laura Bozzelli, Adriano Peron, and César Sánchez. 2022. Expressiveness and Decidability of Temporal Logics for Asynchronous Hyperproperties. In *33rd International Conference on Concurrency Theory (CONCUR 2022) (Leibniz International Proceedings in Informatics (LIPIcs))*. doi:10.4230/LIPIcs.CONCUR.2022.27

[5] J Richakd Büchi. 1990. Weak second-order arithmetic and finite automata. In *The Collected Works of J. Richard Büchi*. 398–424.

[6] Diego Calvanese, Giuseppe De Giacomo, Marco Montali, and Fabio Patrizi. 2018. First-order μ-calculus over generic transition systems and applications to the situation calculus. *Information and Computation* (2018). doi:10.1016/j.ic.2017.08.007

[7] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (Eds.). 2018. *Handbook of Model Checking*. doi:10.1007/978-3-319-10575-8

[8] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. 2014. Temporal Logics for Hyperproperties *(LNCS)*, Martín Abadi and Steve Kremer (Eds.). doi:10.1007/978-3-642-54792-8_15

[9] Michael R. Clarkson and Fred B. Schneider. 2010. Hyperproperties. *J. Comput. Secur.* (2010). doi:10.3233/JCS-2009-0393

[10] Solange Coupet-Grimal. 2003. An Axiomatization of Linear Temporal Logic in the Calculus of Inductive Constructions. *J. Log. Comput.* (2003). doi:10.1093/LOGCOM/13.6.801

[11] Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. 2015. Algorithms for Model Checking HyperLTL and HyperCTL*. In *Computer Aided Verification*, Daniel Kroening and Corina S. Păsăreanu (Eds.).

[12] Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. 1980. On the temporal analysis of fairness. In *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. doi:10.1145/567446.567462

[13] Rajeev Goré and Anthony Peigné. 2026. Verified Tableaux: from Modal Logics to Modal Fixpoint Logics. *Journal of Automated Reasoning* 70 (2026). https://api.semanticscholar.org/CorpusID:288127088

[14] Erich Graedel, Wolfgang Thomas, and Thomas Wilke. 2002. *Automata Logics, and Infinite Games: A Guide to Current Research*. doi:10.1007/3-540-36387-4

[15] Jens Oliver Gutsfeld, Arne Meier, Christoph Ohrem, and Jonni Virtema. 2022. Temporal Team Semantics Revisited. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '22)*. doi:10.1145/3531130.3533360

[16] Joseph Y. Halpern, Ron van der Meyden, and Moshe Y. Vardi. 2004. Complete Axiomatizations for Reasoning about Knowledge and Time. *SIAM J. Comput.* (2004). doi:10.1137/S0097539797320906

[17] Jonas Kastberg Hinrichsen, Léo Stefanesco, Lars Birkedal, and Amin Timany. 2026. A Higher-Order Separation Logic for Verifying Liveness-Preserving Refinements of Distributed Systems. (2026). https://hal.science/hal-05647709 working paper or preprint.

[18] Jonas Kastberg Hinrichsen, Léo Stefanesco, Lars Birkedal, and Amin Timany. 2026. Rocq Mechanisation of "A Higher-Order Separation Logic for Verifying Liveness-Preserving Refinements of Distributed Systems", provided by authors.

[19] G.J. Holzmann. 1997. The model checker SPIN. *IEEE Transactions on Software Engineering* (1997). doi:10.1109/32.588521

[20] Eleftherios Ioannidis, Yannick Zakowski, Steve Zdancewic, and Sebastian Angel. 2025. Structural Temporal Logic for Mechanized Program Verification. OOPSLA2 (2025). doi:10.1145/3763091

[21] Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015. Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning. In *POPL*. doi:10.1145/2676726.2676980

[22] Yonit Kesten and Amir Pnueli. 2002. Complete Proof System for QPTL. *Journal of Logic and Computation* (2002). doi:10.1093/logcom/12.5.701

[23] Jeremy Kong and Alessio Lomuscio. 2017. Model checking multi-agent systems against LDLK specifications. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*. AAAI Press.

[24] Dexter Kozen. 1983. Results on the propositional μ-calculus. *Theoretical Computer Science* (1983). doi:10.1016/0304-3975(82)90125-6 Special Issue Ninth International Colloquium on Automata, Languages and Programming (ICALP) Aarhus, Summer 1982.

[25] Robbert Krebbers, Jacques-Henri Jourdan, Ralf Jung, Joseph Tassarotti, Jan-Oliver Kaiser, Amin Timany, Arthur Charguéraud, and Derek Dreyer. 2018. MoSeL: A General, Extensible Modal Framework for Interactive Proofs in Separation Logic. ICFP (2018). doi:10.1145/3236772

[26] Robbert Krebbers, Amin Timany, and Lars Birkedal. 2017. Interactive Proofs in Higher-Order Concurrent Separation Logic. In *POPL*. doi:10.1145/3009837.3009855

[27] Leslie Lamport. 1992. Hybrid Systems in TLA$^+$. In *Hybrid Systems (Lecture Notes in Computer Science)*, Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel (Eds.). doi:10.1007/3-540-57318-6_25

[28] Peter W. O'Hearn and David J. Pym. 1999. The logic of bunched implications. *Bull. Symb. Log.* (1999). doi:10.2307/421090

[29] W. Reisig. 2013. *Understanding Petri nets. Modeling techniques, analysis methods, case studies. Translated from the German by the author.* doi:10.1007/978-3-642-33278-4

[30] Benedikt Seidl and Salomon Sickert. 2019. A Compositional and Unified Translation of LTL into ω-Automata. *Archive of Formal Proofs* (2019). https://isa-afp.org/entries/LTL_Master_Theorem.html, Formal proof development.

[31] Salomon Sickert. 2016. Linear Temporal Logic. *Archive of Formal Proofs* (2016). https://isa-afp.org/entries/LTL.html, Formal proof development.
[32] Salomon Sickert. 2020. An Efficient Normalisation Procedure for Linear Temporal Logic: Isabelle/HOL Formalisation. *Archive of Formal Proofs* (2020). https://isa-afp.org/entries/LTL_Normal_Form.html, Formal proof development.
[33] N.V. Stenning. 1976. A data transfer protocol. *Computer Networks (1976)* (1976). doi:10.1016/0376-5075(76)90015-5
[34] The MoSeL Team. 2026. Classical Logic in MoSeL. https://gitlab.mpi-sws.org/iris/examples/-/blob/master/theories/cl_logic/bi.v
[35] The MoSeL Team. 2026. MoSeL. https://gitlab.mpi-sws.org/iris/iris/-/blob/master/docs/proof_mode.md
[36] The Iris Lean Team. 2026. Iris in Lean. https://github.com/leanprover-community/iris-lean
[37] The Iris Team. 2026. The Iris Project. https://iris-project.org/
[38] The Isabelle Team. 2026. The Isabelle Proof Assistant. https://isabelle.in.tum.de/
[39] The Lean Team. 2026. The Lean Proof Assistant. https://lean-lang.org/
[40] The Rocq Archive Team. 2016. Rocq Archive of Linear Temporal Logic. https://github.com/rocq-archive/ltl
[41] The Rocq Team. 2026. The Rocq Proof Assistant. https://rocq-prover.org/
[42] Amin Timany, Simon Oddershede Gregersen, Léo Stefanesco, Jonas Kastberg Hinrichsen, Léon Gondelman, Abel Nieto, and Lars Birkedal. 2024. Trillium: Higher-Order Concurrent and Distributed Separation Logic for Intensional Refinement. *Proc. ACM Program. Lang.* POPL (2024). doi:10.1145/3632851
[43] R. J. van Glabbeek. 1990. The linear time - branching time spectrum. In *CONCUR '90 Theories of Concurrency: Unification and Extension*, J. C. M. Baeten and J. W. Klop (Eds.).
[44] Eric Vin, Kyle A. Miller, and Daniel J. Fremont. 2025. LeanLTL: A Unifying Framework for Linear Temporal Logics in Lean (Short Paper) *(LIPIcs)*. doi:10.4230/LIPICS.ITP.2025.37
[45] J. Stanley Warford, David Vega, and Scott M. Staley. 2020. A Calculational Deductive System for Linear Temporal Logic. *ACM Comput. Surv.* (July 2020). doi:10.1145/3387109
[46] John Wiegley. 2026. Rocq Mechanisation of A Calculational Deductive System for Linear Temporal Logic. https://github.com/jwiegley/coq-cds4ltl
[47] Minchao Wu and Rajeev Goré. 2019. Verified Decision Procedures for Modal Logics. In *10th International Conference on Interactive Theorem Proving, ITP 2019, Portland, OR, USA, September 9-12, 2019 (LIPIcs)*. doi:10.4230/LIPICS.ITP.2019.31
[48] Naijun Zhan, Wen Tang, and Dimitar Guelev. 2026. A Complete Proof System for HyperLTL. In *Automated Reasoning*, Armin Biere, Carsten Lutz, and Sara Negri (Eds.). Cham, 377–395.

A The Need for a Post-Terminating State

Consider a definition of traces with no post-termination absorbing state. $\circ P$ must be defined for the terminated trace. Two immediate options present themselves.

Option 1 is to define it as $\circ P \equiv \text{True}$ for the terminated trace. This is problematic as $\Diamond P \equiv \exists n.\ \circ^n P \equiv \text{True}$ for all terminating traces; we can always pick an n > length of trace.

Option 2 is to define it as $\circ P \equiv \text{False}$ for the terminated trace. This is problematic as $\Box P \equiv \forall n.\ \circ^n P \equiv \text{False}$ for all terminating traces; we can always pick an n > length of trace.

Given an absorbing state we can define $\circ P\langle s \rangle$ for the terminated trace $\langle s \rangle$ as $P\bot$. For $\Diamond P$ this guarantees P to hold before termination, whenever P cannot possibly hold for $\bot$ (for example when it asserts anything about the state) For $\Box P$ this guarantees an infinite trace, whenever P cannot possibly hold for $\bot$ (for example when it asserts anything about the state). As covered in §2, the absorbing state is a built-in approach for handling the common requirement that the examined model (e.g., Kripke structure) has always at least one enabled transition per state [7, Chapter 3].

B Overview of Lemmas

An overview of the logic can be found in Fig. 2.

C Overview of Stenning example

We give a high-level model of the reliable transfer of an infinite sequence via the Stenning protocol [33] with a sliding window of 1, in an unreliable network, seen in Fig. 3. The model is a simplified version of the Stenning model in [17]. Notably, we omit the payload of the sequence and focus on proving that any sequence id is eventually reached, signifying the successful transfer of all id's before it.

The model has two actors; a sender A and a receiver B. Each actor can be in one of two states, ready to send $\top$ or receive $\bot$. A is indexed by a sequence id i, starting at 1. B is indexed by an acknowledgement id j, starting at 0. The state of the model is thus $((b_A, i), (b_B, j))$.

The labels of the model designates the acting actor $\alpha \in \{A, B\}$, and their action $\beta \in (\text{Send(i)}, \text{Recv(oi)})$. Here $oi \in nat^?$, where $\bot$ signifies a failed receive as a result of an empty buffer. We use notation such as $\alpha : \beta$ for the label pair.

For every iteration, A sends a message with the current sequence id to B, and then tries to receive. If the receive fails (nothing is in the node's inbound buffer), A loops. If the receive suceeds, A compares the acknowledgement id j to it's current sequence id i. If j is stale ($j < i$), A loops. If j is fresh ($j = i$), A increments i and loops.

For every iteration, the B tries to receive. If the receive fails, B loops. If the receive succeeds, B compares the sequence id i to it's current acknowledgement id j. If i is stale ($i < j$), B sends the current acknowledgement id j, and loops. If i is fresh ($i = j + 1$), B increments j, sends the new acknowledgement id j, and loops.

The complexity of the model arise from the liveness of B depending on A. Progress thus only happens whenever A succesfully receives a fresh id.

The property we wish to prove is:

$$\downarrow_s^{\text{snd}\cdot\text{fst}} 0 \vdash \Diamond \downarrow_s^{\text{snd}\cdot\text{fst}} i$$

That is, the sequence id of A eventually reaches any positive integer. The property holds under fairness assumptions on scheduling and network

$$\forall \alpha.\ \Box \Diamond \downarrow_l^{\text{fst}} \alpha \qquad \forall \alpha, i.\ (\Box \Diamond \downarrow_l (\alpha, \text{Send}(i))) \to (\Box \Diamond \downarrow_l (\neg\alpha, \text{Recv}(-))) \to (\Diamond \downarrow_l (\neg\alpha, \text{Recv}(\textbf{some}\ i)))$$

Syntax:

$$P, Q \in \mathsf{tProp} \quad ::= \mathsf{True} \mid \mathsf{False} \mid P \wedge Q \mid P \vee Q \mid \forall x.\, P \mid \exists x.\, P \mid x = y \mid \downarrow \phi \mid \circ P$$

Derived Constructions:

$$\Box P \triangleq \forall n.\ \circ^n P$$
$$P \cup Q \triangleq \mu X.\, Q \vee (P \wedge \circ X)$$
$$\Diamond P \triangleq \mathsf{True} \cup P$$

Now (↓) Rules

$$\downarrow\text{-MONO}\ \frac{\forall osl.\ \phi\ osl \Rrightarrow \psi\ osl}{\downarrow \phi \vdash \downarrow \psi} \qquad \downarrow\!-\forall\ \forall x.\ \downarrow \phi \dashv\vdash \downarrow(\widehat{\forall}\, x.\, \phi) \qquad \downarrow\!-\exists\ \exists x.\ \downarrow \phi \dashv\vdash \downarrow(\widehat{\exists}\, x.\, \phi)$$

$$\downarrow\text{-NOT}\ \neg \downarrow \phi \vdash \downarrow \widehat{\neg}\, \phi \qquad \downarrow\text{-PURE}\ \downarrow \phi \vdash \exists osl.\ \phi\ osl$$

$$\downarrow\text{-AND}\ \downarrow \phi \wedge \downarrow \psi \dashv\vdash \downarrow(\phi\, \widehat{\wedge}\, \psi) \qquad \downarrow\text{-OR}\ \downarrow \phi \vee \downarrow \psi \dashv\vdash \downarrow(\phi\, \widehat{\vee}\, \psi)$$

$$\downarrow\text{-AGREE}\ \downarrow^f x_1 \wedge \downarrow^f x_2 \vdash x_1 = x_2 \qquad \downarrow\text{-UNPROJ}\ \downarrow^f x \vdash \exists s.\, fs = x \wedge \downarrow s$$

Next (○) Rules:

$$\circ\text{-TAUT}\ \frac{\vdash P}{\vdash \circ P} \qquad \circ\text{-MONO}\ \circ(P \to Q) \vdash \circ P \to \circ Q \qquad \circ\text{-NOT}\ \circ \neg P \dashv\vdash \neg \circ P$$

$$\circ\!-\exists\text{-COMM-R}\ \circ \exists x.\, P \vdash \exists x.\, \circ P \qquad \circ\!-\forall\text{-COMM-R}\ \forall x.\, \circ P \vdash \circ \forall x.\, P \qquad \circ\text{-PURE}\ \circ p \dashv\vdash p$$

$$\circ\text{-FALSE}\ \circ \mathsf{False} \dashv\vdash \mathsf{False} \qquad \circ\text{-TRUE}\ \mathsf{True} \dashv\vdash \circ \mathsf{True}$$

$$\circ\text{-AND-COMM}\ \circ(P \wedge Q) \dashv\vdash \circ P \wedge \circ Q \qquad \circ\text{-OR-COMM}\ \circ(P \vee Q) \dashv\vdash \circ P \vee \circ Q$$

Always (□) Rules:

$$\Box\text{-TAUT}\ \frac{\vdash P}{\vdash \Box P} \qquad \Box\text{-MONO}\ \Box(P \vdash Q) \vdash \Box P \to \Box Q \qquad \Box\text{-COIND}\ \Box(P \to (Q \wedge (P \vee \Box Q))) \vdash P \to \Box Q$$

$$\Box\text{-INTRO}\ \Box(P \to \circ P) \vdash P \to \Box P$$

$$\Box\text{-UNFOLD}\ \Box P \dashv\vdash P \wedge \circ \Box P \qquad \Box\text{-AND-COMM}\ \Box(P \wedge Q) \dashv\vdash \Box P \wedge \Box Q$$

$$\Box\text{-}\circ\text{-COMM}\ \Box \circ P \dashv\vdash \circ \Box P \qquad \Box\text{-IDEMP}\ \Box P \vdash \Box \Box P$$

Until ($P \cup Q \triangleq \mu X.\, Q \vee (P \wedge \circ X)$) and Eventually ($\Diamond P \equiv \mathbf{True} \cup P$) Rules:

$$\cup\text{-UNFOLD}\ P \cup Q \dashv\vdash Q \vee (P \wedge \circ(P \cup Q)) \qquad \cup\text{-IND}\ \Box(Q \vee (P \wedge \circ(P \cup Q) \wedge \circ R) \to R) \vdash P \cup Q \to R$$

$$\cup\text{-FALSE}\ P \cup \mathsf{False} \vdash \mathsf{False}$$

$$\cup\text{-MONO}\ \Box(P_1 \to P_2) \wedge \Box(Q_1 \to Q_2) \vdash P_1 \cup Q_1 \to P_2 \cup Q_2 \qquad \cup\text{-IDEMP}\ P \cup Q \dashv\vdash P \cup (P \cup Q)$$

$$\cup\text{-AND-L}\ P \cup (Q_1 \wedge Q_2) \vdash (P \cup Q_1) \wedge (P \cup Q_2)$$

$$\cup\text{-AND-R}\ (P_1 \cup Q_1) \wedge (P_2 \cup Q_2) \vdash (P_1 \wedge P_2) \cup ((Q_1 \wedge P_2 \cup Q_2) \vee (P_1 \cup Q_1 \wedge Q_2)) \qquad \cup\text{-OR}\ (P \cup Q_1) \vee (P \cup Q_2) \vdash (P \cup Q_1) \vee (P \cup Q_2)$$

Figure 2. Overview of rules. We use $\widehat{_}$ as pointwise lifting.

Either actor is always eventually scheduled, and any message sent infinitely often from A to B (and vice versa), is received by B (resp. A), if they attempt a receive infinitely often.

Finally, we include a safety assumption that the two actors do not get out of sync

$$\exists i.\ \downarrow_s^{\mathsf{fst}\cdot\mathsf{fst}} i \wedge (\downarrow_s^{\mathsf{fst}\cdot\mathsf{snd}} i \vee \downarrow_s^{\mathsf{fst}\cdot\mathsf{snd}} i - 1)$$

That is, B can never be ahead of A, and it can never be more than 1 behind. The assumption is necessary as we do not model the network, for brevity sake, and follows from any sensible model of a network.

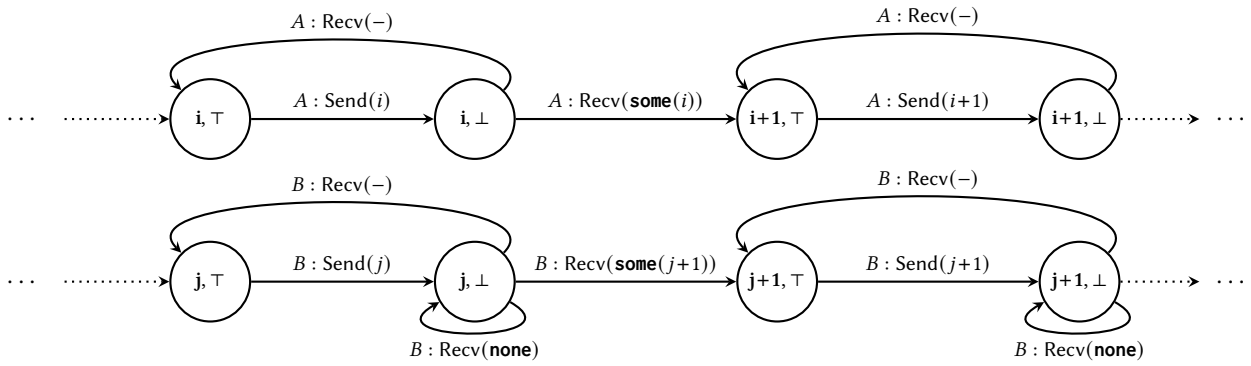


Figure 3. High-level model of the Stenning protocol