Compositional reasoning using intervals and time reversal

Ben Moszkowski

Received: 19 April 2012 / Final revision received: 26 April 2013 / Accepted: 29 April 2013 / Available online: 5 June 2013

Abstract Interval Temporal Logic (ITL) is an established formalism for reasoning about time periods. We investigate some simple kinds of ITL formulas which have application to compositional reasoning and furthermore are closed under conjunction and the conventional temporal operator known both as "box" and "always". Such closures help us modularly construct formulas from simple building blocks in a way which preserves useful compositional properties. The most important class considered here is called the 2-to-1 formulas. They offer an attractive framework for analysing sequential composition in ITL and provide the formal basis for most of the subsequent presentation. A key contribution of this work concerns a useful and apparently new and quite elementary mathematical theorem that 2-to-1 formulas are closed under "box". We also use a natural form of time symmetry with 2-to-1 formulas. This extends known facts about such formulas by looking at them in reverse. An important example of this involves showing that 2-to-1 formulas are also closed under a variant of "box" for prefix subintervals rather than suffix ones. We then apply the compositional formulas obtained with time symmetry to analyse concurrent behaviour involving mutual exclusion in both Peterson's algorithm and a new and more abstract one. At present, our study of mutual exclusion mainly serves as a kind of experimental "proof of concept" and research tool to develop and illustrate some of the logical framework's promising features. We also discuss how time symmetry sometimes assists in reducing reasoning in ITL to conventional linear-time temporal logic.

Keywords Interval Temporal Logic · compositional reasoning · formal verification · time reversal · symmetry · mutual exclusion · Peterson's algorithm

Software Technology Research Laboratory
De Montfort University
Leicester, UK
E-mail: benm@dmu.ac.uk

1 Introduction

Intervals and discrete linear state sequences offer a compellingly natural and flexible way to model computational processes involving hardware or software. **Interval Temporal Logic (ITL)** [47] is an established formalism for reasoning about such phenomena. It has operators for sequentially combining formulas. For example, if A and B are formulas, so are $A^\frown B$ (*"chop"*) and $A^\star$ (*"chop-star"*). These are somewhat analogous to the concatenation and Kleene star operators for regular languages and expressions. ITL can express some imperative programming constructs (e.g., while-loops) and has executable subsets [47].

We first summarise the primary contributions of this presentation and then discuss them in more detail:

- Several classes of compositional ITL formulas which all share the important property that they are closed under conjunction and the conventional temporal operator "always" ($\Box$).
- Various syntactic and semantic applications of time symmetry to such formulas.
- Some useful techniques for compositionally manipulating a number of suitable sequential and parallel combinations of the formulas with others.
- A detailed application of these ideas to mutual exclusion, including the analysis of a novel abstract algorithm as well as Peterson's well-known concrete one [64].
- All of results are accompanied by rigorous, detailed mathematical theorems, lemmas and associated proofs, which are moreover themselves a quite indispensable part in the development of the framework.

Our main contribution concerns a novel categorisation and mathematical analysis of various simple classes of compositional formulas in **Propositional ITL (PITL)** [54, 55, 57] which are closed under conjunction and the conventional temporal operator "always" ($\Box$). The main class we consider consists of what we call **2-to-1 formulas** (which are formally defined in Sect. 4). Briefly, a PITL formula A is defined to be 2-to-1 if the implication $(A;A) \supset A$ if valid, where ";" is a second variant of chop. So if two portions of a system both ensure such a formula's behaviour, then their sequential composition is guaranteed to as well. The 2-to-1 formulas play a *quite central role* in almost all of the techniques presented here. For example, we can show that for the propositional variables p and q, the conventional temporal logic formula $p \supset \Diamond q$ (*"if p is true in the initial state, then q is true in some state"*) is 2-to-1. Our new closure theorem immediately guarantees that the liveness formula $\Box(p \supset \Diamond q)$ (*"whenever p is true, q is true then or later"*) is 2-to-1 as well. Such a formula is suitable for **forward analysis** from a state satisfying p to one satisfying q. The many compositional properties we identify and rigorously prove clearly show that further systematic research about 2-to-1 formulas and other such classes of formulas closed under conjunction and $\Box$, including the relationship between them, is compelling required.

We also propose here a second significant research contribution which exploits the **symmetry of finite linear time** to transform 2-to-1 formulas for **forward analysis** such as $\Box(p \supset \Diamond q)$ into others for **backward analysis** from a state to its predecessors (as described in Sects. 5 and 6). This involves a **two-stage** approach. In the *first stage*, our mathematical framework takes some suitable 2-to-1 formulas and views them *in reverse* in finite time to obtain more formulas which are 2-to-1 in finite time. In the *second stage*, these formulas are then shown to even be 2-to-1 in infinite time. The process of transforming formulas demonstrates the significance of both *syntactic* and *semantic* forms of time symmetry.

The relationship between our use of time symmetry and some relevant earlier work using it is primarily discussed later in Sect. 16.1. We postpone a comparison until then in order that readers will have a better understanding of our framework.

The approach here based on 2-to-1 formulas and time symmetry further develops our ITL-based compositional techniques described in [48–51] since, for example, it helps to systematically obtain additional properties for sequential composition. Moreover, a number of results about 2-to-1 formulas and time symmetry are also applicable to the first-order version of ITL used in our earlier work, but we do not delve into this further.

We will consider a variety of relevant properties and other related categories of PITL formulas for compositional reasoning about sequential and parallel behaviour. The main techniques here can be summarised as ***I****ntroduction*, ***S****equential combining*, ***E****xtension leftward or rightward*, ***P****arallel combining* and ***I****teration* (see Sect. 4.1). This is abbreviated with the shorthand **ISEPI**.

The 2-to-1 formulas and time symmetry are then applied (in Sects. 11–13) to showing by means of backward analysis the correctness of a new high-level abstract algorithm for mutual exclusion as well as the much studied one of Peterson [64]. It is first of all quite important to emphasise that the study of mutual exclusion led us in the first place to the 2-to-1 formulas and time symmetry. However, at present, our study of mutual exclusion mainly serves as a kind of experimental "proof of concept". *It has significantly influenced the development of virtually all aspects of the presentation here and moreover helps to illustrate some of the logical framework's promising features.* Nevertheless, we do not claim that it is sufficiently mature for practical deployment. Readers may indeed experience some difficulties with the intuition behind some formulas. *Therefore, the material on mutual exclusion must be regarded,* at least at present*, as being primarily a powerful* research tool *for the intriguing compositional framework's evolving theory rather than a distinct and independent application on its own. As such, it is for the moment indispensable for understanding the work.*

Our presentation also shows (in Sect. 14) how time symmetry can assist in reducing satisfiability of suitable 2-to-1 formulas and some other PITL formulas to *finite-time* satisfiability of formulas in lower-level point-based temporal logic. This might help provide a way to extend the scope of some algorithms, implemented software tools and mathematical techniques for conventional temporal logic to eventually include suitable subsets of PITL involving 2-to-1 formulas as well.

The proofs given about PITL formulas are semantically based and so do not use a formal axiom system. However, an analysis could in principle include deductions in our complete axiom system for PITL with finite time [54] (see also Bowman and Thompson [10]) and our newer one with infinite time [57].

Readers new to interval-based reasoning will find the approach quite different from those using point-based temporal logics. This applies even to our use of a conventional temporal logic formula such as $\Box(p \supset \Diamond q)$, when, for example, we explain why it is 2-to-1 or use time symmetry on it. In fact, we believe that even readers having experience with intervals will find our presentation quite novel. They should however keep in mind that time symmetry can be rather subtle. It requires an investment of patience and effort to be understood.

For the particular benefit of readers unfamiliar with ITL, we now briefly mention some recent publications by others which reflect current topics where ITL is being applied. They arguably contribute to making a case for the study of ITL's mathematical foundations, which naturally include such issues as compositionality and time symmetry.

The KIV interactive theorem prover [68] has for a number of years included a slightly extended version of ITL for interactive theorem proving via symbolic execution both by itself (e.g., for concurrent algorithms and lock-free techniques [6, 7]) and also as a backend notation which supports Statecharts [78] and UML [2]. The concluding remarks of [7] note the following advantages of ITL:

> Our ITL variant supports classic temporal logic operators as well as program operators.
> The interactive verifier KIV allows us to directly verify parallel programs in a rich programming language using the intuitive proof principle of symbolic execution. An additional translation to a special normal form (as e.g. in TLA [Temporal Logic of Actions [38]]) using explicit program counters is not necessary.

The *Duration Calculus* (DC) of Zhou, Hoare and Ravn [85] extends ITL to real-time. Zhou and Hansen [84] give a comprehensive presentation of various aspects of DC and its application. They include a large bibliography of literature on DC. Olderog and Dierks' recent textbook [59] uses DC as the formal logic in a framework for seamless design flow from specification to verified implementation. This approach also includes timed automata and automata for programmable logic controllers (PLC-automata).

Duan and his group have been investigating the theory and application of *Projection Temporal Logic*, an ITL extension with operators for temporal granularities and framing [13–17] (our later Sect. 13.1 gives an explanation of framing). Some of their recent work on applications such as the specification and verification of asynchronous communication is described in [44] and [83].

Our presentation is a revised and greatly extended version of the earlier one by us in [56] that readers might benefit from because of its much briefer and more superficial format. The focus here differs from that in [56] by concentrating more on the general compositional issues. This is because we have subsequently come to realise that the theory of 2-to-1 formulas and related classes is much more central than its application to time symmetry, which is nevertheless quite intriguing. As a consequence, we have now added various definitions, explanations and other material. The topics we consider have many interesting aspects of relevance to compositional reasoning with and without time symmetry.

More recently, in [58] we present in a concise manner new techniques for systematically and incrementally elucidating connections between 2-to-1 formulas and some associated compositional classes. These are a direct outcome of the research described here and can serve as a quick introduction to the mathematics of such classes. However, the compositional ISEPI techniques, time symmetry and applications to mutual exclusion are not discussed in [58].

Here is our presentation's structure:

- Section 2 overviews PITL.
- Section 3 presents some important point-based subsets of PITL used later on in our analysis.

- Section 4 introduces 2-to-1 formulas, presents various kinds of them and proves that they are closed under conjunction, the temporal operator $\Box$ ("always") as well as the time-wise symmetric operator $\boxed{\text{f}}$, which concerns finite prefix subintervals instead of suffix subintervals. As we discuss there, 2-to-1 formulas can be used for reasoning about various safety and liveness properties. A categorisation is given of general compositional ISEPI techniques for ***I**ntroduction*, ***S**equential combining*, ***E**xtension leftward or rightward*, ***P**arallel combining* and ***I**teration*.
- Section 5 starts our discussion about the application to 2-to-1 formulas of both time symmetry and reductions from infinite time to finite time. It therefore shows how to relate some of the time reversed formulas to other semantically comparable ones expressed in a version of conventional **Propositional Linear-Time Temporal Logic** (PTL) with past time because this is much better known than PITL.
- Section 6 uses time symmetry to obtain a versatile class of 2-to-1 formulas for *backward analysis* from other 2-to-1 formulas for *forward analysis*. Such formulas are extensively used in all subsequent sections.
- Sections 7–10 primarily concern versions of the various ISEPI techniques suitable for compositionally combining 2-to-1 formulas for backward analysis:
  - Section 7 provides ways to compositionally *introduce* such 2-to-1 formulas.
  - Section 8 deals with a compositional technique for sequentially *extending* the scope of the 2-to-1 formulas for backward analysis from an interval's prefix subinterval to the entire interval.
  - Section 9 concerns the *parallel* combining of such 2-to-1 formulas.
  - Section 10 presents techniques for compositional reasoning involving the sequential *iteration* of these 2-to-1 formulas.
- Sections 11–13 concern mutual exclusion:
  - Section 11 looks at an abstract mutual exclusion algorithm. The section applies to the algorithm the results from the previous sections concerning 2-to-1 formulas for backward analysis and ISEPI techniques for sequential and parallel composition of such formulas.
  - Section 12 considers in more detail an individual process in the abstract mutual exclusion algorithm and also its relation to a process in Peterson's algorithm.
  - Section 13 analyses Peterson's algorithm. This is done by formally relating it to the abstract algorithm.
- Section 14 examines further reductions using time symmetry to transform some PITL formulas into ones in conventional point-based temporal logic.
- Section 15 discusses various pertinent issues.
- Section 16 surveys related work.

Our view is that the separation of the underlying mathematics from the subsequent application to mutual exclusion helps make the foundational theoretical aspects of the framework clearer. One could indeed even take this a stage further and argue that in principle the more purely theoretical material on compositionality in Sects. 2–10 could be studied somewhat independently of its application to mutual exclusion in Sects. 11–13. However, in practice, our investigation of the abstract theory and its application to mutual exclusion have been done simultaneously with much cross-fertilisation involving experimentation and trial and error.

As a result, the theory and application of 2-to-1 formulas for backward analysis seem quite interrelated. Indeed, it appears virtually impossible for us to have developed either of them in isolation. We therefore believe they are best understood and appreciated when studied together in a way which offers a more complete picture of the approach in its current form.

The evolution of this work is moreover inextricably connected with the rigorous construction of many theorems, lemmas and associated proofs. This also seems to be an inseparable and quite invaluable and essential part of the exploration process. What we present here gives a picture of the current state of the approach. It continues to progress as we gain more knowledge about the remarkable and extensive mathematical terrain of 2-to-1 formulas for forward and backward analysis.

2 Propositional Interval Temporal Logic

We now describe the version of (quantifier-free) PITL used here. More on ITL and PITL can be found in [47, 54, 55] (see also Kröger and Merz [37], Fisher [19] and the ITL web pages [32]).

Below is the syntax of PITL formulas in BNF, where p is any propositional variable:

$$A ::= \mathit{true} \mid p \mid \neg A \mid A \vee A \mid \mathit{skip} \mid A^\frown A \mid A^\star . \qquad (1)$$

The last two constructs are called **chop** and **chop-star**, respectively. The boolean operators *false*, $A \wedge B$, $A \supset B$ (*implies*) and $A \equiv B$ (*equivalence*) are defined as usual. We refer to $A^\frown B$ as **strong chop** and likewise refer to $A^\star$ as **strong chop-star**. Weak versions are discussed shortly when we present some derived operators.

Time within PITL is modelled by discrete, linear *intervals*. An **interval** $\boldsymbol{\sigma}$ is any finite or ω-sequence of one or more **states** $\sigma_0, \sigma_1, \ldots$ (which are not necessarily distinct from one another). Each σ_i maps every propositional variable p to *true* or *false*. This mapping is denoted as $\sigma_i(p)$. Let $\boldsymbol{\Sigma}$ denote the set of all states. An interval σ has **interval length** $|\sigma| \geq 0$, which, if σ is finite, is the number of σ's states minus 1 and otherwise ω. So if σ is finite, it has states $\sigma_0, \ldots, \sigma_{|\sigma|}$. If the same state occurs twice in σ, it is counted twice for determining σ's interval length. Let $\boldsymbol{\Sigma^+}$ denote the set of finite intervals and $\boldsymbol{\Sigma^\omega}$ denote the set of infinite ones. The (standard) version of PITL used here with state-based propositional variables is called **local PITL**. A **subinterval** of σ is any interval which is a *contiguous* subsequence of σ's states. This includes σ itself.

The notation $\sigma \models A$, defined shortly by induction on A's syntax, denotes that interval σ **satisfies** formula A. Moreover, A is **valid**, denoted $\models A$, if all intervals satisfy it.

Below are the semantics of the first five PITL constructs in (1):

- True: $\sigma \models \mathit{true}$ trivially holds for any σ.
- A variable p: $\sigma \models p$ iff $\sigma_0(p){=}\mathit{true}$ (initially p).
- Negation: $\sigma \models \neg A$ iff $\sigma \not\models A$.
- Disjunction: $\sigma \models A \vee B$ iff $\sigma \models A$ or $\sigma \models B$.
- Skip: $\sigma \models \mathit{skip}$ iff σ has exactly two states (i.e., $|\sigma| = 1$).

Note that an interval σ satisfies *skip* even if σ's two states are identical to each other. For natural numbers i, j with $0 \leq i \leq j \leq |\sigma|$, let $\sigma_{i:j}$ be the finite subinterval $\sigma_i \ldots \sigma_j$ (i.e., $j-i+1$ states). Define $\sigma_{i\uparrow}$ to be σ's suffix subinterval from state σ_i.

Below are semantics for strong chop and chop-star:

- $A^\frown B$: $\sigma \models A^\frown B$ iff for some natural number i: $0 \le i \le |\sigma|$, both $\sigma_{0:i} \models A$ and $\sigma_{i\uparrow} \models B$.
  Note that in the case where $|\sigma| = \omega$, we actually have $i < |\sigma|$.
- $A^\star$: $\sigma \models A^\star$ iff one of the following holds:
  (1) The interval σ has only one state.
  (2) σ is finite and either itself satisfies A or can be split into a finite number of (finite-length) subintervals which share end-states (like chop) and all satisfy A.
  (3) $|\sigma| = \omega$ and σ can be split into ω finite-length intervals sharing end-states (like chop) and each satisfying A.
  Case (3) is called **chop-omega** and denoted as A^ω.

We depict below the behaviour of variable p in a sample 5-state interval σ and denote *true* and *false* by `t` and `f`.

	σ_0	σ_1	σ_2	σ_3	σ_4
p	`t`	`f`	`t`	`f`	`t`

This interval satisfies the following formulas:

$$p \qquad skip^\frown\neg p \qquad p \wedge (true^\frown\neg p) \qquad (p \wedge (skip^\frown skip))^\star.$$

For instance, the formula $skip^\frown\neg p$ is true because $\sigma_0\sigma_1$ satisfies *skip* and $\sigma_1 \ldots \sigma_4$ satisfies $\neg p$ since $\sigma_1(p) = false$. The fourth formula is true because the interval σ's three-state subintervals $\sigma_0\sigma_1\sigma_2$ and $\sigma_2\sigma_3\sigma_4$ both satisfy $p \wedge (skip^\frown skip)$. The interval σ does not satisfy the formulas below:

$$\neg p \qquad skip^\frown p \qquad true^\frown(\neg p \wedge \neg(true^\frown p)).$$

Table 1 shows useful derived PITL operators, including *empty* for one-state intervals and the **weak chop** construct $A;B$ which can ignore B in an infinite interval satisfying A. We derive here ITL's conventional **weak chop-star** A^* from the strong version, although the two are interderivable. In an infinite interval, strong chop-star requires an infinite number of iterations each of finite length, whereas weak chop-star also permits a finite number of iterations with the last having infinite length. The strong variants of chop and chop-star are taken as primitives here to simplify some of the reasoning about time symmetry. However, we extensively use the weak versions for reasoning about possibly nonterminating parts of programs.

We discuss in Sect. 15.4 the reason for our use of the term "empty" to describe one-state intervals even though in language theory it refers the unique empty word with no letters at all.

Let w and w' denote **state formulas** without any temporal operators.

Table 2 contains several sample PITL formulas which are valid. For example, the formula $(w \wedge A) \equiv (empty \wedge w);A$ can be understood as stating that an interval satisfies state formula w and PITL formula A iff the first state of the interval satisfies w and the interval satisfies A. Here the first state is equally regarded as being a one-state interval in its own right. The valid equivalence $(\boxed{\mathrm{f}}\boxed{\mathrm{f}}A) \equiv \boxed{\mathrm{f}}A$ uses $\boxed{\mathrm{f}}A$ to test the formula A in finite subintervals and uses $\boxed{\mathrm{f}}\boxed{\mathrm{f}}A$ to test A in these

$\bigcirc A$	$\widehat{=}$	$skip^\frown A$	Next
$\Diamond A$	$\widehat{=}$	$true^\frown A$	Eventually
$\Box A$	$\widehat{=}$	$\neg\Diamond\neg A$	Henceforth (Always)
$more$	$\widehat{=}$	$\bigcirc\, true$	More than one state
$empty$	$\widehat{=}$	$\neg more$	Only one state
$finite$	$\widehat{=}$	$\Diamond empty$	Finite interval
inf	$\widehat{=}$	$\neg finite$	Infinite interval
$fin\ A$	$\widehat{=}$	$\Box(empty \supset A)$	Weak test of final state
$stable\ A$	$\widehat{=}$	$\Box(more \supset (A \equiv \bigcirc A))$	Stability
$\Diamond_f A$	$\widehat{=}$	$A^\frown true$	Some initial finite subinterval
$\boxed{f} A$	$\widehat{=}$	$\neg\Diamond_f\neg A$	All initial finite subintervals
$A;B$	$\widehat{=}$	$(A^\frown B) \ \vee\ (inf \wedge A)$	Weak chop
$\Diamond_i A$	$\widehat{=}$	$A;true$	Some initial subinterval (even infinite)
$\boxed{i} A$	$\widehat{=}$	$\neg\Diamond_i\neg A$	All initial subintervals (even infinite)
A^*	$\widehat{=}$	$A^\star \vee \big(A^{\star\frown}(inf \wedge A)\big)$	Conventional weak chop-star
A^+	$\widehat{=}$	$A;A^*$	One or more iterations
A^ω	$\widehat{=}$	$inf \wedge A^\star$	Chop-omega
$A \leftarrow B$	$\widehat{=}$	$finite \supset ((fin\ A) \equiv B)$	Temporal assignment
$A \mathrel{<\!\!\sim} B$	$\widehat{=}$	$A \leftarrow B \ \wedge\ (stable\ A; skip)$	Padded temporal assignment

Table 1 Some useful derived PITL operators

$$(finite \wedge \boxed{f}A) \supset A \qquad skip^\star \qquad inf \equiv \Box more$$

$$(w \wedge A); B \ \equiv\ w \wedge (A; B) \qquad A \ \equiv\ (empty; A)$$

$$w \wedge A \ \equiv\ (empty \wedge w); A \qquad finite \supset (A^* \equiv A^\star)$$

$$\boxed{f}(A \wedge B) \equiv (\boxed{f}A \wedge \boxed{f}B) \qquad (\Box\boxed{f}A) \equiv (\boxed{f}\Box A)$$

$$(\boxed{f}\boxed{f}A) \ \equiv\ \boxed{f}A \qquad A^* \ \equiv\ (empty \vee A^+)$$

$$\Diamond_f A \wedge \Diamond_f B \ \equiv\ \Diamond_f(\Diamond_f A \wedge \Diamond_f B)$$

Table 2 Sample valid PITL formulas

finite subintervals' own finite subintervals. The equivalence is valid because the set of finite subintervals contained within an interval's finite subintervals is exactly the same as the set of the interval's own finite subintervals. This is related by time symmetry to the equivalence $(\Box\Box A) \equiv \Box A$ which concerns suffix subintervals and is found even in conventional temporal logic. The last formula $(\Diamond_f A \wedge \Diamond_f B) \equiv \Diamond_f(\Diamond_f A \wedge \Diamond_f B)$ states that two formulas A and B are each true in finite prefix subintervals of an interval exactly if the conjunction $\Diamond_f A \wedge \Diamond_f B$ is true in some finite prefix subinterval (e.g., the larger of the two satisfying A and B, respectively).

3 Some important point-based subsets of PITL

We now present some point-based subsets of PITL which have been found useful in our previous work [55, 57] and come in handy later.

3.1 Subset of PITL with only skip, next and diamond

Let **PTL** denote the subset of PITL formulas in conventional **Propositional Linear-Time Temporal Logic** with just the (derived) temporal operators $\bigcirc$ and $\Diamond$ in Table 1. We use X, X' and Y for PTL formulas. Various useful compositional safety and liveness properties can be expressed in PTL. For instance, we already presented the sample PTL formulas $p \supset \Diamond q$ and $\Box(p \supset \Diamond q)$ in the introduction in Sect. 1. Note that the PITL primitive construct *skip* can be derived in PTL as $\bigcirc \neg \bigcirc \mathit{true}$ (the same as $\bigcirc\, \mathit{empty}$), so we can regard PTL as containing it as well.

We will use PTL here since it can express various compositional formulas. Furthermore, it has much lower computational complexity and better tool support than full PITL, which has nonelementary complexity (a theorem by Kozen described in our own joint work with Halpern [45] (reproduced in [54])). Therefore, Sect. 14 describes some potential transformations using time symmetry from PITL to a slightly enhanced version of PTL with an until operator defined shortly in Sect. 3.3.

3.2 Subset of PTL with just unnested next operators

We extensively use an important subset of PTL involving the operator $\bigcirc$:

Definition 3.1 (Next Logic) The set of PTL formulas in which the only primitive temporal operator is $\bigcirc$ is called *Next Logic (NL)*. The subset of NL in which no $\bigcirc$ is nested within another $\bigcirc$ is denoted as NL^1.

For example, the NL formula $p \wedge \bigcirc q$ is in NL^1, but the NL formula $p \wedge \bigcirc(q \vee \bigcirc p)$ is not.

The variable T denotes formulas in NL^1.

All state formulas (e.g., $p \wedge \neg q$) are in NL^1 because they contain no temporal operators. The important derived PITL constructs *more* and *empty* already defined in Table 1 are also in NL^1. Unlike state formulas, *more* and *empty* can detect whether or not an interval has just one state. However, the primitive construct *skip*, which tests for exactly two states, cannot be expressed in NL^1 because it requires one $\bigcirc$ within another: $\bigcirc \neg \bigcirc \mathit{true}$ (the same as $\bigcirc\, \mathit{empty}$).

In order to further illustrate the nature of NL^1 expressiveness, we list below some more properties which NL^1 formulas *cannot* express, together with PTL formulas not in NL^1 which do capture these properties:

Property	Corresponding formula not in NL^1
The formula $p \wedge \neg q$ is true in the third state	$\bigcirc \bigcirc (p \wedge \neg q)$
The interval has more than two states	$\bigcirc \bigcirc \mathit{true}$ (same as $\bigcirc\, \mathit{more}$)
The interval has exactly two states	$\bigcirc \neg \bigcirc \mathit{true}$ (same as $\bigcirc\, \mathit{empty}$)

The NL^1 formulas play a significant role in the theory of PITL. We therefore strongly encourage readers seriously interested getting a better understanding of ITL to study our presentation in [55], where we systematically describe some natural applications of NL^1 to relating point-based and interval-based temporal logic. Our new complete axiom system for PITL with infinite time [57] likewise makes extensive use of NL^1 formulas and therefore shows how they can be profitably employed.

3.3 PTL with until operator

Our presentation also makes use of a PTL variant called here $\mathbf{PTL^u}$. It has a somewhat restricted *strong* version of the standard temporal operator *until* which is derivable in PITL:

$$T \; \mathit{until} \; A \quad \widehat{=} \quad (\mathit{skip} \wedge T)^{\star\frown} A.$$

Only NL^1 formulas are permitted on our restricted *until* 's left side, as the definition indicates by the use of T. This is because if the left operand is not in NL^1, the restricted *until* will not work properly. For example, the formula $(\bigcirc\bigcirc p)\;\mathit{until}\;q$ actually reduces to *false*. Now $\mathrm{PTL^U}$ is more expressive than PTL (e.g., see [37]), but reducible to it using auxiliary variables to mimic *until* . For example, when considering the satisfiability of the formula $p \wedge \bigcirc(p\;\mathit{until}\;q) \wedge \neg(p\;\mathit{until}\;q)$, we can transform it into the formula below with an extra auxiliary variable r:

$$p \;\wedge\; \bigcirc r \;\wedge\; \neg r \;\wedge\; \Box\big(r \;\equiv\; q \vee (p \wedge \bigcirc r)\big) \;\wedge\; \Box(r \supset \Diamond q).$$

We make extensive use of $\mathrm{PTL^U}$ in our recent axiomatic completeness proof for PITL with infinite time [57]. Section 15.3 later shows an alternative way to derive *until* without chop-star.

Section 5 (particularly regarding formula (13)) and Sect. 14 include reductions from PITL to $\mathrm{PTL^U}$. We use $\mathrm{PTL^U}$ because formulas in it can be more expressive than in PTL. Nevertheless, they can be readily transformed to PTL, although, as we noted already, this necessitates the introduction of some auxiliary variables. When compared with PITL, the differences between PTL and $\mathrm{PTL^U}$ are certainly quite small.

4 2-to-1 formulas

We now discuss in more detail the 2-to-1 formulas, which were already briefly considered in Sect. 1. These are the main class of formulas closed under conjunction and the temporal operator $\Box$ in our presentation here. Such closure properties help to modularly construct formulas from simple building blocks in a way guaranteed to ensure that the results preserve some useful compositional properties. Many of the properties of 2-to-1 formulas which we consider concern sequential composition. However, parallel composition of formulas is not neglected either.

In this section we focus on formally defining 2-to-1 formulas and studying some of their fundamental theoretical properties. The presentation mostly looks at the relatively abstract mathematics of compositional reasoning rather than any particular application of the formulas. We believe that the issues explored at such a level of abstraction are themselves an important contribution to work in the area. Nevertheless, this material provides a solid and practical basis when the formulas are extensively used later on with time symmetry in Sects. 5 and 6 and in our experimental analysis of mutual exclusion in Sects. 11–13.

Definition 4.1 (2-to-1 Formulas) Any PITL formula A for which the implication $(A;A) \supset A$ is valid is called a **2-to-1 formula**.

For example, the state formulas *true* and p are 2-to-1. In the first case, *true* is trivially true for any interval, so the implication $(\mathit{true};\mathit{true}) \supset \mathit{true}$ is valid. Here is a proof for the case of p:

Proof (p is 2-to-1) Suppose that an interval σ satisfies $p;p$. We can readily show from the PITL semantics of weak chop (see Table 1) that σ has a prefix subinterval σ' (perhaps σ itself) which satisfies p. Hence, the first state of σ' also satisfies p. Now the first states of σ and σ' are identical, so σ itself satisfies p and consequently also the implication $(p;p) \supset p$. Therefore, every interval satisfies this implication, so it is indeed valid. Consequently, p is 2-to-1. □

In contrast to *true* and p, the formula *skip* is not 2-to-1 and perhaps even the simplest example of this. Observe that the chop formula *skip*; *skip* is satisfied solely by intervals with precisely 3 states, whereas *skip* checks that an interval has exactly 2 states. Hence, the implication $(\mathit{skip};\mathit{skip}) \supset \mathit{skip}$ is not satisfied by 3-state intervals and is consequently not valid.

As the next Lemma 4.2 shows, it does not actually matter whether we use weak or strong chop to define 2-to-1 formulas:

Lemma 4.2 *The following is an alternative way to characterise 2-to-1 formulas using strong chop instead of weak chop:*

$$\models \quad (A^\frown A) \quad \supset \quad A.$$

Proof The chain of equivalences below demonstrates that the two implications $(A;A) \supset A$ and $(A^\frown A) \supset A$ are in fact semantically indistinguishable:

$$\begin{array}{ll} (A;A) \supset A & \\ \quad\equiv & \big((A^\frown A) \vee (\mathit{inf} \wedge A)\big) \supset A \\ \quad\equiv & \big((A^\frown A) \supset A\big) \wedge \big((\mathit{inf} \wedge A) \supset A\big) \\ \quad\equiv & \big((A^\frown A) \supset A\big) \wedge \mathit{true} \\ \quad\equiv & (A^\frown A) \supset A. \end{array}$$

The first equivalence simply re-expresses $A;A$ using the definition of weak chop in Table 1 in Sect. 2. The third equivalence holds since the implication $(\mathit{inf} \wedge A) \supset A$ is valid for any PITL formula A and can therefore be replaced by the formula *true*. Therefore the two characterisations of 2-to-1 formulas are semantically equivalent. □

We prefer to define 2-to-1 formulas using weak chop because it better copes with nontermination in applications (such as for mutual exclusion in Sects. 11–13). Hence, for consistency we will mostly stick with this convention in our presentation here. Nevertheless, the variant with strong chop can sometimes help to slightly shorten proofs.

Local PITL is decidable (see our earlier work with Halpern [45] (reproduced in [54])), but it has nonelementary complexity (a theorem by Kozen presented there). Therefore, the PITL subset consisting of 2-to-1 formulas is likewise decidable. Our recent axiomatic completeness proof for PITL with infinite time [57] ensures that a corresponding PITL theorem can be deduced for any 2-to-1 formula.

We can generalise the previous 2-to-1 example p to be any state formula w. Any NL^1 formula T (see Definition 3.1 in Sect. 3.2) is 2-to-1. Furthermore, any formulas $\Diamond C$ and $\diamondplus C$ are 2-to-1, where $\diamondplus C$ (defined in Table 1) tests that C is true in some prefix subinterval, possibly the interval itself even if it is infinite. The cases for NL^1 formulas and $\diamondplus C$ can subsume the case for a state formula w because it is in NL^1 and additionally semantically equivalent to the PITL formula $\diamondplus w$. Let us now consider one lemma dealing with all these cases and another concerning the conjunction of 2-to-1 formulas:

Lemma 4.3 *All of the following are 2-to-1 formulas:*

1. *Any state formula w.*
2. *Any NL^1 formula T.*
3. *Any PITL formula of the form $\Diamond C$.*
4. *Any PITL formula of the form $\diamondplus C$.*
5. *Any PITL formula of the form $w \supset \diamondplus B$, for any state formula w and PITL formula B.*

Proof We examine each of these separately:

- A state formula w: If an interval σ satisfies the formula $w;w$ then the semantics of PITL ensures that σ's first state must satisfy w. Hence, σ does as well. As we already noted, this case can alternatively be subsumed by either the next one for NL^1 formulas or the later one for $\diamondplus C$.
- An NL^1 formula T: Let σ be an interval satisfying $T;T$ and let σ' be the subinterval satisfying the left instance of T. We use case analysis to show that the interval σ indeed satisfies T as well.
  - If σ' has only one state, then σ itself must satisfy the right-hand instance of T.
  - Otherwise, both σ and σ' have two or more states. An NL^1 formula cannot test beyond the second state and distinguish between the intervals σ and σ'. Consequently, if σ' satisfies T, so must σ.
- A PITL formula of the form $\Diamond C$: This follows from the next chain of valid implications involving the definition of weak chop in Table 1:

$$(\Diamond C);\Diamond C \quad\supset\quad \big((\Diamond C)^\frown \Diamond C\big) \vee (\mathit{inf} \wedge \Diamond C) \quad\supset\quad (\Diamond\Diamond C) \vee \Diamond C \quad\supset\quad \Diamond C.$$

  For the purposes of comparison, here is a somewhat shorter valid chain of implications using Lemma 4.2's alternative characterisation of 2-to-1 formulas based on strong chop (i.e., $\models (A^\frown A) \supset A$):

$$(\Diamond C)^\frown \Diamond C \quad\supset\quad \Diamond\Diamond C \quad\supset\quad \Diamond C.$$

– A PITL formula of the form $\Diamond\!\!\!\!\!\diamond C$: Suppose an interval σ satisfies $(\Diamond\!\!\!\!\!\diamond C);\Diamond\!\!\!\!\!\diamond C$. Then some prefix subinterval σ' of σ (perhaps σ itself) satisfies the left instance of $\Diamond\!\!\!\!\!\diamond C$ and furthermore σ satisfies $\Diamond\!\!\!\!\!\diamond\Diamond\!\!\!\!\!\diamond C$. Moreover, some prefix subinterval σ'' of σ' (perhaps σ' itself) satisfies the subformula C. Now σ'' is also a prefix subinterval of σ, so consequently σ satisfies $\Diamond\!\!\!\!\!\diamond C$.
  Here is a corresponding chain of valid implications:

$$(\Diamond\!\!\!\!\!\diamond C);\Diamond\!\!\!\!\!\diamond C \quad\supset\quad (\Diamond\!\!\!\!\!\diamond C);\mathit{true} \quad\supset\quad \Diamond\!\!\!\!\!\diamond\Diamond\!\!\!\!\!\diamond C \quad\supset\quad \Diamond\!\!\!\!\!\diamond C.$$

– A PITL formula of the form $w \supset \Diamond\!\!\!\!\!\diamond B$: The equivalence chain below invokes the case for $\Diamond\!\!\!\!\!\diamond C$ to also handle any formula $w \supset \Diamond\!\!\!\!\!\diamond B$:

$$\begin{aligned}\Diamond\!\!\!\!\!\diamond(w\supset B) \quad&\equiv\quad \Diamond\!\!\!\!\!\diamond\big((\neg w)\vee B\big) \quad\equiv\quad \Diamond\!\!\!\!\!\diamond(\neg w)\ \vee\ \Diamond\!\!\!\!\!\diamond B\\ &\equiv\quad (\neg w)\vee\Diamond\!\!\!\!\!\diamond B \quad\equiv\quad w\supset\Diamond\!\!\!\!\!\diamond B.\end{aligned}$$ □

Lemma 4.4 *For any 2-to-1 formulas A and B, the conjunction $A \wedge B$ is a 2-to-1 formula as well. That is, if $\models (A;A)\supset A$ and $\models (B;B)\supset B$, then also $\models \big((A\wedge B);(A\wedge B)\big)\supset(A\wedge B)$.*

Proof Here is a simple semantic proof with four steps:

1	$\models$	$A;A \quad\supset\quad A$	Assumption
2	$\models$	$B;B \quad\supset\quad B$	Assumption
3	$\models$	$(A\wedge B);(A\wedge B) \quad\supset\quad (A;A)\ \wedge\ (B;B)$	PITL
4	$\models$	$(A\wedge B);(A\wedge B) \quad\supset\quad A\ \wedge\ B$	1-3, Prop.

The mention of "PITL" in Step 3 refers to some routine semantic reasoning about intervals which we do not further justify here. However, we provide detailed deductions for valid properties of this kind in our recent axiomatic completeness proof for PITL with infinite time [57]. We can summarise the proof as a chain of valid implications:

$$(A\wedge B);(A\wedge B) \quad\supset\quad (A;A)\wedge(B;B) \quad\supset\quad A\wedge B.$$ □

The next theorem about 2-to-1 formulas appears to us to be an important, yet previously unknown elementary mathematical property about compositionality:

Theorem 4.5 *If A is 2-to-1, so is $\Box A$. That is, from the valid implication $\models (A;A)\supset A$ follows the next one: $\models ((\Box A);\Box A)\supset\Box A$.*

Proof Our goal is to prove the validity of the implication below for any 2-to-1 formula A:

$$\models\quad (\Box A);\Box A \quad\supset\quad \Box A. \tag{2}$$

The proof of validity is a little simpler if we use Lemma 4.2's alternative characterisation of 2-to-1 formulas involving strong chop (i.e., $\models (A^\frown A)\supset A$) to establish the validity of the next semantically equivalent implication:

$$\models\quad (\Box A)^\frown\Box A \quad\supset\quad \Box A. \tag{3}$$

Let σ be an interval satisfying $(\Box A)^\frown\Box A$. We now show that σ also satisfies $\Box A$. The semantics of strong chop ensures that there exists at least one pair of subintervals σ' and σ'' of σ which share a state, combine to make σ and both satisfy the subformula $\Box A$. Here is a diagrammatic representation of this:

$$\overbrace{\underbrace{(\Box A)}_{\sigma'}{}^\frown\underbrace{\Box A}_{\sigma''}}^{\sigma}\,.$$

From the semantics of $\Box$ we have that every suffix subinterval of σ' and σ'' (including σ' and σ'' themselves) satisfies the subformula A. Let us now consider an arbitrary suffix subinterval σ''' of the overall interval σ. We want to show that it satisfies A and hence σ satisfies $\Box A$. There are two subcases:

- **σ''' consists of a suffix of σ' followed by all of σ'' (perhaps even σ itself):** Now the suffix subinterval of σ' and the subinterval σ'' both satisfy A. Therefore, σ''' satisfies the formula $A^\frown A$. The assumption that A is 2-to-1 and Lemma 4.2 then yield that σ''' likewise satisfies A.
- **σ''' is a suffix of σ'' (perhaps even σ'' itself):** Hence, σ''' immediately satisfies the 2-to-1 formula A.

Therefore, σ satisfies $\Box A$. Consequently, implication (3) is valid, as is (2), so $\Box A$ is indeed 2-to-1.

Observe that we can alternatively express this reasoning about the interval σ and the formula $(\Box A)^\frown\Box A$ by means of a chain of valid implications starting with $(\Box A)^\frown\Box A$ and ending with $\Box A$:

$$(\Box A)^\frown\Box A \quad\supset\quad \Box\big(A \vee (A^\frown A)\big) \quad\supset\quad \Box\big(A \vee A\big) \quad\supset\quad \Box A. \tag{4}$$

$\Box$

Lemma 4.6 *For any NL^1 formula T and PITL formulas B and C, the following are 2-to-1 formulas:*

$$\Box T \qquad \Box\Diamond C \qquad \Box\Diamond\!\!\!\!\!\;{\scriptscriptstyle i}\;\, C \qquad \Box(w \supset \Diamond\!\!\!\!\!\;{\scriptscriptstyle i}\;\, B).$$

Proof This readily follows from Lemma 4.3 about some simple kinds of 2-to-1 formulas together with Theorem 4.5. $\Box$

Recall that T subsumes w, so $\Box T$ likewise subsumes $\Box w$.

The 2-to-1 formulas of the form $\Box(w \supset \Diamond\!\!\!\!\!\;{\scriptscriptstyle i}\;\, B)$ can express some standard temporal liveness properties. For example, the PTL formula $\Diamond q$ is semantically equivalent to $\Diamond\!\!\!\!\!\;{\scriptscriptstyle i}\;\,\Diamond q$, so consequently the conventional PTL formula $\Box(p \supset \Diamond q)$ is in fact 2-to-1. Indeed, its subformula $p \supset \Diamond q$ is also 2-to-1 because the semantic equivalence of $\Diamond q$ and $\Diamond\!\!\!\!\!\;{\scriptscriptstyle i}\;\,\Diamond q$ ensures that the implication can be expressed as $p \supset \Diamond\!\!\!\!\!\;{\scriptscriptstyle i}\;\,\Diamond q$.

Let us now discuss why the following three frequently occurring formulas (all defined in Table 1) are 2-to-1:

$$\mathit{finite} \qquad \mathit{fin}\, w \qquad \mathit{inf},$$

where w is any state formula. The first one *finite* is 2-to-1 because it denotes $\Diamond\mathit{empty}$, which is 2-to-1 by Lemma 4.3. The formula $\mathit{fin}\, w$ denotes $\Box(\mathit{empty} \supset w)$. The subformula $\mathit{empty} \supset w$ is in NL^1, so $\Box(\mathit{empty} \supset w)$ and $\mathit{fin}\, w$ are 2-to-1 by Lemma 4.6. It then follows from this that *inf*, which denotes $\neg\mathit{finite}$, is also 2-to-1 since it is semantically equivalent to $\mathit{fin}\,\mathit{false}$. Alternatively, *inf* is 2-to-1 because it can be expressed as $\Box\mathit{more}$. Now *more* is in NL^1, so $\Box\mathit{more}$ is likewise 2-to-1 by Lemma 4.6.

4.1 Introduction, combining and extension of 2-to-1 formulas

Our interest here is in compositionally proving the validity of implications of following form:

$$w \wedge Sys \quad \supset \quad A \wedge \mathit{fin}\, w',$$

where w is a state formula about the initial state, *Sys* expresses some abstract or concrete system's behaviour in PITL, A is a 2-to-1 formula and w' is a state formula about the final state if the system terminates. Now we can build *Sys* by starting with various simple formulas corresponding to individual concrete or abstract program steps. These are then combined in different ways, such as sequentially (e.g., using chop) or in parallel (using logical-and). For example, *Sys* could be the sequential composition $Sys'; Sys''$ of two parts Sys' and Sys''. Suppose we have already proved the validity of the following two implications for Sys' and Sys'', respectively:

$$\models \quad w \wedge Sys' \quad \supset \quad A \wedge \mathit{fin}\, w''$$
$$\models \quad w'' \wedge Sys'' \quad \supset \quad A \wedge \mathit{fin}\, w'.$$

The validity of the previous implication for *Sys* then follows from the validity of these, in part because the two instances of the 2-to-1 formula A can be combined into a single one.

Our mutual exclusion examples discussed later in Sects. 11–13 involve two processes running in parallel, with each containing several sequential parts. We first employ a technique for showing that some of the system's individual steps imply 2-to-1 formulas. We regard this as a way to introduce 2-to-1 formulas. These can then be combined together (e.g., sequentially or in parallel) or extended using some of the other techniques to obtain 2-to-1 formulas about bigger portions of the overall system. Eventually we show that the entire system with its initial condition implies a 2-to-1 formula.

Let us now discuss four general kinds of techniques to help compositionally reason about 2-to-1 formulas. Each is associated with one or two valid generic implications concerning such formulas. We later present some specific suitable implications when we look at the four techniques individually in greater detail. However, these implications are not meant to be exhaustive. Below is a list of the main categories we consider:

Introduction of a 2-to-1 formula A	$\models A' \supset A$
Sequential combining of two copies of a 2-to-1 formula A	$\models (A; A) \supset A$
Extension of a 2-to-1 formula A **leftward or rightward**	$\models (A'; A) \supset A$
	$\models (A; A') \supset A$
Parallel combining of two 2-to-1 formulas A and A'	$\models (A \wedge A') \supset A''$.

The shorthand **ISEP** can be used as an abbreviation for the four parts ***I**ntroduction*, ***S**equential combining*, ***E**xtension leftward or rightward* and ***P**arallel combining*. The later Sects. 6–9 cover in detail ISEP techniques for a class of 2-to-1 formulas for backward analysis. The basic theory of ISEP techniques can even be formalised in PITL with just chop and *skip* and so without chop-star. The theory therefore seems fairly elementary from a mathematical standpoint.

Section 10 adds a further technique for ***I**teration* of 2-to-1 formulas for backward analysis. The abbreviation **ISEPI** enlarges ISEP to include this as well. The ISEPI techniques are later applied to mutual exclusion in Sects. 11–13.

We now illustrate how the first three ISEP techniques can be used together to combine several sequential formulas in order to obtain from them a single 2-to-1 formula. Let *Sys* be a *hypothetical* system with four sequential parts somehow or another expressed in PITL as the formulas $Sys_1, \dots, Sys_4$. We have *Sys* itself denote the sequential composition of $Sys_1, \dots, Sys_4$:

$$Sys \quad \widehat{=} \quad Sys_1; Sys_2; Sys_3; Sys_4.$$

Now further assume that $Sys_1, \dots, Sys_4$ have the associated valid implications below, which also include five state formulas $w_1, \dots, w_5$ to serve as pre- and post-conditions:

$$\begin{array}{llllll} \models & w_1 \wedge Sys_1 & \supset & \Box\neg p \ \wedge \ \mathit{fin}\, w_2 \\ \models & w_2 \wedge Sys_2 & \supset & (\mathit{finite} \wedge \mathit{fin}\, p) \ \wedge \ \mathit{fin}\, w_3 \\ \models & w_3 \wedge Sys_3 & \supset & \Diamond\Box q \ \wedge \ \mathit{fin}\, w_4 \\ \models & w_4 \wedge Sys_4 & \supset & (\mathit{finite} \wedge \mathit{fin}\, q) \ \wedge \ \mathit{fin}\, w_5. \end{array} \tag{5}$$

Our goal here is to compositionally prove that the four valid implications in (5) together ensure that *Sys* implies the 2-to-1 liveness formula $\Box(p \supset \Diamond q)$ as expressed by the next valid implication:

$$\models \quad w_1 \wedge Sys \quad \supset \quad \Box(p \supset \Diamond q) \ \wedge \ \mathit{fin}\, w_5. \tag{6}$$

It happens that all the subformulas $\Box\neg p$, $(\mathit{finite} \wedge \mathit{fin}\, p)$, $\Diamond\Box q$ and $(\mathit{finite} \wedge \mathit{fin}\, q)$ in (5) are in fact themselves 2-to-1. However, this point is not essential here since our sole aim is to show the validity of implication (6) relating *Sys* with the 2-to-1 formula $\Box(p \supset \Diamond q)$.

Below is a more detailed discussion which explains and motivates each ISEP technique and relates the first three of them to our example:

– **ISEP *Introduction* of a 2-to-1 formula:** Here we show that some formula A' concerning a system step implies the desired 2-to-1 formula A:

$$\models \quad A' \quad \supset \quad A.$$

In our example (5), ISEP ***Introduction*** concerns three subformulas $\Box\neg p$, $\Diamond\Box q$ and $\mathit{finite} \wedge \mathit{fin}\, q$ for which we can formalise some valid PTL implications:

$$\models \Box\neg p \supset \Box(p \supset \Diamond q) \qquad \models \Diamond\Box q \supset \Box(p \supset \Diamond q) \qquad \models (\mathit{finite} \wedge \mathit{fin}\, q) \supset \Box(p \supset \Diamond q).$$

These ways for ISEP ***Introduction*** of the 2-to-1 formula $\Box(p \supset \Diamond q)$ provide a means to obtain from three of the four valid implications in (5) the valid implications below for Sys_1, Sys_3 and Sys_4, respectively:

$$\begin{array}{llllll} \models & w_1 \wedge Sys_1 & \supset & \Box(p \supset \Diamond q) \ \wedge \ \mathit{fin}\, w_2 \\ \models & w_3 \wedge Sys_3 & \supset & \Box(p \supset \Diamond q) \ \wedge \ \mathit{fin}\, w_4 \\ \models & w_4 \wedge Sys_4 & \supset & \Box(p \supset \Diamond q) \ \wedge \ \mathit{fin}\, w_5. \end{array} \tag{7}$$

Incidentally, the justification for obtaining $\Box(p \supset \Diamond q)$ from $\mathit{finite} \wedge \mathit{fin}\, q$ can be subsumed by the case for $\Diamond\Box q$ owing to the next chain of valid implications:

$$\mathit{finite} \wedge \mathit{fin}\, q \quad \supset \quad \Diamond\Box q \quad \supset \quad \Box(p \supset \Diamond q).$$

The valid implications such as $\models (\Box\neg p) \supset \Box(p \supset \Diamond q)$ for ISEP **Introduction** *of a 2-to-1-formula are quite important since they can provide a way to start a compositional analysis involving this formula.*

It is straightforward to check that if we have a valid implication $\models A' \supset A$ for ISEP ***I****ntroduction*, then the ones below can also be used for ISEP ***I****ntroduction*:

$$\models \quad \Box A' \supset \Box A \qquad \models \quad \boxed{\text{f}} A' \supset \boxed{\text{f}} A.$$

For example, from $\models \neg p \supset (p \supset \Diamond q)$ follows $\models \Box\neg p \supset \Box(p \supset \Diamond q)$.

- **ISEP *Sequential combining* of two instances of a 2-to-1 formula:** Here we take two sequential instances of a 2-to-1 formula A and merge them together:

  $$\models \quad A; A \quad \supset \quad A.$$

  This with the particular 2-to-1 formula $\Box(p \supset \Diamond q)$ together provide a way to reduce the two valid implications in (7) for Sys_3 and Sys_4 to the next valid one concerning their sequential composition $Sys_3; Sys_4$:

  $$\models \quad w_3 \wedge (Sys_3; Sys_4) \quad \supset \quad \Box(p \supset \Diamond q) \wedge \mathit{fin}\, w_5. \tag{8}$$

  Theorems and lemmas about closures provide ways to obtain an instance of an ISEP technique for ***S****equential combining* from a simpler variant of itself. For example, Theorem 4.5 ensures that $\models (A; A) \supset A$ yields $\models ((\Box A); \Box A) \supset \Box A$.

- **ISEP *Extension* of a 2-to-1 formula *leftward or rightward*:** The previous ISEP technique of ***S****equentially combining* two instances of a 2-to-1 formula A such as $\Box(p \supset \Diamond q)$ seems quite attractive. Unfortunately, it is not always the case that two adjacent subintervals *both* satisfy such a 2-to-1 formula A so that the overall interval automatically also does. However, if one of the subintervals satisfies A, then we can try to simplify the sequential compositions $A'; A$ and $A; A'$ involving A and some other suitable formula A'. The next two valid implications show the two possible ways to perform the ISEP technique of ***E****xtending leftward or rightward* by merging A and A' together into A:

  $$\models \quad A'; A \quad \supset \quad A \qquad\qquad \models \quad A; A' \quad \supset \quad A.$$

  Of course, the implications do not work for arbitrary A', but we shortly consider some actual practical instances.
  Observe that the previous ISEP technique of **S**equential combining *of a 2-to-1 formula with itself (i.e.,* $\models (A; A) \supset A$*) is in fact just a special case of ISEP* **E**xtending leftward or rightward, *where* A' *is identical to the 2-to-1 formula* A.
  It seems that sequential extension can be highly dependent on the nature of A'. The next valid implication illustrates the first case $\models (A'; A) \supset A$:

  $$\models \quad (\mathit{finite} \wedge \mathit{fin}\, p); \Box(p \supset \Diamond q) \quad \supset \quad \Box(p \supset \Diamond q). \tag{9}$$

  Here we take A to be the 2-to-1 formula $\Box(p \supset \Diamond q)$ and extend it leftward by the formula $\mathit{finite} \wedge \mathit{fin}\, p$ which plays the role of A'. Implication (9) is valid because the instance of p in the left operand of the chop ensures that p is also initially true in the right operand's subinterval. Therefore, the right-hand subinterval moreover satisfies $\Diamond q$, so the prefix subintervals of the overall interval which start before the right-hand subinterval and contain it likewise satisfy $\Diamond q$, and hence also the 2-to-1 formula $p \supset \Diamond q$. We can then use valid implication (9) to obtain from the implication for Sys_2 in (5) and the later one for $Sys_3; Sys_4$ in (8) the next valid implication for $Sys_2; Sys_3; Sys_4$:

  $$\models \quad w_2 \wedge (Sys_2; Sys_3; Sys_4) \quad \supset \quad \Box(p \supset \Diamond q) \wedge \mathit{fin}\, w_5. \tag{10}$$

Once again using the fact that $\Box(p \supset \Diamond q)$ is 2-to-1, we sequentially combine its two instances in the earlier implication for Sys_1 in (5) and the other implication (10) for $Sys_2; Sys_3; Sys_4$ to arrive at our overall goal, the validity of implication (6) for Sys.
Sect. 8 consider ways to obtain an instance of an ISEP technique for ***E**xtending leftward or rightward* from a simpler variant of itself (e.g., see Theorems 8.1 and 8.7).

Here is a chain of valid implications summarising of all of the ISEP transformations which we have so far applied on the sequential composition of the original subformulas $\Box\neg p$, $(\mathit{finite} \wedge \mathit{fin}\, p)$, $\Diamond\Box q$ and $(\mathit{finite} \wedge \mathit{fin}\, q)$ in (5):

$$\begin{array}{lll}
& \underbrace{(\Box\neg p)}_{Sys_1}; (\mathit{finite} \wedge \mathit{fin}\, p); \underbrace{(\Diamond\Box q)}_{Sys_3}; \underbrace{(\mathit{finite} \wedge \mathit{fin}\, q)}_{Sys_4} & \\
& & \textbf{I}\text{ntroduction} \\
\supset & \Box(p \supset \Diamond q); (\mathit{finite} \wedge \mathit{fin}\, p); \underbrace{\Box(p \supset \Diamond q); \Box(p \supset \Diamond q)}_{Sys_3 \text{ and } Sys_4} & \\
& & \textbf{S}\text{equential combining} \\
\supset & \Box(p \supset \Diamond q); \underbrace{(\mathit{finite} \wedge \mathit{fin}\, p); \Box(p \supset \Diamond q)}_{Sys_2 \text{ and } Sys_3;Sys_4} & \\
& & \textbf{E}\text{xtending leftward} \\
\supset & \underbrace{\Box(p \supset \Diamond q); \Box(p \supset \Diamond q)}_{Sys_1 \text{ and } Sys_2;Sys_3;Sys_4} & \\
& & \textbf{S}\text{equential combining} \\
\supset & \Box(p \supset \Diamond q). &
\end{array}$$

Underbraces indicate the subformulas reduced to the 2-to-1 formula $\Box(p \supset \Diamond q)$ in each step and also give the associated parts of Sys. Instead of the first step's reductions of each of the pair of 2-to-1 formulas $\Diamond\Box q$ and $\mathit{finite} \wedge \mathit{fin}\, q$ to $\Box(p \supset \Diamond q)$, we can alternatively use ISEP ***I**ntroduction* to reduce $\mathit{finite} \wedge \mathit{fin}\, q$ to $\Diamond\Box q$, and then invoke ISEP ***S**equential combining* on $(\Diamond\Box q); \Diamond\Box q$ to obtain $\Diamond\Box q$. We follow that by a second application of ISEP ***I**ntroduction* to arrive at our goal $\Box(p \supset \Diamond q)$. Here is a chain of valid implications summarising this:

$$\begin{array}{lll}
& (\Diamond\Box q); \underbrace{(\mathit{finite} \wedge \mathit{fin}\, q)}_{Sys_4} & \\
& & \textbf{I}\text{ntroduction} \\
\supset & \underbrace{(\Diamond\Box q); \Diamond\Box q}_{Sys_3 \text{ and } Sys_4} & \\
& & \textbf{S}\text{equential combining} \\
\supset & \underbrace{\Diamond\Box q}_{Sys_3;Sys_4} & \\
& & \textbf{I}\text{ntroduction} \\
\supset & \Box(p \supset \Diamond q) &
\end{array}$$

We now consider the last of the four ISEP techniques, namely **ISEP *Parallel combining* of two suitable 2-to-1 formulas**. Consider a hypothetical system Sys' constructed as the conjunction $Sys'_1 \wedge Sys'_2$ of two parts Sys'_1 and Sys'_2, both somehow expressed in PITL. Suppose we have the following valid implications for Sys'_1 and Sys'_2:

$$\begin{array}{lllll}
\models & w'_{1,1} \wedge Sys'_1 & \supset & A \wedge \mathit{fin}\, w'_{1,2} & \\
\models & w'_{2,1} \wedge Sys'_2 & \supset & A' \wedge \mathit{fin}\, w'_{2,2}, &
\end{array}$$

Description of ISEPI technique	Basis	Use
***I**ntroduction* (simple version): $\models A' \supset A$	(19)	(62)
***I**ntroduction* (with relaxed assumption): $\models A' \supset A$	(20)	(63)
***S**equential combining* of 2-to-1 formula: $\models (A; A) \supset A$	Def. 4.1	(64), (75)
***E**xtend* a 2-to-1 formula *rightward*: $\models (A; A') \supset A$	(25)	(65)
***P**arallel combining* of 2-to-1 formulas: $\models (A \wedge A') \supset A''$	(30)	(50), (66)
***I**teration* of $+$-to-1 formula: $\models A^+ \supset A$	(34)	(35)
***I**teration* of "almost" $*$-to-1 formula: $\models w \wedge A^* \supset A$	(40)	(54)

Table 3 Examples of ISEPI-based compositional reasoning about 2-to-1 formulas

where the state formulas $w'_{1,1}, \ldots, w'_{2,2}$ serve as pre- and post-conditions. ISEP ***P**arallel combining* provides a way to obtain a similar implication concerning Sys' from these two. Here is the most straightforward such implication which is valid:

$$\models \quad (w'_{1,1} \wedge w'_{2,1}) \;\wedge\; (Sys'_1 \wedge Sys'_2) \quad\supset\quad A \wedge A' \;\wedge\; fin(w'_{1,2} \wedge w'_{2,2}).$$

However, we are particularly interested in cases where A and A' are 2-to-1 formulas and moreover their conjunction $A \wedge A'$ implies some formula A'' which is noticeably simpler than the conjunction:

$$\models \quad A \wedge A' \quad\supset\quad A''.$$

Here is a valid PTL formula illustrating the ISEP technique of ***P**arallel combining*:

$$\models \quad \Box(p \supset \bigcirc p) \;\wedge\; \Box(q \supset \bigcirc \neg p) \quad\supset\quad \Box\neg(p \wedge q). \tag{11}$$

The following is another PTL example of ISEP ***P**arallel combining*:

$$\models \quad \Box(p \supset \Diamond\Box p) \;\wedge\; \Box(q \supset \Diamond\Box\neg p) \quad\supset\quad \Box\neg(p \wedge q).$$

ISEP ***P**arallel combining* finds application in Sects. 11–13 when we want to merge together the 2-to-1 formulas obtained for each of two parallel processes concerning mutual exclusion. Observe that from $\models (A \wedge A') \supset A''$ readily follows $\models ((\Box A) \wedge (\Box A')) \supset \Box A''$. This semantic inference rule can be used to prove the validity of the two implications just given concerning $\Box\neg(p \wedge q)$ from simpler ones about $\neg(p \wedge q)$.

Later Sects. 6–9 will consider the ISEP techniques of ***I**ntroduction*, ***S**equential combining*, ***E**xtension* and ***P**arallel combining* on a class of formulas which are suitable for backward analysis. For the convenience of readers, Table 3 provides an index to various additional instances of the implications subsequently mentioned for the various ISEP techniques. This includes two extra entries for combining ***I**terations* of a *$+$-to-1 formula* and an *"almost" $*$-to-1 formula*, which we describe later on in Sect. 10, so in fact all the ISEPI techniques are represented in Table 3.

Remark 4.7 It is interesting to note that in our applications of the ISEPI techniques considered above and later on, the concrete instances of *all* the formulas A, A' and A'' found in the implications are always 2-to-1 formulas. For example, all three $\Box$-subformulas in implication (11), which involves ISEPI ***P**arallel combining* (i.e., $\models (A \wedge A') \supset A''$), are 2-to-1 by Lemma 4.6 because in each of them, the operand of $\Box$ is in NL^1. In fact, the sole exception to formulas being 2-to-1 is just the statement of Theorem 8.7 in Sect. 8.2 for extending a 2-to-1 formula A to the right: $\models (A; A') \supset A$. However, even there the generic formula for A' is in a class called *1-to-$\boxed{f}$ formulas* (see Definition 8.2 in Sect. 8.1) which, like the class of 2-to-1-formulas, is closed under conjunction and the temporal operator $\Box$ (as stated in Sect. 8.1 in Lemma 8.4 and Theorem 8.5). In our application of Theorem 8.7 in Sect. 12, the concrete instance of A' is in fact both 1-to-$\boxed{f}$ and 2-to-1.

4.2 2-to-1 formulas involving finite prefix subintervals

The earlier Theorem 4.5 shows that the class of 2-to-1 formulas is closed under the operator $\Box$, which concerns *suffix subintervals*. It is natural to ask whether time symmetry can help extend the result to *prefix subintervals* and the associated operator $\boxed{f}$. In this section we demonstrate that this is indeed the case. The result is needed when we later consider in Sect. 6 a class of 2-to-1 formulas suitable for backward analysis. These 2-to-1 formulas play a central role in practically all of the subsequent sections, including Sects. 11–13 on mutual exclusion.

Theorem 4.8 *If A itself is 2-to-1 for finite time, so is $\boxed{f}A$* for all intervals*, including infinite ones. More precisely, if $\models \big(finite \wedge (A; A)\big) \supset A$, then $\models \big((\boxed{f}A); \boxed{f}A\big) \supset \boxed{f}A$.*

Proof The proof is largely based on applying time symmetry to the earlier proof for Theorem 4.5, which concerns $\Box$ and suffix subintervals instead of $\boxed{f}$ and prefix subintervals. The earlier chain of valid implications (4) in Theorem 4.5 can be adapted for use with $\boxed{f}A$ in place of $\Box A$:

$$(\boxed{f}A)^\frown\boxed{f}A \quad\supset\quad \boxed{f}\big(A \vee (A^\frown A)\big) \quad\supset\quad \boxed{f}\big(A \vee A\big) \quad\supset\quad \boxed{f}A. \tag{12}$$

$\Box$

The following is a simple corollary of Theorem 4.8:

Corollary 4.9 *If a formula $\boxed{f}A$ is 2-to-1 for finite time, it is itself likewise 2-to-1 for all intervals, including infinite ones. More precisely, if $\models \big(finite \wedge ((\boxed{f}A); \boxed{f}A)\big) \supset \boxed{f}A$, then $\models \big((\boxed{f}A); \boxed{f}A\big) \supset \boxed{f}A$.*

Proof We start with $(\boxed{f}A); \boxed{f}A$. Now the PITL formulas $\boxed{f}A$ and $\boxed{f}\boxed{f}A$ are semantically equivalent since they both inspect exactly the finite prefix subintervals. The assumption that $\boxed{f}A$ is 2-to-1 for finite time together with Theorem 4.8 ensures that $\boxed{f}\boxed{f}A$ is 2-to-1 for all intervals. Hence, so is the equivalent formula $\boxed{f}A$. $\Box$

Remark 4.10 It is not hard to adapt the results in this section to deal with $\boxed{i}A$, which is the weak version of $\boxed{f}A$ defined in Table 1. We omit the details here.

A formula $\boxed{f}A$ can in principle be 2-to-1 even if A itself is not 2-to-1. The formula $\boxed{f}skip$ is a (not especially useful) example. This is because $\boxed{f}skip$ is semantically equivalent to the 2-to-1 formula *false*, but the operand *skip* is not 2-to-1 by our earlier discussion near the beginning of this Sect. 4. At present we are not aware of any such formulas with some practical benefits.

5 Time reversal and reflection

In this section we consider two complementary ways to exploit time symmetry. The first is syntactic and the second is semantic.

One way to extend known facts and techniques is by interpreting them *in reverse*. For example, as we discussed in Sect. 1, the 2-to-1 PTL formula $p \supset \Diamond q$ can be viewed as a **forward analysis** from a state in which p is true to one in which q is true. For **backward analysis**, we in essence reverse our perspective by means of the formula $(\mathit{fin}\, p) \supset \Diamond q$ ("*if p is true in the final state, then q is true in some state*"). This implication considers the behaviour of p in a finite interval's last state rather than the first one. In the two sample implications, the subformula $\Diamond q$ has the same semantic meaning in both the forward or reversed perspectives. The reversed way of reasoning can with care provide a basis for performing backward analysis from a situation in a state to some activities which lead up to it. For instance, an analysis of a system fault could investigate various plausible anomalies which must precede it. The next statement is also an example: "If I am wearing shoes, then they must have been previously placed on my feet".

We will look at some simple and natural *syntactic* transformations on formulas which involve time symmetry and are referred to here as **time reversal**. These transformations are in general limited to *finite intervals*, so we employ a **two-stage** approach to also obtain results for *infinite time*. For example, we can prove validity of suitable formulas for infinite time after using time reversal to establish their validity for finite time. The current section includes some compositional uses of the two-stage process on the class of 2-to-1 formulas already introduced in Sect. 4. Various 2-to-1 formulas are then later applied to doing backward analysis of mutual exclusion in Sects. 11–13.

For any PITL formula A, define the temporal reversal A^r by induction on A's syntax to act like A in reverse:

$$\mathit{true}^r \mathrel{\widehat{=}} \mathit{true} \qquad p^r \mathrel{\widehat{=}} \mathit{fin}\, p \qquad (\neg A)^r \mathrel{\widehat{=}} \neg(A^r) \qquad (A \vee B)^r \mathrel{\widehat{=}} A^r \vee B^r$$
$$\mathit{skip}^r \mathrel{\widehat{=}} \mathit{skip} \qquad (A^\frown B)^r \mathrel{\widehat{=}} B^{r\frown} A^r \qquad (A^\star)^r \mathrel{\widehat{=}} (A^r)^\star.$$

For instance, more^r (the same as $(\mathit{skip}^\frown \mathit{true})^r$) reduces to $\mathit{true}^\frown \mathit{skip}$, which is semantically equivalent to *more* in finite intervals (although not in infinite ones). Similarly, $(\boxed{\scriptstyle f} A)^r$ reduces to $\Box(A^r)$.

For a finite interval σ, let σ^r denote the interval $\sigma_{|\sigma|} \ldots \sigma_0$ which temporally reverses σ. Observe that any such σ equals the twice reversed interval σ^{rr}. Here are some simple lemmas concerning time reversed intervals and formulas:

Lemma 5.1 *For any finite interval σ and PITL formula A, the following are equivalent statements:*

(a) $\sigma \models A$
(b) $\sigma^r \models A^r$.

Proof We do induction on formula A's syntax. □

Lemma 5.2 *Any PITL formula A is semantically equivalent to A^{rr} in all finite intervals. This can be expressed by the valid implication below:*

$$\models \mathit{finite} \supset (A \equiv A^{rr}).$$

Proof We use Lemma 5.1 together with the equivalence of σ and σ^{rr} to show that A and A^{rr} have the same truth values for every finite interval σ:

$$\sigma \models A \quad \text{iff} \quad \sigma^r \models A^r \quad \text{iff} \quad \sigma^{rr} \models A^{rr} \quad \text{iff} \quad \sigma \models A^{rr}. \qquad \Box$$

Lemma 5.3 *For any PITL formula A, the following statements are equivalent:*

(a) $\models$ *finite* $\supset A$
(b) $\models$ *finite* $\supset A^r$.

Proof The formula *finite* $\supset A$ is valid iff all finite intervals satisfy A. Let $(\Sigma^+)^r$ denote the set of reversed finite intervals. This in fact equals Σ^+. Time reversal of the intervals creates a 1-to-1 mapping between Σ^+ and itself. Furthermore, Lemma 5.1 ensures that each finite interval σ satisfies A iff the finite interval σ^r satisfies A^r. Hence, (a) and (b) are indeed equivalent statements. $\Box$

Note that PITL with just finite time, like some other temporal logics such as quantified PTL, expresses the regular languages with words having one or more letters (as we discuss in [54]). The set of regular languages for any (finite) alphabet is closed under word reversal. This explains semantically why reversal cannot increase PITL's expressiveness.

The next *semantic* concept provides a further application of time symmetry:

Definition 5.4 (Reflections) A PITL formula A **reflects** another PITL formula B if $\models$ *finite* $\supset (A \equiv B^r)$. We call A a **reflection** of B.

For example, the state formula $w \vee w'$ reflects *fin*$(w \vee w')$. The 2-to-1 PTL formula $\Diamond w$ reflects itself and so can be said to be **self-reflecting**.

It is important to keep in mind that time reversal and reflection both involve time symmetry, but time reversal is a *syntactic* concept, whereas reflection is a *semantic* one. In practice, we often employ both techniques together.

We now consider some other examples of reflection in order for readers to gain fluency with the concept in the context of PITL. This will help when we later look in Sect. 6 at some properties of reflections of 2-to-1 formulas. The formula $(\mathit{fin}\, p) \supset \Diamond q$ reflects the formula $p \supset \Diamond q$. They indeed exhibit symmetrical behaviour in finite intervals. The first formula $(\mathit{fin}\, p) \supset \Diamond q$ ensures that if p is true in the *final* state, then some state has q true. The second formula $p \supset \Diamond q$ ensures that if p is true in the *initial* state, then some state has q true. It follows that the next formula reflects the 2-to-1 PTL formula $\Box(p \supset \Diamond q)$:

$$\boxed{\scriptstyle f}\big((\mathit{fin}\, p) \;\supset\; \Diamond q\big). \tag{13}$$

It is not hard to see how p is reflected to be $\mathit{fin}\, p$. Similarly, $\Box$ becomes $\boxed{\scriptstyle f}$. We later show in Sect. 6 that formula (13) is likewise 2-to-1. This formula ensures that whenever p is true in an interval state, then q is either true in that same state or some earlier one. Recall from Sect. 3.3 the version of PTL called $\mathrm{PTL}^{\mathrm{U}}$ and having a strong *until* operator. The $\mathrm{PTL}^{\mathrm{U}}$ formula below has the same semantics as the PITL formula (13), although we do not claim that this is obvious:

$$\Box\neg p \;\vee\; \big((\neg p)\ \mathit{until}\ q\big).$$

The left conjunct $\Box\neg p$ deals with intervals where p is never true. In such intervals, q does not need to be true either, so we can ignore its behaviour. The right disjunct

$(\neg p)$ *until* q rather opaquely ensures that if, on the other hand, q is somewhere true, then p will stay false until the first time q is true. This suffices to guarantee that the first instance of p cannot precede the first instance of q in the interval.

Let us now look at some trickier examples of reflection involving 2-to-1 formulas and the operators *skip* and $\bigcirc$. The formula $\Diamond(\mathit{skip} \wedge q)$ reflects the 2-to-1 formula NL^1 formula $\bigcirc q$. Let us consider why this is so. For any finite interval, the formula $\bigcirc q$ ensures that the interval has at least two states with q true in the *second* state. The formula $\Diamond(\mathit{skip} \wedge q)$ likewise ensures that the interval has at least two states with q true in the *penultimate* state (i.e., the one which is next to last). Consequently, any finite interval σ indeed satisfies one of the formulas $\Diamond(\mathit{skip} \wedge q)$ and $\bigcirc q$ iff the interval's reversal σ^r satisfies the other. The next formula reflects the 2-to-1 formula $\Box(p \supset \bigcirc q)$ and by the presentation in Sect. 6 is likewise 2-to-1:

$$\boxed{\scriptstyle f}\big((\mathit{fin}\, p) \;\supset\; \Diamond(\mathit{skip} \wedge q)\big). \tag{14}$$

The only tricky part of the reflection here is when we time-wise reverse the effect of $\bigcirc q$ by reflecting it using $\Diamond(\mathit{skip} \wedge q)$ as discussed above.

Consider what kind of finite intervals are satisfied by formula (14). First of all, a finite interval satisfies the subformula $(\mathit{fin}\, p) \supset \Diamond(\mathit{skip} \wedge q)$ in (14) iff the propositional variable p is false in the interval's last state or the interval has at least two states and the propositional variable q is true in the interval's penultimate state. So if p ends up in the last state being true, then the interval has two or more states and the last one is immediately preceded by another with q true. The effect of the subformula $(\mathit{fin}\, p) \supset \Diamond(\mathit{skip} \wedge q)$ is therefore to make the overall formula (14) test that within each finite prefix subinterval of an interval, if p is true in the final state, then the subinterval has at least two states and q is true in the one just before the final state. This is identical to testing that any state in the overall interval with p true is immediately preceded by another state with q true. The PTL formula below has the same semantics as PITL formula (14):

$$\neg p \;\wedge\; \Box\big((\mathit{more} \wedge \neg q) \supset \bigcirc \neg p\big).$$

We now demonstrate that every formula has a reflection:

Lemma 5.5 *For any PITL formula A, the formula A^r is a reflection of A. In fact, the formulas A and A^r reflect each other.*

Proof Lemma 5.2 ensures for any PITL formula A the valid implication $\models \mathit{finite} \supset (A \equiv A^{rr})$. Therefore, by Definition 5.4 about reflections, the formula A is a reflection of A^r. In addition, we have the trivially valid implication $\models \mathit{finite} \supset (A^r \equiv A^r)$. From this and Definition 5.4 about reflections, the formula A^r is a reflection of A. Consequently, the formulas A and A^r indeed reflect each other. $\Box$

It also follows from our discussion that A reflects B iff B reflects A. Reflecting can sometimes aid in avoiding redundant finite-time proofs in two directions. Instead, we try to do a proof in one time direction and then with care reflect the result to apply the other way around. For example, later on in Sect. 6 we reflect some syntactic classes of 2-to-1 formulas to obtain further classes of 2-to-1 formulas. Sect. 14 discusses how reflection can help reduce reasoning involving $\boxed{\scriptstyle f}$ to simpler PTL-based reasoning.

Here is another example of reflecting based on the previously mentioned chains of implications (4) and (12), which concern the closure of 2-to-1 formulas under $\Box$ and $\boxed{\mathrm{f}}$, respectively:

$$\begin{array}{l}(\Box A)^\frown \Box A \;\supset\; \Box\big(A \vee (A^\frown A)\big) \;\supset\; \Box\big(A \vee A\big) \;\supset\; \Box A \\ (\boxed{\mathrm{f}} A^r)^\frown \boxed{\mathrm{f}} A^r \;\supset\; \boxed{\mathrm{f}}\big(A^r \vee (A^{r\frown} A^r)\big) \;\supset\; \boxed{\mathrm{f}}\big(A^r \vee A^r\big) \;\supset\; \boxed{\mathrm{f}} A^r.\end{array}$$

Remark 5.6 We can alternatively define A^r to be a primitive operator in a variant of PITL called PITL^r. However, it seems at present simpler to work in conventional PITL.

5.1 PTL with past time

In our later application of time symmetry to compositional reasoning with 2-to-1 formulas, we sometimes compare PITL formulas to others in a version of PTL with past time, denoted here as PTL$^-$. It is not a subset of conventional PITL because that does not have past time. Our experience is that even readers with previous experience with ITL will find the unfamiliar processes of viewing formulas in reverse and interval-based backward analysis somewhat challenging. Consequently, it seems beneficial to compare PITL formulas obtained using time symmetry with semantically quite similar formulas in a more widely known formalism such as PTL$^-$.

Time is modelled in PTL$^-$ as being linear and discrete (like for PITL and PTL) but having a bounded past. The syntax of PTL is modified to include the two additional primitive operators $\ominus X$ (read *previous* X) and $\diamondminus X$ (read *once* X). The semantics of a PTL formula X is now expressed as $(\sigma, k) \models X$, where k is any natural number not exceeding $|\sigma|$. The purpose of k is to indicate the *present* state. For example, the semantics of $\ominus$ and $\diamondminus$ are as follows:

$$\begin{array}{lll}(\sigma,k) \models \ominus X & \text{iff} & k > 0 \text{ and } (\sigma, k-1) \models X \\ (\sigma,k) \models \diamondminus X & \text{iff} & \text{for some } j : 0 \le j \le k,\;\; (\sigma, j) \models X.\end{array}$$

Consider the sample formula below:

$$p \;\wedge\; \ominus\neg p \;\wedge\; \Diamond q \;\wedge\; \diamondminus r.$$

This is satisfied by any pair (σ, k) with $k \ge 1$ where p is true in the state σ_k, false in the previous one σ_{k-1}, q is true in the state σ_k or after it, and r is true in the state σ_k or before it.

The derived PTL$^-$ operator ***first*** is defined as follows to test for the first state of an interval:

$$\mathit{first} \quad \widehat{=} \quad \neg\ominus \mathit{true}.$$

We later use the operator *first* to help us relate formulas in PITL with others in PTL$^-$. For example, the following two examples in PTL$^-$ and PTL, respectively, are satisfied by the same intervals:

$$\mathit{first} \wedge \bigcirc(p \supset \ominus q) \qquad \mathit{more} \wedge \big((\bigcirc p) \;\supset\; q\big).$$

More precisely, for any interval σ, the pair $(\sigma, 0)$ satisfies the left-hand PTL$^-$ formula iff σ satisfies the right-hand PTL formula. The PTL$^-$ formula expresses that there are at least two states and the first one, which is the present state, has no past. Furthermore, if p is true in the second state, q is true in its predecessor, the first state. The second formula is in PTL and expresses that the interval has at least two states (with no past), and if p is true in the second one, then q is true in the first. So both formulas concern the same kind of behaviour.

A PTL$^-$ formula X is defined to be satisfiable iff $(\sigma, k) \models X$ holds for some pair (σ, k) with $k \leq |\sigma|$. The formula X is valid iff $(\sigma, k) \models X$ holds for every pair (σ, k) with $k \leq |\sigma|$.

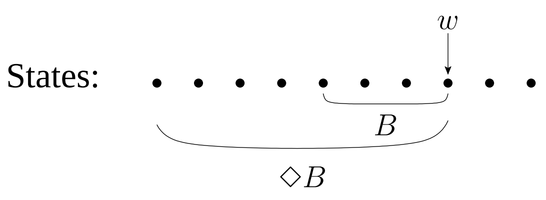
Duan [13, 14] and Bowman et al. [9] present versions of ITL with past-time constructs (see also Gomez and Bowman [22]). So in principle, PTL$^-$ can be regarded as a subset of PITL with past time.

6 2-to-1 formulas for backward analysis

Recall Theorem 4.8 in Sect. 4.2 which establishes that if a PITL formula A is 2-to-1 for finite intervals, then the PITL formula $\boxed{\mathrm{f}}A$ is 2-to-1 for all intervals, including even infinite ones. Let us now consider a significant class of such $\boxed{\mathrm{f}}$-formulas which are shown to be 2-to-1 with the help of time symmetry. They offer a natural compositional framework for *backward analysis*. The previously mentioned PITL formula $\boxed{\mathrm{f}}\big((\mathit{fin}\, p) \supset \Diamond q\big)$ is an example.

The PITL formula $\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ is a generalisation of $\boxed{\mathrm{f}}\big((\mathit{fin}\, p) \supset \Diamond q\big)$ and tests that in any finite prefix interval where w ends true, it is preceded by B. The subformula B therefore represents some activity observable (non-strictly) prior to any state where w is true. Such formulas provide a way to do backward analysis when we want to reason about what must have preceded a state with w true. They will be extensively investigated and applied in our presentation.

Below is an informal graphical representation of a 10-state interval containing some finite 8-state prefix subinterval which satisfies $(\mathit{fin}\, w) \supset \Diamond B$ and ends with w true:



The role which the 2-to-1 formula $\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ plays here is similar to the one for formulas in the past-time variant PTL$^-$ of PTL (see Sect. 5.1) having the form $\Box(w \supset X)$, where the only temporal operators in the PTL$^-$ formula X are past-time ones.

Perhaps the most important result we need is the following one about a key property of the PITL formula $\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$:

Theorem 6.1 *For any state formula w and PITL formula B, the following formula is 2-to-1:*

$$\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big). \tag{15}$$

Proof The operand $(\mathit{fin}\, w) \supset \Diamond B$ can be reflected to obtain the formula $w \supset \Diamond\mkern-11mu{\scriptstyle\mathrm{i}}\mkern6mu B^r$, which is 2-to-1 by our previous Lemma 4.3. Hence, the formula $(\mathit{fin}\, w) \supset \Diamond B$ is itself

2-to-1 for finite intervals. It follows from this and Theorem 4.8 that $\boxed{\scriptstyle f}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ is 2-to-1 for all intervals.

There is also an alternative proof involving the reflection of $\Box$. We can reflect $\boxed{\scriptstyle f}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ to be $\Box\big(w \supset \diamondminus B^r\big)$. By Lemma 4.6, this $\Box$-formula is 2-to-1. Hence, the formula $\boxed{\scriptstyle f}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ is 2-to-1 for finite intervals. By Corollary 4.9, this formula is 2-to-1 for all intervals, including infinite ones. $\Box$

Let us now consider the next instance of $\boxed{\scriptstyle f}\big((\mathit{fin}\, w) \supset \Diamond B\big)$:

$$\boxed{\scriptstyle f}\big((\mathit{fin}\, p) \supset \Diamond(\mathit{skip} \wedge q)\big). \tag{16}$$

We already mentioned formula (16) as (14) when previously defining and explaining the concept of reflecting formulas. It is true for intervals when each state with p true is immediately preceded by one with q true. This is because the formula ensures that any finite prefix subinterval ending with p true in the subinterval's last state has q equal true in the subinterval's penultimate state. So any state with p true must be immediately preceded by one with q true.

We explained when previously discussing the earlier instance of (16) as formula (14) that it is a reflection of the 2-to-1 PTL formula $\Box(p \supset \bigcirc q)$.

Let us now relate formula (16) to one in PTL$^-$, the version of PTL with past time previously discussed in Sect. 5.1. We believe that this will help readers better familiarise themselves with our approach. Formula (16) is comparable to the next PTL$^-$ formula with the standard past-time operator $\ominus$ for examining the previous state:

$$\mathit{first} \;\wedge\; \Box(p \supset \ominus q). \tag{17}$$

By "comparable", we mean here that an interval σ satisfies the first formula (16) iff the pair $(\sigma, 0)$ satisfies the second formula (17). Our use of the PTL$^-$ derived construct *first* in formula (17) ensures that the second subformula $\Box(p \supset \ominus q)$ only considers intervals with no past. This is in order to conform to the time model for PITL which, unlike PTL$^-$, lacks past time.

It can be useful to consider the simple case where the interval σ has just one state. Observe that σ satisfies the first formula (16) iff p is false in that state. Similarly, the pair $(\sigma, 0)$ satisfies the PTL$^-$ formula (17) iff p is false in σ's single state. If σ has exactly two states, then either p is false in both of them or else the initial state has p false and q true and the second one has p true.

Pnueli [66] and Lichtenstein, Pnueli and Zuck [41] give early accounts about how to formalise safety properties for mutual exclusion using past-time formulas of the form $\Box(w \supset X)$, where the temporal formula X only concerns past states and perhaps the current state, but not future ones. We later look at such approaches in more detail in Sect. 16.2.

Here is another example of a $\boxed{\scriptstyle f}$-formula which is an instance of (15) and hence 2-to-1 by Theorem 6.1:

$$\boxed{\scriptstyle f}\big((\mathit{fin}\, p) \;\supset\; \Diamond\neg p\big). \tag{18}$$

This is analogous to the next PTL$^-$ formula with the past-time variant $\diamondminus$ of $\Diamond$:

$$\mathit{first} \;\wedge\; \Box(p \supset \diamondminus\neg p).$$

Introduction	$\models A' \supset A$	Sect. 7
Sequential combining	$\models (A; A) \supset A$	Sect. 6
Extension rightward	$\models (A; A') \supset A$	Sect. 8
Parallel combining	$\models (A_1 \wedge A_2) \supset A'$	Sect. 9
Iteration	$\models A^{+} \supset A,\ \models (w \wedge A^{*}) \supset A$	Sect. 10

Table 4 ISEPI compositional techniques for a 2-to-1 formula for backward analysis

7 ISEPI introduction of 2-to-1 formulas for backward analysis

Recall the ISEPI techniques previously described in Sect. 4.1. A large part of this Sect. 7 and the subsequent Sects. 8–10 concerns the ISEPI techniques for 2-to-1 formulas for backward analysis, including iteration of such formulas. The 2-to-1 formulas and their ISEPI techniques will also be extensively used for backward analysis when we formally study mutual exclusion in Sects. 11–13.

Table 4 gives a summary of our presentation of ISEPI techniques in the previous, current and next sections concerning 2-to-1 formulas for backward analysis. In this section we consider the ISEPI technique of ***I**ntroduction* for use with backward analysis. It can provide a way for a 2-to-1 formula A to be implied from another one A' (i.e., $\models A' \supset A$).

We now discuss two simple valid implications to do the ISEPI technique of ***I**ntroduction* with the 2-to-1 formula $\boxed{\text{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$. As we already mentioned in Sect. 4.1, such implications can be quite important since they provide a way to start a compositional analysis involving the 2-to-1 formula. Therefore, readers should make sure that they understand the material here. Instances of the implications are later used in our analysis of mutual exclusion in Sects. 11–13.

The first valid implication for ISEPI ***I**ntroduction* considered here concerns situations where the state formula w is everywhere false. The implication provides a way to introduce from a quite simple 2-to-1 formula $\Box\neg w$ in PTL the much more complicated 2-to-1 PITL formula $\boxed{\text{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$:

$$\models \quad \Box\neg w \quad \supset \quad \boxed{\text{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big). \tag{19}$$

Proof (Validity of (19)*)* This follows from the fact that if in an interval the state formula w is always false, then the PTL formula $\mathit{fin}\, w$ is false in every finite subinterval. Hence, for such an interval the implication $(\mathit{fin}\, w) \supset \Diamond B$ is trivially true in all finite subintervals. It also follows that the details of B are irrelevant. □

We can alternatively show the validity of implication (19) by observing that the formula $\Box\neg w$ is equivalent to $\boxed{\text{f}}\, \mathit{fin}\, \neg w$ (i.e., $\models \Box\neg w \equiv \boxed{\text{f}}\, \mathit{fin}\, \neg w$). Now $\boxed{\text{f}}\, \mathit{fin}\, \neg w$ is semantically equivalent to $\boxed{\text{f}}\neg\, \mathit{fin}\, w$ and in addition, simple propositional reasoning ensures that $\neg\, \mathit{fin}\, w$ implies $(\mathit{fin}\, w) \supset \Diamond B$ in each finite prefix subinterval.

The next valid implication is an example of (19) and its simple form of ISEPI ***I**ntroduction*:

$$\models \quad \Box\neg p \quad \supset \quad \boxed{\text{f}}\big((\mathit{fin}\, p) \supset \Diamond q\big).$$

This can be interpreted as stating that if p is always false, then every state with p true is (non-strictly) preceded by a state with q true.

Below is a variant of (19) for ISEPI ***I**ntroduction* which *relaxes* the requirement in finite intervals that w is everywhere false. Instead, w only has to be false in all states except for perhaps the last one:

$$\models \quad \Box(\mathit{more} \supset \neg w) \;\wedge\; (\mathit{inf} \vee \Diamond B) \quad\supset\quad \boxed{\text{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big). \tag{20}$$

Observe that if the subformula B is in PTL, so is the antecedent of (20).

Proof (Validity of (20)*)* We consider the two cases for finite and infinite intervals separately. The case for infinite ones is easier, so we look at it first.

- For any *infinite interval* σ, the formula *more* is true for each of σ's suffix subintervals (including σ itself). As a result, the formula $\Box(\mathit{more} \supset \neg w)$ is semantically equivalent to $\Box\neg w$. Therefore, the previous valid implication (19) ensures that σ also satisfies $\boxed{\text{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$.
- On the other hand, suppose σ is a *finite interval* which satisfies the antecedent of (20). Hence, σ satisfies $\Box(\mathit{more} \supset \neg w)$, so in each proper prefix subinterval σ' of σ, the PTL formula $\mathit{fin}\, w$ is false. This in turn ensures that $(\mathit{fin}\, w) \supset \Diamond B$ is true in all such σ'. In addition, σ itself satisfies $\Diamond B$, so it likewise satisfies the implication $(\mathit{fin}\, w) \supset \Diamond B$. Hence, each prefix subinterval of σ, including σ itself, satisfies $(\mathit{fin}\, w) \supset \Diamond B$. Therefore, σ also satisfies $\boxed{\text{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$. $\Box$

Below is a simple valid instance of (20) and the *relaxed* form of ISEPI ***I**ntroduction*:

$$\models \quad \Box(\mathit{more} \supset \neg p) \;\wedge\; (\mathit{inf} \vee \Diamond q) \quad\supset\quad \boxed{\text{f}}\big((\mathit{fin}\, p) \supset \Diamond q\big).$$

In the later Sects. 11–13 on mutual exclusion, the first simpler variant (19) of ISEPI ***I**ntroduction* will be used when a process is not in its critical section. The second relaxed version (20) finds application for the process step in which a request is made to enter the critical section.

8 ISEPI extension of 2-to-1 formulas rightward for backward analysis

We have so far presented the 2-to-1 formulas for backward analysis and looked at associated ISEPI techniques for ***I**ntroduction* and ***S**equential combining*. Here is an example of the ISEPI technique for ***E**xtending rightward* already discussed in Sect. 4.1:

$$\models \quad (\neg p \wedge \Box q); \Box q \quad\supset\quad \neg p \wedge \Box q. \tag{21}$$

Below is a proof using a chain of valid implications showing that the 2-to-1 formula $\neg p \wedge \Box q$ is extended rightward by the 2-to-1 formula $\Box q$:

$$(\neg p \wedge \Box q); \Box q \quad\supset\quad \neg p \wedge (\Box q; \Box q) \quad\supset\quad \neg p \wedge \Box q.$$

We later use implication (21) in Sect. 8.2 when we illustrate how to *incrementally* obtain another variant of the ISEPI technique for ***E**xtending rightward* a 2-to-1 formula.

The earlier Theorem 4.8 in Sect. 4.2 concerns 2-to-1 formulas being closed under $\boxed{\text{f}}$. The next Theorem 8.1, which naturally generalises Theorem 4.8, provides an incremental way to adapt the ISEPI technique of ***E**xtending rightward* a formula A using another one A' to ***E**xtending rightward* the formula $\boxed{\text{f}}A$ using $\boxed{\text{f}}A'$.

Theorem 8.1 *For any PITL formulas A and A', we have the semantic inference rule below:*

$$\models \quad \big(\mathit{finite} \wedge (A; A')\big) \supset A \quad \Rightarrow \quad \models \quad \big((\boxed{f}A); \boxed{f}A'\big) \supset \boxed{f}A. \tag{22}$$

Proof The reasoning in Theorem 4.8's proof can be readily adapted for application to (22) by simply using two formulas A and A' instead of just one. For example, here is a chain of valid implications which generalises the earlier one (12):

$$(\boxed{f}A)^\frown \boxed{f}A' \quad \supset \quad \boxed{f}\big(A \vee (A^\frown A')\big) \quad \supset \quad \boxed{f}\big(A \vee A\big) \quad \supset \quad \boxed{f}A. \qquad \square$$

Theorem 8.1 is later used in Sect. 8.2 in Theorem 8.7's proof. Incidentally, a symmetric variant of Theorem 8.1 to generalise Theorem 4.5 using $\Box$ instead of $\boxed{f}$ is possible (i.e., $\models (A'; A) \supset A \;\; \Rightarrow \models ((\Box A'); \Box A) \supset \Box A$). This can facilitate adapting the ISEPI technique of ***E**xtending leftward* a formula A using another one A' to ***E**xtending leftward* the formula $\Box A$ using $\Box A'$.

8.1 A class of formulas for use with ISEPI extending rightward

There are various classes of formulas which, like the 2-to-1 formulas, are closed under conjunction and $\Box$. Our presentation now considers one for use in the next Sect. 8.2 with the ISEPI technique for ***E**xtending rightward* the 2-to-1 formula $\boxed{f}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ for backward analysis. This material finds later application in our analysis of mutual exclusion in Sects. 11–13 when we merge some sequential steps of a process together.

Definition 8.2 (1-to-$\boxed{f}$ formulas) Any PITL formula A for which the implication $A \supset \boxed{f}A$ is valid is called a **1-to-$\boxed{f}$ formula**.

A 1-to-$\boxed{f}$ formula therefore has the property that if it is true in an interval, then it is also true in all the interval's finite prefix subintervals. The 1-to-$\boxed{f}$ formulas include all state formulas and $\boxed{f}$-formulas as well as the formula *finite* because the next three implications are all valid:

$$\models w \supset \boxed{f}w \qquad \models (\boxed{f}B) \supset \boxed{f}\boxed{f}B \qquad \models \mathit{finite} \supset \boxed{f}\mathit{finite}.$$

The NL^1 formula *more* is not 1-to-$\boxed{f}$ since any one-state interval falsifies the implication $\mathit{more} \supset \boxed{f}\mathit{more}$. However, we have the next lemma for a general syntactic class of NL^1 formulas involving *more*:

Lemma 8.3 *For any NL^1 formula T, the NL^1 implication $\mathit{more} \supset T$ is a 1-to-$\boxed{f}$ formula.*

Proof We consider two cases for intervals with just one state and with more than one state. In each case, we show that the intervals indeed satisfy the following implication:

$$(\mathit{more} \supset T) \quad \supset \quad \boxed{f}(\mathit{more} \supset T). \tag{23}$$

If an interval σ has just one state, then the PTL formula *more* is false, so the interval satisfies $\mathit{more} \supset T$. Furthermore, in a one-state interval, any PITL formula A is semantically equivalent to $\boxed{f}A$. Therefore, the interval σ satisfies the PITL formula $\boxed{f}(\mathit{more} \supset T)$ and hence also implication (23).

Now consider an interval σ which has more than one state and satisfies the NL^1 implication $\mathit{more} \supset T$. It follows that the interval also satisfies *more* and therefore the NL^1 formula T as well. The formula T, like any NL^1 formula, can only test at most the first two states of an interval, so all of σ's finite prefix subintervals with two or more states also satisfy T. It follows that every finite prefix subinterval of σ, including the initial one-state one, satisfies the implication $\mathit{more} \supset T$. Therefore, σ itself satisfies the $\boxed{f}$-formula $\boxed{f}(\mathit{more} \supset T)$ and hence also implication (23). $\square$

Lemma 8.4 *The 1-to-$\boxed{f}$ formulas are closed under conjunction. That is, if* $\models A \supset \boxed{f}A$ *and* $\models B \supset \boxed{f}B$*, then also* $\models (A \wedge B) \supset \boxed{f}(A \wedge B)$.

Proof Here is a simple semantic proof with four steps:

1	$\models$	$A \supset \boxed{f}A$	Assumption
2	$\models$	$B \supset \boxed{f}B$	Assumption
3	$\models$	$\boxed{f}A \wedge \boxed{f}B \equiv \boxed{f}(A \wedge B)$	PITL
4	$\models$	$A \wedge B \supset \boxed{f}(A \wedge B)$	1-3, Prop. $\square$

Theorem 8.5 below for 1-to-$\boxed{f}$ formulas is somewhat analogous to the earlier Theorem 4.5 concerning closure under $\Box$ for 2-to-1 formulas:

Theorem 8.5 *If A is 1-to-$\boxed{f}$, so is $\Box A$. That is, from the valid implication* $\models A \supset \boxed{f}A$ *follows the next one:*

$$\models \quad \Box A \quad \supset \quad \boxed{f}\Box A.$$

Proof Here is a short semantic proof:

1	$\models$	$A \supset \boxed{f}A$	Assumption
2	$\models$	$\Box A \supset \Box\boxed{f}A$	1, PTL
3	$\models$	$\Box\boxed{f}A \equiv \boxed{f}\Box A$	PITL
4	$\models$	$\Box A \supset \boxed{f}\Box A$	2, 3, Prop. $\square$

Now for any NL^1 formula T, the implication $\mathit{more} \supset T$ is also in NL^1. So we already have by Lemma 4.6 that the PTL formula $\Box(\mathit{more} \supset T)$ is 2-to-1. It follows from Lemma 8.3 and Theorem 8.5 that $\Box(\mathit{more} \supset T)$ is also 1-to-$\boxed{f}$. This includes the PTL formula $\mathit{stable}\,p$ defined in Table 1. The operator *stable* frequently occurs in applications of ITL, so it is convenient that a formula such as $\mathit{stable}\,p$ is both 2-to-1 and 1-to-$\boxed{f}$. The PTL formulas w, $\Box w$ and *finite* are likewise 2-to-1 and 1-to-$\boxed{f}$. The formula $\Box q$ already mentioned in the sample valid implication (21) is an example. We shortly make use of the formula $\Box q$ being 1-to-$\boxed{f}$.

Remark 8.6 Another simple example of formulas which are closed under conjunction and $\Box$ is the set of **1-to-$\Box$** formulas for any A for which $A \supset \Box A$ is valid. These are to a degree time-wise symmetric versions of the 1-to-$\boxed{f}$ ones. We do not further discuss here the theory of the 1-to-$\Box$ formulas but briefly encounter them later in Sect. 13.3 (when we analyse formula (92)).

8.2 Incremental version of ISEPI technique to extend a 2-to-1 formula rightward

We now provide an application of the 1-to-$\boxed{f}$ formulas just presented in Sect. 8.1. The main result here is Theorem 8.7, which provides a way to do the ISEPI technique of ***E**xtending rightward* the 2-to-1 formula $\boxed{f}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ for backward analysis. Theorem 8.7 concerns a semantic inference rule for ensuring that if B is ***E**xtended rightward* by a *suitable* PITL formula C, then the 2-to-1 formula $\boxed{f}\big((\mathit{fin}\, w) \supset \Diamond B\big)$, which contains B as a subformula, is itself ***E**xtended rightward* by the conjunction $w \wedge C$.

Theorem 8.7 *Let B be a PITL formula, C be a 1-to-$\boxed{f}$ formula and w be a state formula. Then the following semantic inference rule is sound:*

$$\models (B; C) \supset B \quad \Rightarrow \quad \models \big((\boxed{f}B'); (w \wedge C)\big) \supset \boxed{f}B', \tag{24}$$

where $\boxed{f}B'$ is simply the 2-to-1 formula $\boxed{f}\big((\mathit{fin}\, w) \supset \Diamond B\big)$.

The second implication in the semantic inference rule is identical to the following one which does not abbreviate $\boxed{f}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ as $\boxed{f}B'$:

$$\Big(\boxed{f}\big((\mathit{fin}\, w) \supset \Diamond B\big)\Big); (w \wedge C) \ \supset \ \boxed{f}\big((\mathit{fin}\, w) \supset \Diamond B\big). \tag{25}$$

Here is now the proof of Theorem 8.7:

Proof (Theorem 8.7) If we have the valid implication $\models (B; C) \supset B$, then the following instance of the same ISEPI technique for ***E**xtending rightward* is also valid:

$$\models \quad \big((\mathit{fin}\, w) \supset \Diamond B\big); (w \wedge C) \quad \supset \quad \big((\mathit{fin}\, w) \supset \Diamond B\big). \tag{26}$$

Here is a chain of valid implications to justify this from its sole required assumption $\models (B; C) \supset B$:

$$\begin{aligned} \big((\mathit{fin}\, w) \supset \Diamond B\big); (w \wedge C) \quad &\supset \quad \Big(\big((\mathit{fin}\, w) \supset \Diamond B\big) \wedge \mathit{fin}\, w\Big); C \\ &\supset \quad (\Diamond B); C \quad \supset \quad \Diamond(B; C) \quad \supset \quad \Diamond B \quad \supset \quad \big((\mathit{fin}\, w) \supset \Diamond B\big). \end{aligned} \tag{27}$$

From implication (26) and Theorem 8.1 then follows the validity of the next implication that is a variation of (26):

$$\models \quad \Big(\boxed{f}\big((\mathit{fin}\, w) \supset \Diamond B\big)\Big); \boxed{f}(w \wedge C) \quad \supset \quad \boxed{f}\big((\mathit{fin}\, w) \supset \Diamond B\big). \tag{28}$$

The formula $\boxed{f}(w \wedge C)$ can be re-expressed as $w \wedge \boxed{f}C$. This and our assumption that C is 1-to-$\boxed{f}$ permit us to obtain the next chain of valid implications:

$$w \ \wedge \ C \quad \supset \quad w \wedge \boxed{f}C \quad \supset \quad \boxed{f}(w \wedge C).$$

Consequently, the next implication is valid:

$$\models \quad \Big(\boxed{f}\big((\mathit{fin}\, w) \supset \Diamond B\big)\Big); (w \wedge C) \quad \supset \quad \Big(\boxed{f}\big((\mathit{fin}\, w) \supset \Diamond B\big)\Big); \boxed{f}(w \wedge C).$$

This together with the earlier one (28) ensures the validity of implication (25) and therefore the desired soundness of semantic inference rule (24). □

Like the semantic inference rule in the previous Theorem 8.1, semantic inference rule (24) provides an *incremental* way to obtain an instance of an ISEPI technique from a simpler variant of it.

We illustrate the use of Theorem 8.7 by taking as examples of w, B and C the propositional variable p, and the two PTL formulas $\neg p \wedge \Box q$ and $\Box q$, respectively. Here is the associate instance of $\models (B; C) \supset B$:

$$\models \quad (\neg p \wedge \Box q); \Box q \quad \supset \quad \neg p \wedge \Box q.$$

This was already presented as implication (21) at the beginning of this section to provide a simple example of the ISEPI technique of ***E**xtending rightward*. It was furthermore shown there to be valid. The formula $\Box q$ is 1-to-$\boxed{f}$ (and 2-to-1). Implication (21) can therefore serve as the first implication required by Theorem 8.7's semantic inference rule (24) to obtain the sample instance below of implication (25) for the ISEPI technique of ***E**xtending rightward* the 2-to-1 formula $\boxed{f}\big((\mathit{fin}\, p) \supset \Diamond(\neg p \wedge \Box q)\big)$:

$$\models \quad \Big(\boxed{f}\big((\mathit{fin}\, p) \supset \Diamond(\neg p \wedge \Box q)\big)\Big); (p \wedge \Box q) \quad \supset \quad \boxed{f}\big((\mathit{fin}\, p) \supset \Diamond(\neg p \wedge \Box q)\big).$$

The particular instance of C we later use in our analysis in Sect. 13 of mutual exclusion for Peterson's algorithm is the conjunction (70) of two formulas each of form $\Box(\mathit{more} \supset T)$, where T is in NL^1. Such formulas are conveniently both 2-to-1 and 1-to-$\boxed{f}$, as we already noted above in Sect. 8.1. The PTL formulas w, $\Box w$ and *finite* are also 1-to-$\boxed{f}$ (and additionally 2-to-1), so they can likewise be included in such conjunctions which serve as instances of C.

9 ISEPI parallel combining of 2-to-1 formulas for backward analysis

The next Lemma 9.1 involves an instance of the ISEPI technique already discussed in Sect. 4.1 for the ***P**arallel combining* of two suitable 2-to-1 formulas. It will be needed later on to obtain Corollary 11.1 in Sect. 11.2.3. That lemma concerns mutual exclusion and gives a way to establish that two processes operating in parallel are not simultaneously in their critical sections. We consider here the conjunction of two 2-to-1 $\boxed{f}$-formulas each of the form $\boxed{f}\big((\mathit{fin}\, w) \supset \Diamond B\big)$:

Lemma 9.1 *For any state formulas w and w' and PITL formulas B and B', suppose the following implication is valid:*

$$\models \quad \Diamond B \;\wedge\; \Diamond B' \quad \supset \quad \mathit{inf}. \tag{29}$$

Then the next implication for use with ISEPI **P**arallel combining *and backward analysis is also valid:*

$$\models \quad \boxed{f}\big((\mathit{fin}\, w) \supset \Diamond B\big) \;\wedge\; \boxed{f}\big((\mathit{fin}\, w') \supset \Diamond B'\big) \quad \supset \quad \Box\neg(w \wedge w'). \tag{30}$$

The assumption (29) states that the formulas B and B' cannot both occur in suffix subintervals of any finite interval.

Before proving Lemma 9.1, we discuss some illustrative examples of implications (29) and (30), respectively:

$$\models \ \Diamond(\mathit{skip} \wedge q) \ \wedge \ \Diamond(\mathit{skip} \wedge \neg q) \quad \supset \quad \mathit{inf} \tag{31}$$

$$\models \ \boxed{f}\big((\mathit{fin}\, p) \supset \Diamond(\mathit{skip} \wedge q)\big) \ \wedge \ \boxed{f}\big((\mathit{fin}\, p') \supset \Diamond(\mathit{skip} \wedge \neg q)\big) \quad \supset \quad \Box\neg(p \wedge p'). \tag{32}$$

The first implication (31) expresses that a finite interval with two or more states cannot have the propositional variable q being both true and false in the penultimate state (i.e., the one next to last). Observe that the antecedent of implication (31) is actually unsatisfiable, so the implication is vacuously true. The second implication (32) involves backward analysis to specify that any state with p true is immediately preceded by one with q true, and similarly each state with p' true is immediately preceded one with q false. This implication is comparable to the valid PTL$^-$ formula below:

$$\models \quad \Box(p \supset \circleddash q) \ \wedge \ \Box(p' \supset \circleddash \neg q) \quad \supset \quad \Box\neg(p \wedge p').$$

By "comparable", we mean here that an interval σ satisfies the PITL implication (32) iff the pair $(\sigma, 0)$ satisfies the PTL$^-$ formula.

Proof (Lemma 9.1) Here is a proof in steps which assumes the validity of (29):

– The next implication is valid by the assumed validity of (29) together with propositional reasoning:

$$\models \quad \neg\mathit{inf} \ \wedge \ \big((\mathit{fin}\, w) \supset \Diamond B\big) \ \wedge \ \big((\mathit{fin}\, w') \supset \Diamond B'\big) \quad \supset \quad (\neg\, \mathit{fin}\, w) \vee (\neg\, \mathit{fin}\, w').$$

This contains the subformula $\neg\mathit{inf}$ in the antecedent and so concerns behaviour in finite intervals.

– We then have the following chain of valid implications involving PTL-based reasoning about the operator *fin*:

$$\begin{aligned}(\neg\, \mathit{fin}\, w) \ \vee \ (\neg\, \mathit{fin}\, w') \quad &\supset \quad (\mathit{fin}\, \neg w) \ \vee \ (\mathit{fin}\, \neg w') \\ &\supset \quad \mathit{fin}(\neg w \ \vee \ \neg w') \quad \supset \quad \mathit{fin}\, \neg(w \wedge w').\end{aligned}$$

The valid implication below, which is suitable for ISEPI ***P****arallel combining*, subsequently results from combining the previous one and this chain:

$$\models \quad \neg\mathit{inf} \ \wedge \ \big((\mathit{fin}\, w) \supset \Diamond B\big) \ \wedge \ \big((\mathit{fin}\, w') \supset \Diamond B'\big) \quad \supset \quad \mathit{fin}\, \neg(w \wedge w').$$

– The following implication for ISEPI ***P****arallel combining*, which is about finite prefix subintervals, is consequently valid:

$$\models \quad \boxed{f}\neg\mathit{inf} \ \wedge \ \boxed{f}\big((\mathit{fin}\, w) \supset \Diamond B\big) \ \wedge \ \boxed{f}\big((\mathit{fin}\, w') \supset \Diamond B'\big) \quad \supset \quad \boxed{f}\, \mathit{fin}\, \neg(w \wedge w').$$

This is because for any PITL formulas $A_1, \dots, A_n$ and A', if the implication $(A_1 \wedge \cdots \wedge A_n) \supset A'$ is valid, so is $\big((\boxed{f}A_1) \wedge \cdots \wedge (\boxed{f}A_n)\big) \supset \boxed{f}A'$ (and indeed also $\big((\Box A_1) \wedge \cdots \wedge (\Box A_n)\big) \supset \Box A'$).

– The subformula $\boxed{f}\neg\mathit{inf}$ is trivially true because $\neg\mathit{inf}$ is semantically equivalent to *finite* and therefore true in all finite intervals. Furthermore, the subformula $\boxed{f}\, \mathit{fin}\, \neg(w \wedge w')$ and the PTL formula $\Box\neg(w \wedge w')$ are semantically equivalent. This is because for any interval σ, the set of the final states of σ's finite prefix subintervals (which $\boxed{f}\, \mathit{fin}\, \neg(w \wedge w')$ examines) and the set of σ's states (which $\Box\neg(w \wedge w')$ examines) are identical. Hence, the previous valid implication is semantically equivalent to our goal (30), which is therefore also valid. $\Box$

10 ISEPI iteration of 2-to-1 formulas for backward analysis

We now define some natural variants of 2-to-1 formulas which involve the iterative constructs chop-star and chop-plus instead of chop. It turns out that the 2-to-1 formula $\boxed{\mathrm{f}}\big((\mathit{fin}\ w) \supset \Diamond B\big)$ for backward analysis has special connections with such variants. The associated ISEPI technique for ***Iteration*** finds application in our analysis of mutual exclusion in Sects. 11–13 when we consider multiple requests by a process to a shared resource.

Definition 10.1 ($*$-to-1 formulas) Any PITL formula A for which the implication $A^* \supset A$ is valid is called a **$*$-to-1 formula**.

Definition 10.2 ($+$-to-1 formulas) Any PITL formula A for which the implication $A^+ \supset A$ is valid is called a **$+$-to-1 formula**.

Here is a brief summary of the three classes of formulas we have defined for sequential composition:

2-to-1 formulas	$\models (A; A) \supset A$
$*$-to-1 formulas	$\models A^* \supset A$
$+$-to-1 formulas	$\models A^+ \supset A.$

The three categories are all closed under conjunction and $\Box$ (e.g., see Lemmas 10.4 and 10.5 below for $+$-to-1 formulas). It is not hard to see that any formula A which is $+$-to-1 is 2-to-1:

Lemma 10.3 *Every $+$-to-1 formula is also 2-to-1.*

Proof Observe that for any PITL formula A, we have that $A; A$ implies A^+: $\models (A; A) \supset A^+$. Now if A is $+$-to-1, then A^+ in turn implies A. Hence, by transitivity, the formula $A; A$ implies A as well, so the formula A is indeed 2-to-1.

Here is a corresponding chain of two valid implications:

$$A; A \quad \supset \quad A^+ \quad \supset \quad A. \qquad \Box$$

Likewise, any formula which is $*$-to-1 is also 2-to-1 and $+$-to-1 as well because of the valid PITL implications $\models (A; A) \supset A^*$ and $\models A^+ \supset A^*$, which respectively yield the following two chains of valid implications:

$$\begin{array}{ccccc} A; A & \supset & A^* & \supset & A \\ A^+ & \supset & A^* & \supset & A. \end{array}$$

However, the three categories are by no means identical. Below are sample formulas which illustrate this point:

	2-to-1	$*$-to-1	$+$-to-1
finite	✓		
p	✓		✓
empty	✓	✓	✓

The reason not every 2-to-1 formula is also $+$-to-1 is because of the situation in *infinite time*. Consider the 2-to-1 formula *finite*. Any infinite interval satisfies finite^+ but not *finite*. The same reasoning holds if we replace finite^+ by finite^*, so the formula *finite* is therefore also not $*$-to-1.

All 2-to-1 $\boxed{\mathrm{f}}$-formulas are also $+$-to-1 as is later shown in Theorem 10.7. Furthermore, the $\boxed{\mathrm{f}}$-formulas of the form $\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$, which we already considered for backward analysis, are subsequently shown in a meaningful formal sense to be nearly members of the class of $*$-to-1 formulas. Certain instances of such formulas can then be profitably used in our analysis of mutual exclusion when we want to compositionally analyse the behaviour of a process making multiple requests to a shared resource.

Let us now discuss further properties of 2-to-1 and $+$-to-1 formulas. We make some use of $*$-to-1 formulas as well and later mention in Sect. 15.5 their connection with our earlier work on compositionality in ITL.

Lemma 10.4 *For any $+$-to-1 formulas A and B, their conjunction $A \wedge B$ is $+$-to-1 as well. That is, if $\models A^+ \supset A$ and $\models B^+ \supset B$, then also $\models (A \wedge B)^+ \supset (A \wedge B)$.*

Proof The formula $(A \wedge B)^+$ implies both A^+ and B^+ and consequently also their conjunction $A^+ \wedge B^+$. Our assumption that A and B are both $+$-to-1 then guarantees that this implies $A \wedge B$. Here is a corresponding chain of valid implications:

$$(A \wedge B)^+ \quad \supset \quad A^+ \wedge B^+ \quad \supset \quad A \wedge B. \qquad \Box$$

Lemma 10.5 *If A is $+$-to-1, so is $\Box A$. That is, if $\models A^+ \supset A$, then also $\models (\Box A)^+ \supset \Box A$.*

Proof Let σ be an interval which satisfies $(\Box A)^+$. Our proof will check that each suffix subinterval of σ, including σ itself, satisfies A^+ and hence also A. Therefore, σ satisfies $\Box A$. There are two cases to consider which depend on whether the number of iterations is finite or infinite:

- **The chop-plus involves a finite number of sequential iterations of $\Box A$:** It follows that σ satisfies the PITL formula $(\Box A)^{\star}{}^\frown \Box A$ containing strong versions of chop and chop-star. Each of σ's suffix subintervals can then be shown to satisfy A^+ and so also A, since A is $+$-to-1. Hence, σ satisfies $\Box A$.
- **The chop-plus involves ω sequential iterations of $\Box A$ (so the interval is infinite):** We can readily check that each suffix subinterval σ' of σ satisfies A^ω and hence also A^+. Therefore, σ' satisfies A itself because A is $+$-to-1. Consequently, σ satisfies $\Box A$. $\Box$

The next theorem is the converse of Lemma 10.3, but necessarily restricted to finite intervals for reasons given shortly:

Lemma 10.6 *If A is 2-to-1, then it is $+$-to-1 for* finite time*, that is, the implication below is valid:*

$$\mathit{finite} \quad \supset \quad (A^+ \supset A). \tag{33}$$

Proof Let σ be a finite interval satisfying A^+. We want to show that σ satisfies A as well. Now for some natural number $k \geq 1$, σ satisfies k instances of A sequentially combined with $k-1$ chops between then. For example, if k is 3, then σ satisfies $A; A; A$. Note that in finite intervals, strong and weak chop have the same semantics. We do induction on the number of chops in the formula $A; \ldots ; A$ and employ the assumption that A is 2-to-1 to demonstrate that σ satisfies A itself. Therefore, A is indeed +-to-1 for finite-time intervals and hence implication (33) is valid. $\square$

We already pointed out above that the formula *finite* is an example of a 2-to-1 formula which is not +-to-1 in infinite time. This explains Lemma 10.6's requirement about finite time. However, the next Theorem 10.7 demonstrates that all $\boxed{\mathrm{f}}$-formulas which are 2-to-1 formulas are also +-to-1 even for infinite time. Such formulas are moreover later used in our analysis of mutual exclusion in Sects. 11–13 for multiple requests by a process to a shared resource.

Theorem 10.7 *Any 2-to-1 formula $\boxed{\mathrm{f}}B$ is also +-to-1 for all intervals, including infinite ones:*

$$\models \quad (\boxed{\mathrm{f}}B)^{+} \quad \supset \quad \boxed{\mathrm{f}}B. \tag{34}$$

The proof of Theorem 10.7 is given shortly.

Note that in contrast to a 2-to-1 $\boxed{\mathrm{f}}$-formula, a 2-to-1 $\Box$-formula, which looks at suffix subintervals rather than the prefix ones examined by $\boxed{\mathrm{f}}$, is not necessarily +-to-1. We can take the formula $\Box$*finite* to serve as an example of this. It is semantically equivalent to *finite*, which we already pointed out is not +-to-1 in infinite intervals.

We use Theorem 10.7 to provide an ISEPI technique for ***Iteration*** with chop-plus. This has the form $\models A^{+} \supset A$. The theorem ensures that the implication is indeed valid if we take A to be the 2-to-1 formula $\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ for backward analysis:

$$\models \quad \Big(\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)\Big)^{+} \quad \supset \quad \boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big). \tag{35}$$

Before proving Theorem 10.7, we present a lemma which concerns a semantic inference rule used in the proof:

Lemma 10.8 *For any PITL formulas C and C', the next semantic inference rule is sound:*

$$\models (\mathit{finite} \wedge C^{+}) \supset \boxed{\mathrm{f}}C' \quad \Rightarrow \quad \models C^\frown C^{\bigstar} \supset \boxed{\mathrm{f}}C'. \tag{36}$$

Proof We start by assuming the validity of the implication $(\mathit{finite} \wedge C^{+}) \supset \boxed{\mathrm{f}}C'$. Our goal is to show from this that any interval σ which satisfies $C^\frown C^{\bigstar}$ also satisfies $\boxed{\mathrm{f}}C'$. Let σ' be any finite prefix subinterval of σ (including σ itself if it is finite). Our proof will show that any such σ' satisfies C' and hence σ itself satisfies $\boxed{\mathrm{f}}C'$. Now σ satisfies $C^\frown C^{\bigstar}$, so σ' is contained in a finite prefix subinterval σ'' which likewise satisfies $C^\frown C^{\bigstar}$ and so also both C^{+} and $\mathit{finite} \wedge C^{+}$. Hence by the assumption, σ'' also satisfies $\boxed{\mathrm{f}}C'$, so its prefix subinterval σ' satisfies C'. It follows that all of the finite prefix subintervals of σ indeed satisfy C', and therefore σ itself satisfies $\boxed{\mathrm{f}}C'$. $\square$

We now supply Theorem 10.7's proof:

Proof (Theorem 10.7) **Case for finite time**: Lemma 10.6 ensures that (33) is valid for finite time for any 2-to-1 formula, so the next instance of (33), which is moreover a variant of (34), is valid as well:

$$\big(\mathit{finite} \ \wedge \ (\boxed{\mathrm{f}}B)^{+}\big) \quad \supset \quad \boxed{\mathrm{f}}B.$$

Case for infinite time: Our goal here is to show the validity of the next implication:

$$\big(\mathit{inf} \ \wedge\ (\boxed{\mathrm{f}}B)^+\big) \quad\supset\quad \boxed{\mathrm{f}}B.$$

Let A denote $\boxed{\mathrm{f}}B$. We re-express $\mathit{inf} \wedge A^+$:

$$\models \quad \mathit{inf} \wedge A^+ \quad\equiv\quad (\mathit{inf} \wedge A^\star) \ \vee\ \big(A^{\star\frown}(\mathit{inf} \wedge A)\big).$$

The subformula $A^{\star\frown}(\mathit{inf} \wedge A)$ is re-expressible as $\mathit{inf} \wedge (A^{\star\frown}A)$, so our semantic proof can be divided into two parts:

$$\models \quad \mathit{inf} \wedge (A^{\star\frown}A) \quad\supset\quad A \tag{37}$$

$$\models \quad \mathit{inf} \wedge A^\star \quad\supset\quad A. \tag{38}$$

Subcase for (37)*:* We already have $A^+ \supset A$ valid for finite time. Therefore, the chain of implications below is valid since A is 2-to-1 and $A^\star$ occurs in the finite left of $^\frown$:

$$\begin{aligned} A^{\star\frown}A \quad&\supset\quad (\mathit{empty} \vee A^+)^\frown A \quad\supset\quad (\mathit{empty}^\frown A) \vee (A^{+\frown}A) \\ &\supset\quad A \vee (A^\frown A) \quad\supset\quad A \vee (A;A) \quad\supset\quad A \vee A \quad\supset\quad A. \end{aligned}$$

Hence, formula (37) is valid.

Subcase for (38)*:* Recall that A denotes here the $\boxed{\mathrm{f}}$-formula $\boxed{\mathrm{f}}B$. Furthermore, the PITL equivalence $\boxed{\mathrm{f}}B \equiv \boxed{\mathrm{f}}\boxed{\mathrm{f}}B$ is valid (much like the valid PTL equivalence $\models \Box p \equiv \Box\Box p$). Hence, we have $\models A \equiv \boxed{\mathrm{f}}A$. Lemma 10.8 permits us to take an instance of the sound semantic inference rule (36) with C and C' both A:

$$\models (\mathit{finite} \wedge A^+) \supset \boxed{\mathrm{f}}A \quad\Rightarrow\quad \models A^\frown A^\star \supset \boxed{\mathrm{f}}A.$$

We then replace each $\boxed{\mathrm{f}}A$ by A using $\models A \equiv \boxed{\mathrm{f}}A$:

$$\models (\mathit{finite} \wedge A^+) \supset A \quad\Rightarrow\quad \models A^\frown A^\star \supset A.$$

We already have the validity of $\big(\mathit{finite} \wedge (\boxed{\mathrm{f}}B)^+\big) \supset \boxed{\mathrm{f}}B$ from the case for finite time. This is re-expressed using A instead of $\boxed{\mathrm{f}}B$ to obtain $\models (\mathit{finite} \wedge A^+) \supset A$. The semantic inference rule then yields that the implication $A^\frown A^\star \supset A$ is also valid. In infinite time, $A^\star$ and $A^\frown A^\star$ are semantically equivalent, so our goal (38) is valid.

The combination of (37) and (38) ensures (34) is valid for infinite intervals.

Our proof's two cases for finite and infinite intervals then yield (34)'s validity for all intervals. $\Box$

Remark 10.9 Let us briefly note without proof some interesting facts not needed here. Recall from Lemma 4.3 that any formula w, T or $\Diamond_{\mathrm{i}}C$ is 2-to-1. They are in fact also +-to-1 even for infinite time. Also, if B is 2-to-1, so are the two formulas $w \supset (B \wedge \mathit{fin}\, w)$ and $\Box\big(w \supset (B \wedge \mathit{fin}\, w)\big)$. Reflection helps ensure that $\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset (B \wedge w)\big)$ is as well. This $\boxed{\mathrm{f}}$-formula is +-to-1 even for infinite time. Furthermore, if the formula $B^* \supset B$ is valid (i.e., B is $*$-to-1), then so is $C^* \supset C$, where C is any of these three formulas.

10.1 Zero or more sequential iterations of a 2-to-1 formula

When we later compositionally analyse how a process can make multiple accesses to a shared resource, it is natural to include the case where no accesses are performed. So it would be convenient in such circumstances to use for backward analysis some $*$-to-1 formulas introduced in Definition 10.1 at the beginning of this Sect. 10. Now every $\boxed{\mathrm{f}}$-formula $\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ has already been shown to be 2-to-1 (Theorem 6.1) and therefore also $+$-to-1 (Theorem 10.7 and implication (35)). However, we now show that $\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ is unfortunately not necessarily $*$-to-1. Nevertheless, we offer a workaround which is nearly $*$-to-1 and quite suitable for using as an ISEPI technique for ***Iteration*** when we look at mutual exclusion in Sects. 11–13.

Let us now present two lemmas concerning the relationship between instances of the $\boxed{\mathrm{f}}$-formula $\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ for backward analysis and the class of $*$-to-1 formulas:

Lemma 10.10 *Not every formula* $\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ *is $*$-to-1.*

Proof We exhibit such a $\boxed{\mathrm{f}}$-formula and an interval which satisfies the weak chop-star of the formula but not the formula itself. Consider the previous $\boxed{\mathrm{f}}$-formula (16), which is reproduced below:

$$\boxed{\mathrm{f}}\big((\mathit{fin}\, p) \supset \Diamond(\mathit{skip} \wedge q)\big).$$

Let σ be a one-state interval with the propositional variable p true. We show that σ falsifies the next implication:

$$\Big(\boxed{\mathrm{f}}\big((\mathit{fin}\, p) \supset \Diamond(\mathit{skip} \wedge q)\big)\Big)^{*} \quad \supset \quad \boxed{\mathrm{f}}\big((\mathit{fin}\, p) \supset \Diamond(\mathit{skip} \wedge q)\big). \tag{39}$$

Now σ, like every one-state interval, trivially satisfies any weak chop-star formula A^{*}. Therefore, σ satisfies (39)'s antecedent. In a one-state interval, the consequent of implication (39) reduces to the PTL formula $p \supset (\mathit{skip} \wedge q)$. However, the interval σ, which sets p to true, cannot satisfy the subformula *skip* since that requires at least two states to be present. □

The earlier sample $\boxed{\mathrm{f}}$-formula (18) is also not $*$-to-1. This is because in a one-state interval, $\boxed{\mathrm{f}}\big((\mathit{fin}\, p) \supset \Diamond\neg p\big)$ simplifies to $p \supset \neg p$, which is falsified if the interval sets p to *true*.

The next lemma shows how instances of the $\boxed{\mathrm{f}}$-formula $\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ can always be regarded as "almost" $*$-to-1 if we require w to initially equal *false*:

Lemma 10.11 *For any formula* $\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$*, the next implication is valid:*

$$\models \quad \neg w \;\wedge\; \Big(\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)\Big)^{*} \quad \supset \quad \boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big). \tag{40}$$

Proof Let C denote $\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$. We already used Theorem 10.7 to show that C is $+$-to-1 (i.e., see the earlier valid implication (35)). Therefore, we have the following:

$$\models \quad C^{+} \quad \supset \quad C. \tag{41}$$

This is the ISEPI technique of ***Iteration*** for chop-plus, as we previously mentioned with regard to (35). We can also show the following:

$$\models \quad \mathit{empty} \wedge \neg w \quad \supset \quad C. \tag{42}$$

This is because in a one-state interval any formula $\boxed{\mathrm{f}}A$ is semantically identical to its operand A. In particular, C is identical to $(\mathit{fin}\, w) \supset \Diamond B$, which in a one-state interval further simplifies to $w \supset B$. So a one-state interval which satisfies $\neg w$ also satisfies C.

Let us now look at merging the two implications (41) and (42) into a single one from which we can later on obtain that C is "almost" $*$-to-1:

$$\models \quad C^{+} \vee (\mathit{empty} \wedge \neg w) \quad \supset \quad C. \tag{43}$$

Simple propositional reasoning ensures that the conjunction $\neg w \wedge (\mathit{empty} \vee C^{+})$ implies the antecedent $C^{+} \vee (\mathit{empty} \wedge \neg w)$ in (43):

$$\models \quad \neg w \wedge (\mathit{empty} \vee C^{+}) \quad \supset \quad C^{+} \vee (\mathit{empty} \wedge \neg w).$$

We can then combine this and (43) using further straightforward propositional reasoning:

$$\models \quad \neg w \wedge (\mathit{empty} \vee C^{+}) \quad \supset \quad C.$$

The PITL equivalence $A^{*} \equiv (\mathit{empty} \vee A^{+})$ is valid for any formula A. We use it to simplify $\mathit{empty} \vee C^{+}$ to obtain the next valid implication:

$$\models \quad \neg w \ \wedge \ C^{*} \quad \supset \quad C.$$

This is in fact identical to our goal (40). □

For the convenience of readers, Table 5 lists the ISEPI techniques presented in Sects. 6–10 specifically for use with the 2-to-1 formulas for backward analysis. The table can be used for reference when we apply the techniques to mutual exclusion in the next three Sects. 11–13.

11 Analysis of an abstract mutual exclusion algorithm

In this section and the next two, we consider how to apply compositional backward analysis and the previously introduced 2-to-1 formulas of the form $\boxed{\mathrm{f}}((\mathit{fin}\, w) \supset \Diamond B)$ to *mutual exclusion* and Peterson's algorithm [64]. These have provided us with a rich and stimulating initial testing ground for developing and experimenting with our ideas about backward analysis in ITL. They together also serve as a proof-of-concept of the approach and at least at present are a rather inseparable part of our exploration of time symmetry. We certainly do not claim that our research has reached a stage where it is ready to be deployed in practical problems.

Figure 1 shows a version of Peterson's algorithm. One reason for looking at it is because it is a quite elegant and popular example of mutual exclusion and seems to serve as a kind of benchmark for formal techniques. We will have much more to say about Peterson's algorithm in Sect. 13 where we formalise in PITL a version of it with two **concrete processes** $\boldsymbol{P_0}$ and $\boldsymbol{P_1}$. However, our analysis of mutual exclusion initially mostly focuses on a more abstract and higher-level

Introduction (first variant): Formula (19) in Sect. 7:

$$\models \quad \Box\neg w \quad \supset \quad \boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big).$$

Introduction (second variant): Formula (20) in Sect. 7:

$$\models \quad \Box(\mathit{more} \supset \neg w) \;\wedge\; (\mathit{inf} \vee \Diamond B) \quad \supset \quad \boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big).$$

Sequential combining: (See Theorem 6.1)

$$\Big(\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)\Big); \boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big) \quad \supset \quad \boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big).$$

Extending rightward: Formula (25) in Sect. 8.2:

$$\models \quad \Big(\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)\Big); (w \wedge C) \;\supset\; \boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big),$$

where C is a 1-to-$\boxed{\mathrm{f}}$ formula and extends B rightward (i.e., $\models (B; C) \supset B$).

Parallel combining: Formula (30) in Sect. 9:

$$\models \quad \boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big) \;\wedge\; \boxed{\mathrm{f}}\big((\mathit{fin}\, w') \supset \Diamond B'\big) \quad \supset \quad \Box\neg(w \wedge w'),$$

where $\models (\Diamond B \wedge \Diamond B') \supset \mathit{inf}$.

Iteration (version for +-to-1 formula): Formula (35) in Sect. 10:

$$\models \quad \Big(\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)\Big)^{+} \quad \supset \quad \boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big).$$

Iteration (version for "almost" $*$-to-1 formula): Formula (40) in Sect. 10.1:

$$\models \quad \neg w \;\wedge\; \Big(\boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)\Big)^{*} \quad \supset \quad \boxed{\mathrm{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big).$$

Table 5 Summary of ISEPI techniques for 2-to-1 formulas for backward analysis

algorithm with two **abstract processes Q_0** and **Q_1**. It contains shared aspects of several algorithms and proofs. This is in part because our study of compositionality in Peterson's algorithm has helped us see benefits of applying time symmetry to formalising in PITL some abstract issues arising in mutual exclusion. In the future we would of course like to gain more experience by considering other applications and also a range of modelling assumptions, but our research has not yet progressed to this stage.

Let us now review the notion of mutual exclusion. It is one way to ensure that multiple processes safely access a shared resource. Examples of it include cash machines accessing a single bank account and processes utilising a shared printer.

Process P_0	Process P_1
1. $noop_0$;	1. $noop_1$;
2. $flag_0 := 1$;	2. $flag_1 := 1$;
3. $turn := 1$;	3. $turn := 0$;
4. $await(flag_1 = 0 \vee turn = 0)$;	4. $await(flag_0 = 0 \vee turn = 1)$;
5. $noop_0$; (critical section)	5. $noop_1$; (critical section)
6. $flag_0 := 0$;	6. $flag_1 := 0$;
7. $noop_0$	7. $noop_1$

Initially $flag_0 = flag_1 = 0$. The starting value of *turn* is unimportant.

Each statement $noop_i$ denotes a "no-operation" which does no assignments.

Fig. 1 A version of Peterson's algorithm with two concrete processes P_0 and P_1

Here is the general structure of a single access by one abstract process:

(a) *Noncritical* section;
(b) *Request* exclusive right to resource;
(c) *Critical* section with exclusive access; (44)
(d) *Release* exclusive right to resource;
(e) *Noncritical* section.

Taubenfeld's textbook [76] gives English-language proofs of mutual exclusion for various algorithms, starting with Peterson's (as is indeed often the case in textbooks). Unlike mutual exclusion proofs in conventional point-based temporal logic such as those by Pnueli [66] and Kröger and Merz [37], ours does not use a comprehensive set of labels for all relevant program steps. This reflects the quite different nature of point- and interval-based approaches, which can be respectively referred to as *endogenous* and *exogenous*. We have more to say about this later on in Sect. 15.2.

The abstract processes Q_0 and Q_1 and the associated analysis capture some general features of mutual exclusion which apply to many concrete algorithms, not just Peterson's. *A major benefit of the abstract framework is that it involves a fairly direct application of the ISEPI techniques for 2-to-1 formulas for backward analysis presented earlier in Sects. 6–10.*

At first glance, it might seem easier to formalise something specific and tangible such as the processes P_0 and P_1 in Peterson's algorithm than to formalise the more abstract processes Q_0 and Q_1. However, before we can reason about the concrete Peterson processes P_0 and P_1 in PITL, their individual statements need to be expressed as PITL formulas. This requires some further explanation and justification about the modelling assumptions used for concurrency and so is deferred until later in Sect. 13. Furthermore, any analysis dealing just with the concrete processes P_0 and P_1 in Peterson's algorithm is of course much more limited than an analogous one about a higher-level framework for abstract processes Q_0 and Q_1 which can be adapted to many algorithms, including Peterson's. In distinct contrast to the situation with modelling and reasoning about Peterson's algorithm in PITL, our ISEPI techniques for 2-to-1 formulas presented in Sects. 6–10 can be almost immediately used to provide a fairly concise and high-level analysis of the abstract processes Q_0 and Q_1.

Nevertheless, it can be confusing to work just with the abstract algorithm without having any motivation provided by a concrete one. Therefore, Fig. 1 shows our processes P_0 and P_1 for Peterson's algorithm. We will only discuss certain aspects of them here which help with understanding the abstract processes Q_0 and Q_1 and the associated correctness formulas.

The concrete processes P_0 and P_1 in Fig. 1 together have three program variables flag_0, flag_1 and turn with values in $\{0, 1\}$. To stay propositional, let 0, 1 and $=$ stand for *false*, *true* and $\equiv$, respectively. Both flag_0 and flag_1 are initialised to 0, but *turn*'s starting value is unimportant. The statements noop_0 and noop_1 are simply "no-operations" or "no-ops" during which time the processes do not assign their respective variables values. Real processes would likely examine and modify other variables besides flag_i and *turn*, but we ignore them here. The PITL semantics of noop_i and other statements in each concrete process P_i are given in Sect. 13. We do not need to know their details in our analysis of the abstract processes Q_0 and Q_1.

11.1 Model with the abstract processes Q_0 and Q_1

The abstract processes Q_0 and Q_1 are modelled as executing together with initialisation, and expressed as $\bigwedge_{i\in\{0,1\}}(\mathit{init}_i \wedge Q_i)$. Here init_i is some state formula for initialising Q_i's variables. The analysis assumes that each process Q_i has an *auxiliary* boolean variable cs_i true exactly when Q_i is in its critical section and that init_i sets cs_i to false:

$$\models \quad \mathit{init}_i \quad \supset \quad \neg \mathit{cs}_i. \tag{45}$$

Section 13 gives a concrete instance $\boldsymbol{P'_i}$ in (71) for each abstract process Q_i. We use P'_i to serve as a variant of Peterson's algorithm with additional formulas describing the behaviour of cs_i. In Sect. 13 we likewise define in (76) the concrete version $\boldsymbol{\mathit{pinit}_i}$ of init_i to be the state formula $\mathit{flag}_i = 0 \wedge \neg \mathit{cs}_i$. As is already noted in Fig. 1, the initial value of the shared writable variable *turn* is not important. For our analysis of the abstract algorithm, we need for each init_i that the implication (45) is valid. This is certainly the case with the concrete formula $\mathit{flag}_i = 0 \wedge \neg \mathit{cs}_i$, as the next valid implication demonstrates:

$$\models \quad \mathit{flag}_i = 0 \ \wedge \ \neg \mathit{cs}_i \quad \supset \quad \neg \mathit{cs}_i.$$

Our goal here is to have two general **Abstract Assumptions** which together with the valid implication (45) (i.e., $\models \mathit{init}_i \supset \neg \mathit{cs}_i$) suffice to ensure that the abstract processes Q_0 and Q_1 are never simultaneously in their critical sections (line (c) in (44)). The Abstract Assumptions will be shortly introduced in Table 6 in the next Sect. 11.2.

We now briefly summarise our goal concerning mutual exclusion. Our aim is to prove $\neg(\mathit{cs}_0 \wedge \mathit{cs}_1)$ is always true, as stated in the next implication:

$$\models \quad \bigwedge_{i\in\{0,1\}} (\mathit{init}_i \wedge Q_i) \quad \supset \quad \Box\neg(\mathit{cs}_0 \wedge \mathit{cs}_1).$$

The Abstract Assumptions will indeed be shown in Sect. 11.2 to be sufficient to guarantee the validity of this.

First Abstract Assumption for each $i \in \{0,1\}$

$$\models \quad \mathit{init}_i \ \wedge\ Q_i \quad \supset \quad \mathit{AbsSafe}_i \ \wedge\ \mathit{fin}\, \mathit{init}_i, \tag{46}$$

where $\mathit{AbsSafe}_i$ is defined as $\boxed{f}((\mathit{fin}\ cs_i) \ \supset\ \Diamond D_i)$ for some D_i.

Second Abstract Assumption

$$\models \quad \Diamond D_0 \ \wedge\ \Diamond D_1 \quad \supset \quad \mathit{inf} \tag{47}$$

Table 6 The first and second abstract assumptions

Our abstract analysis can also be generalised to *multiple accesses* to the shared resource by processes as is discussed later in Sect. 11.3. Properties besides mutual exclusion such as freedom from deadlock are not considered here, but they could be shown with our compositional techniques for liveness [50, 51] using 2-to-1 formulas for forward analysis such as $\Box(p \supset \Diamond q)$.

11.2 Basic mutual exclusion for the abstract processes

We now introduce the two **Abstract Assumptions** (46) and (47). They concern mutual exclusion for the abstract processes Q_0 and Q_1 and describe some temporal behaviour. Table 6 shows these Abstract Assumptions, which are individually referred to as the **First Assumption** (46) and the **Second Assumption** (47). The following 2-to-1 formula $\boldsymbol{\mathit{AbsSafe}_i}$ is used in the First Assumption (46) to describe a safety property for Q_i:

$$\mathit{AbsSafe}_i \quad \widehat{=} \quad \boxed{f}((\mathit{fin}\ cs_i) \ \supset\ \Diamond D_i). \tag{48}$$

This formula $\mathit{AbsSafe}_i$ is an instance of the 2-to-1 formula $\boxed{f}((\mathit{fin}\, w) \supset \Diamond B)$ introduced in Sect. 6 for compositional backward analysis. Our experience is that many readers have trouble grasping the intuition behind the Abstract Assumptions (46) and (47) and in particular the role of the abstract formulas D_i and $\mathit{AbsSafe}_i$. We therefore first discuss concrete instances of D_i and $\mathit{AbsSafe}_i$ and only then give a general explanation of the Abstract Assumptions. One aim here is to preview some aspects of our analysis of Peterson's algorithm in Sect. 13.

11.2.1 Justification of the first abstract assumption

As already noted above in Sect. 11.1, we later on define a variant process $\boldsymbol{P'_i}$ of each process P_i in (71) in Sect. 13 (more precisely, in Sect. 13.2). The purpose of P'_i is to ensure that the auxiliary variable cs_i is indeed true exactly when the process P_i is in its critical section. The later definition of process P'_i in (71) is reproduced below for the convenience of readers in order to assist in our explanation of D_i and $\mathit{AbsSafe}_i$:

$$\begin{array}{rcl} P'_i & \widehat{=} & \big(P_i\text{'s lines 1–3} \ \wedge\ \mathit{stable}\ cs_i\big); \\ & & \big(P_i\text{'s line 4} \ \wedge\ cs_i <\!\!\sim \mathit{true}\big); \\ & & \big(P_i\text{'s line 5} \ \wedge\ cs_i <\!\!\sim \mathit{false}\big); \\ & & \big(P_i\text{'s lines 6–7} \ \wedge\ \mathit{stable}\ cs_i\big). \end{array}$$

Recall the derived constructs *stable* and padded temporal assignment ($\lessdot\!\sim$) given in Table 1. The actual concrete instance of D_i defined and used in Sect. 13.3 for our analysis of Peterson's algorithm is the formula $\boldsymbol{E_i}$ given below (see (73)):

$$E_i \quad \widehat{=} \quad \mathit{flag}_i = 1 \ \wedge\ \mathit{turn} = 1 - i \ \wedge\ \Diamond(\mathit{flag}_{1-i} = 0 \ \vee\ \mathit{turn} = i) \ \wedge\ \mathit{nochange}_i.$$

Here $\mathit{nochange}_i$ is defined later in Sect. 13.1 as part of the PITL semantics of the statements in Peterson's algorithm. The formula $\mathit{nochange}_i$ specifies that the process P_i is not assigning any values to flag_i and turn. Further details about $\mathit{nochange}_i$ are not needed for the moment.

Let us consider a concrete formula $\boldsymbol{\mathit{PeteSafe}_i}$ which summarises some key behaviour observable when process P_i' is its critical section. We later formally define $\mathit{PeteSafe}_i$ as (74) in Sect. 13.3 to be a concrete version of $\mathit{AbsSafe}_i$ with the concrete instance E_i of the abstract formula D_i. The definition is reproduced below for the convenience of readers:

$$\mathit{PeteSafe}_i \quad \widehat{=} \quad \boxed{\scriptstyle f}\big((\mathit{fin}\ cs_i) \ \supset\ \Diamond E_i\big).$$

We now informally show that if the first state of an interval σ satisfies the initialisation formula $\mathit{flag}_i = 0 \wedge \neg cs_i$ for process P_i' and the interval σ itself satisfies the formula P_i', then each finite prefix subinterval $\boldsymbol{\sigma'}$ of σ satisfies the next concrete formula, and hence σ satisfies $\mathit{PeteSafe}_i$:

$$(\mathit{fin}\ cs_i) \quad \supset \quad \Diamond E_i.$$

Here is the main goal expressed as an implication which the overall interval σ satisfies:

$$\mathit{flag}_i = 0 \ \wedge\ \neg cs_i \ \wedge\ P_i' \quad \supset \quad \boxed{\scriptstyle f}\big((\mathit{fin}\ cs_i) \ \supset\ \Diamond E_i\big).$$

Suppose the finite prefix subinterval σ' of σ satisfies $\mathit{fin}\ cs_i$. It follows that cs_i is true in the last state of σ' and hence process P_i' is in its critical section during this state. Inspection of our definitions of P_i' and P_i reveal that whenever the variable cs_i is true, this is preceded by the execution of the following sequence of statements in P_i:

$$\mathit{flag}_i := 1; \mathit{turn} := 1 - i; \mathit{await}(\mathit{flag}_{1-i} = 0 \ \vee\ \mathit{turn} = i).$$

In the state immediately after the first two of these statements, the formula $\mathit{flag}_i = 1 \wedge \mathit{turn} = 1-i$ is true. That state in fact marks the start of some suffix subinterval $\boldsymbol{\sigma''}$ in σ' which satisfies E_i. This is because when the process manages to enter its critical section following the execution of the *await* statement, the test $\mathit{flag}_{1-1} = 0 \vee \mathit{turn} = i$ must have succeeded. So the state with $cs_i = 1$ will be preceded by a state with $\mathit{flag}_{1-1} = 0 \vee \mathit{turn} = i$. That state is itself preceded by a state with $\mathit{flag}_i = 1 \wedge \mathit{turn} = 1-i$. Furthermore, from the moment that $\mathit{flag}_i = 1 \wedge \mathit{turn} = 1-i$ is true in that state until when the process leaves its critical section, the process does not assign either flag_i or turn, so the formula $\mathit{nochange}_i$ holds. Therefore, the suffix subinterval σ'' of σ' (itself a finite prefix of σ) satisfies the following formulas:

$$\mathit{flag}_i = 1 \wedge \mathit{turn} = 1 - i \qquad \Diamond(\mathit{flag}_{1-1} = 0 \vee \mathit{turn} = i) \qquad \mathit{nochange}_i.$$

Consequently, σ'' satisfies E_i, which is simply the conjunction of these, and σ' itself satisfies $\Diamond E_i$ and hence also $(\mathit{fin}\ cs_i) \supset \Diamond E_i$. It follows that the overall interval σ satisfies the formula $\boxed{f}\big((\mathit{fin}\ cs_i) \supset \Diamond E_i\big)$, which is the same as $\mathit{PeteSafe}_i$. Therefore, the following concrete instance of the First Assumption (46) for the concrete instance P_i' of the abstract process Q_i is valid:

$$\models \quad \mathit{flag}_i = 0 \ \wedge\ \neg cs_i \ \wedge\ P_i' \quad \supset \quad \boxed{f}\big((\mathit{fin}\ cs_i) \supset \Diamond E_i\big) \ \wedge\ \mathit{fin}(\mathit{flag}_i = 0 \ \wedge\ \neg cs_i). \tag{49}$$

The formula $\boxed{f}\big((\mathit{fin}\ cs_i) \supset \Diamond E_i\big)$ is identical to the concrete version $\mathit{PeteSafe}_i$ of $\mathit{AbsSafe}_i$. In (49) we also mention the concrete formula $\mathit{fin}(\mathit{flag}_i = 0 \wedge \neg cs_i)$ about the last state if the process P_i' terminates. This formula, which is a concrete version of the formula $\mathit{fin}\ \mathit{init}_i$ in the First Assumption (46), is easy to check from the behaviour of P_i'. In our analysis of Peterson's algorithm in Sect. 13, we let the concrete instance pinit_i of the abstract initialisation formula init_i denote this conjunction $\mathit{flag}_i = 0 \wedge \neg cs_i$ (as already noted in Sect. 11.1).

The First Assumption (46) is an abstracted version of implication (49), where we leave the fine points of Q_i, D_i and the 2-to-1 formula $\mathit{AbsSafe}_i$ largely unspecified. The formula $\mathit{fin}\ \mathit{init}_i$ in the First Assumption later helps to iterate Q_i using Q_i^* in Sect. 11.3 when we compositionally reason about multiple requests to access a shared resource. This concludes our motivation for the First Assumption (46).

11.2.2 Justification of the second abstract assumption

We now turn to motivating the Second Assumption (47) in Table 6. Let us consider the undesirable situation where both processes P_0' and P_1' in Peterson's algorithm somehow end up simultaneously in their critical sections. Suppose the processes are running in an interval σ. Therefore, the concrete instance (49) of the First Assumption (46) ensures that σ satisfies $\mathit{PeteSafe}_0$ and $\mathit{PeteSafe}_1$. Furthermore, the failure of mutual exclusion means that σ has a state satisfying $cs_0 \wedge cs_1$. Let σ' denote the finite prefix subinterval of σ ending with that state. It follows that σ' satisfies $\mathit{fin}(cs_0 \wedge cs_1)$. The combination of the fact that σ satisfies $\mathit{PeteSafe}_0$ and $\mathit{PeteSafe}_1$ together with the definition of $\mathit{PeteSafe}_i$ moreover ensures that σ' must satisfy the concrete implications $(\mathit{fin}\ cs_0) \supset \Diamond E_0$ and $(\mathit{fin}\ cs_1) \supset \Diamond E_1$. Here is a summary of this:

$$\sigma' \models \mathit{fin}(cs_0 \wedge cs_1) \qquad \sigma' \models (\mathit{fin}\ cs_0) \supset \Diamond E_0 \qquad \sigma' \models (\mathit{fin}\ cs_1) \supset \Diamond E_1.$$

Hence, σ' also satisfies the two formula $\Diamond E_0$ and $\Diamond E_1$. Consequently, if we can somehow prove that in fact the conjunction $(\Diamond E_0) \wedge (\Diamond E_1)$ is not satisfied by any finite interval (such as σ'), it follows from a proof by contradiction that σ' does not exist. Instead, all finite prefix subintervals of σ satisfy $\mathit{fin}\ \neg(cs_0 \wedge cs_1)$, so σ itself satisfies $\boxed{f}\ \mathit{fin}\ \neg(cs_0 \wedge cs_1)$, which is semantically equivalent to $\Box\neg(cs_0 \wedge cs_1)$. This demonstrates that mutual exclusion is achieved. Indeed, we later show in Sect. 13.3 (Lemma 13.9) the validity of following implication about $\Diamond E_0$ and $\Diamond E_1$ not being simultaneously satisfiable in a finite interval:

$$\models \quad \Diamond E_0 \ \wedge\ \Diamond E_1 \quad \supset \quad \mathit{inf}.$$

The Second Assumption (47) in Table 6 is simply a much more abstract version of this implication which can capture behaviour in many mutual exclusion algorithms. This concludes our motivation for the Second Assumption.

11.2.3 Proof of mutual exclusion from the two abstract assumptions

Lemma 11.2 shortly establishes that the First and Second Assumptions together suffice to ensure mutual exclusion for the abstract processes Q_0 and Q_1 as stated in the following implication:

$$\models \quad \bigwedge_{i\in\{0,1\}} (\mathit{init}_i \wedge Q_i) \quad \supset \quad \Box\neg(cs_0 \wedge cs_1).$$

Let us also point out that we already encountered in Lemma 9.1 the implication (29) which serves as an assumption and is moreover *exactly like* the Second Assumption (47). Now Lemma 9.1 concerns the ISEPI technique for ***P****arallel combining* (i.e., $\models (A_1 \wedge A_2) \supset A'$) of two suitable 2-to-1 formulas such as $\mathit{AbsSafe}_0$ and $\mathit{AbsSafe}_1$ which are intended for backward analysis. Recall that Lemma 9.1 states that if the implication $(\Diamond B \wedge \Diamond B') \supset \mathit{inf}$ is valid, then so is the implication below (which reproduces formula (30) in Lemma 9.1's statement):

$$\models \quad \boxed{\scriptstyle f}\big((\mathit{fin}\, w) \supset \Diamond B\big) \;\wedge\; \boxed{\scriptstyle f}\big((\mathit{fin}\, w') \supset \Diamond B'\big) \quad \supset \quad \Box\neg(w \wedge w').$$

The Second Assumption (47) $\models (\Diamond D_0 \wedge \Diamond D_1) \supset \mathit{inf}$ for abstract processes Q_0 and Q_1 is indeed just a straightforward instance of the assumption $\models (\Diamond B \wedge \Diamond B') \supset \mathit{inf}$. Therefore, a version of Lemma 9.1 can be specialised to deal with $\mathit{AbsSafe}_0$, $\mathit{AbsSafe}_1$, cs_0 and cs_1:

Corollary 11.1 *Suppose the Second Assumption* (47) *in Table 6 holds. Then it ensures that the abstract safety formulas imply mutual exclusion as expressed by the valid implication below which combines two instances of the 2-to-1 formula $\mathit{AbsSafe}_i$ in parallel:*

$$\models \quad \mathit{AbsSafe}_0 \;\wedge\; \mathit{AbsSafe}_1 \quad \supset \quad \Box\neg(cs_0 \wedge cs_1). \tag{50}$$

Corollary 11.1 helps in a simple proof of the next Lemma 11.2 concerning mutual exclusion for the abstract processes Q_0 and Q_1:

Lemma 11.2 *The First Assumption* (46) *and Second Assumption* (47) *together ensure that two abstract processes can achieve mutual exclusion as formalised in the valid implication below:*

$$\models \quad \bigwedge_{i\in\{0,1\}} (\mathit{init}_i \wedge Q_i) \quad \supset \quad \Box\neg(cs_0 \wedge cs_1). \tag{51}$$

Proof The Second Assumption (47) and Corollary 11.1 together ensure the valid implication (50). We then use simple propositional reasoning to combine this implication with the First Assumption (46) to obtain the desired valid implication (51), thus ensuring mutual exclusion for the abstract processes Q_0 and Q_1.

Below is a chain of valid implications which capture the main reasoning:

$$\textstyle\bigwedge_{i\in\{0,1\}} (\mathit{init}_i \wedge Q_i) \quad \supset \quad \mathit{AbsSafe}_0 \;\wedge\; \mathit{AbsSafe}_1 \quad \supset \quad \Box\neg(cs_0 \wedge cs_1). \qquad \Box$$

11.3 Multiple accesses by a process to a shared resource

Let us now consider a lemma showing that the two Abstract Assumptions (46) and (47) ensure validity of an implication which describes mutual exclusion for multiple accesses to the critical sections expressed with weak chop-star:

Lemma 11.3 *If Assumptions* (46) *and* (47) *in Table 6 hold, then the next formula concerning multiple requests for a shared resource is valid:*

$$\models \quad \bigwedge_{i\in\{0,1\}} (init_i \wedge Q_i^*) \quad \supset \quad \Box\neg(cs_0 \wedge cs_1). \tag{52}$$

Proof Recall the First Assumption (46):

$$\models \quad init_i \;\wedge\; Q_i \quad \supset \quad AbsSafe_i \;\wedge\; fin\, init_i.$$

This has the form $\models (w \wedge A) \supset (B \wedge fin\, w)$. Our earlier work on compositional reasoning (e.g., [48–51]) discusses semantic inference rules for various combinations of such formulas. Here is a version of one from [48] which is quite suitable for our purposes here:

$$\models (w \wedge A) \supset (B \wedge fin\, w) \quad \Rightarrow \quad \models (w \wedge A^*) \supset (B^* \wedge fin\, w).$$

For example, we can prove this for finite time by doing induction on interval length. The rule yields from the First Assumption (46) a valid generalisation to multiple exclusive accesses by one process:

$$\models \quad init_i \;\wedge\; Q_i^* \quad \supset \quad (AbsSafe_i)^* \;\wedge\; fin\, init_i. \tag{53}$$

The formula $AbsSafe_i$, which has the form $\boxed{f}((fin\, w) \supset \Diamond B)$, is “almost” $*$-to-1 by Lemma 10.11 in Sect. 10.1. This is formalised by a valid implication for the ISEPI technique of ***Iteration*** (see also the valid implication (40) and Table 5 in Sect. 10.1):

$$\models \quad \neg cs_i \;\wedge\; (AbsSafe_i)^* \quad \supset \quad AbsSafe_i. \tag{54}$$

As noted earlier, we assume that $init_i$ implies $\neg cs_i$ (see implication (45)), so we can replace $\neg cs_i$ in (54) by $init_i$:

$$\models \quad init_i \;\wedge\; (AbsSafe_i)^* \quad \supset \quad AbsSafe_i. \tag{55}$$

Propositional reasoning then permits us to combine implications (53) and (55) into the following one:

$$\models \quad init_i \;\wedge\; Q_i^* \quad \supset \quad AbsSafe_i \;\wedge\; fin\, init_i. \tag{56}$$

The Second Assumption (47) together with Corollary 11.1 about the conjunction $AbsSafe_0 \wedge AbsSafe_1$ and some further propositional reasoning then yields our goal (52).

Below is a chain of valid implications to capture the main reasoning:

$$\begin{aligned} \textstyle\bigwedge_{i\in\{0,1\}}(init_i \wedge Q_i) \quad &\supset \quad \textstyle\bigwedge_{i\in\{0,1\}}(\neg cs_i \;\wedge\; (AbsSafe_i)^*) \\ &\supset \quad AbsSafe_0 \;\wedge\; AbsSafe_1 \quad \supset \quad \Box\neg(cs_0 \wedge cs_1). \end{aligned}$$

□

$$\models \quad init_i \wedge Q_i \quad \supset \quad \big((\Box\neg cs_i); R_i; (cs_i \wedge D'_i); \Box\neg cs_i\big) \wedge \mathit{fin}\, init_i \tag{58}$$

$$\models \quad D'_i \quad \supset \quad \boxed{\text{f}}D'_i \tag{59}$$

$$\models \quad D_i; D'_i \quad \supset \quad D_i \tag{60}$$

where R_i is defined as follows:

$$R_i \quad \widehat{=} \quad \Box(\mathit{more} \supset \neg cs_i) \wedge (\mathit{inf} \vee \Diamond D_i). \tag{61}$$

Table 7 The third abstract assumption

12 Correctness of an individual abstract process

Recall that the First Assumption (46) in Table 6 in Sect. 11.2 states that a single abstract process Q_i ensures that the 2-to-1 formula $\mathit{AbsSafe}_i$ is true:

$$\models \quad init_i \wedge Q_i \quad \supset \quad \mathit{AbsSafe}_i \wedge \mathit{fin}\, init_i.$$

The rather abstract First Assumption gives no details about how Q_i achieves this. We shortly define what we call the *Third Abstract Assumption* or more briefly the *Third Assumption*. This contains some sufficient conditions concerning the overall structure of the sequential behaviour of Q_i. Later in this section's Lemma 12.1, these conditions are shown to indeed ensure the validity of the First Assumption for a single abstract process. Subsequently in Sect. 13.3 we prove that an individual process in Peterson's algorithm fulfils the Third Assumption (see Theorem 13.4). Therefore, Lemma 12.1 guarantees that it fulfils the First Assumption as well.

For the convenience of readers, we reproduce below our earlier informal outline of an abstract process previously given as (44) at the start of Sect. 11:

(a) *Noncritical* section;
(b) *Request* exclusive right to resource;
(c) *Critical* section with exclusive access; (57)
(d) *Release* exclusive right to resource;
(e) *Noncritical* section.

Table 7 shows some formulas (58)–(61) which collectively make up the **Third Abstract Assumption** (also referred to as the **Third Assumption**) about the behaviour of abstract process Q_i's steps. As we already noted, the Third Assumption is meant to precisely model the informal description in (57).

It is important to observe that we have fashioned the individual formulas in the Third Assumption in Table 7 so that they are readily suitable for use with the compositional ISEPI techniques already presented in Sects. 6–10 for backward analysis with 2-to-1 formulas. Table 8 lists several ISEPI techniques for the 2-to-1 formula $\mathit{AbsSafe}_i$ which are all instances of the ones in the previous Table 5 in Sect. 10.1. The various formulas found in the Third Assumption in Table 7 find application with most of the entries for ISEPI techniques in Table 8.

Let us now consider each of the Third Assumption's parts individually:

– **Third Assumption's implication** (58)**:**

$$\models \quad init_i \wedge Q_i \quad \supset \quad \big((\Box\neg cs_i); R_i; (cs_i \wedge D'_i); \Box\neg cs_i\big) \wedge \mathit{fin}\, init_i.$$

Introduction (first variant): Instance of formula (19) in Sect. 7:

$$\models \quad \Box\neg cs_i \quad\supset\quad \mathit{AbsSafe}_i. \tag{62}$$

Introduction (second variant): Instance of formula (20) in Sect. 7:

$$\models \quad \Box(\mathit{more} \supset \neg cs_i) \;\wedge\; (\mathit{inf} \vee \Diamond D_i) \quad\supset\quad \mathit{AbsSafe}_i. \tag{63}$$

Sequential combining: (See Theorem 6.1)

$$\models \quad \mathit{AbsSafe}_i; \mathit{AbsSafe}_i \quad\supset\quad \mathit{AbsSafe}_i. \tag{64}$$

Extending rightward: Instance of formula (25) in Sect. 8.2:

$$\models \quad \mathit{AbsSafe}_i; (cs_i \wedge D'_i) \quad\supset\quad \mathit{AbsSafe}_i, \tag{65}$$

where D'_i is a 1-to-$\boxed{f}$ formula and extends D_i rightward (i.e., $\models (D_i; D'_i) \supset D_i$).

Parallel combining: Instance of formula (30) in Sect. 9:

$$\models \quad \mathit{AbsSafe}_0 \wedge \mathit{AbsSafe}_1 \quad\supset\quad \Box\neg(cs_0 \wedge cs_1), \tag{66}$$

where $\models (\Diamond D_0 \wedge \Diamond D_1) \supset \mathit{inf}$.

Iteration (version for $+$-to-1 formula): Instance of formula (35) in Sect. 10:

$$\models \quad (\mathit{AbsSafe}_i)^{+} \quad\supset\quad \mathit{AbsSafe}_i.$$

Iteration (version for “almost” $*$-to-1 formula): Instance (54) in Sect. 13.3 of formula (40) in Sect. 10.1:

$$\models \quad \neg cs_i \;\wedge\; (\mathit{AbsSafe}_i)^{*} \quad\supset\quad \mathit{AbsSafe}_i.$$

Table 8 Summary of ISEPI techniques with the 2-to-1 formula $\mathit{AbsSafe}_i$

This primarily ensures that process Q_i achieves four sequential phases corresponding to lines (a), (b), (c) and the pair of lines (d)-(e) of the abstract process in (57). A feature of the Third Assumption's first implication (58) is that it is abstract and compositional enough to not need a detailed labelling of individual program steps in Q_i. We instead sequentially compose them using the chop operator in PITL. Implication (58) asserts that when Q_i operates with the state formula init_i initially true, then the steps are sequentially performed and also init_i is true in the last state if there is one. Here are the lines in the abstract process in (57) and the corresponding individual steps:

(a)	(b)	(c)	(d)-(e)
$\Box\neg cs_i$	R_i	$cs_i \wedge D'_i$	$\Box\neg cs_i$.

The first and fourth of the abstract steps are both the formula $\Box\neg cs_i$. This concerns a period of time when Q_i is not in its critical section and does not even try to enter it. The second formula R_i is for when Q_i has succeeded or failed to enter the critical section in line (b) in the abstract process in (57). The formula R_i's definition (61) is discussed shortly.

– **Third Assumption's implications** (59) **and** (60)**:**

$$\models \quad D'_i \quad \supset \quad \boxed{\text{f}}D'_i \qquad\qquad \models \quad D_i; D'_i \quad \supset \quad D_i.$$

These together restrict the formula D'_i to being a 1-to-$\boxed{\text{f}}$ formula (Definition 8.2 in Sect. 8.1) and also ensure that it extends D_i rightward. We require the two assumptions so that we can invoke Theorem 8.7 (found in Sect. 8.2) for the ISEPI technique of ***E****xtending rightward* a 2-to-1 formula for backward analysis using a 1-to-$\boxed{\text{f}}$ formula. Table 8 includes an instance (65) of the ISEPI technique for ***E****xtending rightward* the 2-to-1 formula $\mathit{AbsSafe}_i$. This is obtained using Theorem 8.7.

We illustrate D_i and D'_i with a simple contrived example. If D_i is the formula $p_i \wedge \Diamond p'_i \wedge \mathit{stable}\, p_i$, then D'_i could be the formula $\mathit{stable}\, p_i$, which is in fact both 1-to-$\boxed{\text{f}}$ and 2-to-1 (as discussed in Sect. 8.1 after Theorem 8.5). The chain of valid implications below shows the ISEPI technique of ***E****xtending rightward* the formula $p_i \wedge \Diamond p'_i \wedge \mathit{stable}\, p_i$ (which serves as D_i) using the formula $\mathit{stable}\, p_i$:

$$\begin{aligned}&(p_i \;\wedge\; \Diamond p'_i \;\wedge\; \mathit{stable}\, p_i); \mathit{stable}\, p_i \quad \supset \quad p_i \;\wedge\; \Diamond p'_i \;\wedge\; (\mathit{stable}\, p_i; \mathit{stable}\, p_i)\\ &\quad \supset \quad p_i \;\wedge\; \Diamond p'_i \;\wedge\; \mathit{stable}\, p_i.\end{aligned}$$

These sample D_i and D'_i are only for illustrative purposes since they are unlikely to properly ensure mutual exclusion.

– **Third Assumption's definition of R_i in** (61)**:**

$$R_i \quad \widehat{=} \quad \Box(\mathit{more} \supset \neg cs_i) \;\wedge\; (\mathit{inf} \vee \Diamond D_i).$$

This concerns the step in line (b) of (57) when the process awaits entry into its critical section. The definition captures the idea that the process waits infinitely long in vain to enter the critical section or succeeds with the formula D_i true in some suffix of the interval associated with R_i. We can immediately use this formula as an instance of the earlier valid implication (20) in Sect. 7 for the ISEPI technique of ***I****ntroduction* of $\mathit{AbsSafe}_i$. Implication (63) in Table 8 corresponds to this.

The next Lemma 12.1 formally states that the conditions in the Third Assumption suffice to imply the First Assumption:

Lemma 12.1 *For any formulas init_i, Q_i, D_i, D'_i, if all three implications* (58)–(60) *in the Third Assumption are valid, then so is the First Assumption* (46).

Proof We first prove the validity of the next formula (67) and make use of the valid implications (62)–(65) in Table 8 concerning ISEPI techniques for $\mathit{AbsSafe}_i$:

$$\models \quad (\Box\neg cs_i); R_i; (cs_i \wedge D'_i); \Box\neg cs_i \quad \supset \quad \mathit{AbsSafe}_i. \tag{67}$$

1	$\models \Box\neg cs_i \quad \supset \quad \mathit{AbsSafe}_i$	(62) [ISEPI]
2	$\models \Box(\mathit{more} \supset \neg cs_i) \;\wedge\; (\mathit{inf} \vee \Diamond D_i) \quad \supset \quad \mathit{AbsSafe}_i$	(63) [ISEPI]
3	$\models R_i \quad \supset \quad \mathit{AbsSafe}_i$	2, Def. of R
4	$\models \mathit{AbsSafe}_i; (cs_i \wedge D'_i) \quad \supset \quad \mathit{AbsSafe}_i$	(65) [ISEPI]
5	$\models R_i; (cs_i \wedge D'_i) \quad \supset \quad \mathit{AbsSafe}_i$	3, 4, PITL
6	$\models (\Box\neg cs_i); R_i; (cs_i \wedge D'_i); \Box\neg cs_i \quad \supset \quad \mathit{AbsSafe}_i; \mathit{AbsSafe}_i; \mathit{AbsSafe}_i$	1, 5, PITL
7	$\models \mathit{AbsSafe}_i; \mathit{AbsSafe}_i \quad \supset \quad \mathit{AbsSafe}_i$	(64) [ISEPI]
8	$\models \mathit{AbsSafe}_i; \mathit{AbsSafe}_i; \mathit{AbsSafe}_i \quad \supset \quad \mathit{AbsSafe}_i$	7, PITL
9	$\models (\Box\neg cs_i); R_i; (cs_i \wedge D'_i); \Box\neg cs_i \quad \supset \quad \mathit{AbsSafe}_i$	6, 8, Prop.

This can be summarised as a chain of valid implications clearly showing our application to $AbsSafe_i$ of the ISEPI techniques discussed in Sects. 6–8 for ***I**ntroducing*, ***S**equential combining* and ***E**xtending rightward* such 2-to-1 formulas for backward analysis. We underline the parts of formulas which get reduced to $AbsSafe_i$:

$$\begin{aligned}\underline{(\Box\neg cs_i)}; R_i; (cs_i \wedge D'_i); \underline{\Box\neg cs_i} \quad &\supset \quad AbsSafe_i; \underline{R_i}; (cs_i \wedge D'_i); AbsSafe_i \\ \supset \quad & AbsSafe_i; \underline{AbsSafe_i; (cs_i \wedge D'_i)}; AbsSafe_i \\ \supset \quad & \underline{AbsSafe_i; AbsSafe_i; AbsSafe_i} \quad \supset \quad AbsSafe_i.\end{aligned}$$

It then follows from implication (58) in the Third Assumption together with implication (67) that the Third Assumption indeed suffices to ensure the First Assumption (46)'s validity. □

Suppose we instead let the abstract process Q_i itself be *defined* to be the following:

$$(\Box\neg cs_i); R_i; (cs_i \wedge D'_i); \Box\neg cs_i.$$

Then the Third Assumption's first formula (58) can be simplified as shown below:

$$\models \quad init_i \wedge Q_i \quad \supset \quad fin\; init_i.$$

13 Analysis of Peterson's algorithm

Before showing mutual exclusion for Peterson's algorithm (given earlier in Fig. 1 in Sect. 11), we capture the processes' behaviour in PITL. Varying concurrency assumptions can be made. We discuss one possible way which illustrates some compositional techniques and time symmetry, and furthermore serves as an initial proof-of-concept. This follows the practice of Pnueli [66], Barringer, Kuiper and Pnueli [5] and many other researchers over the years who have used Peterson's algorithm as a sort of canonical benchmark for studying mutual exclusion. Peterson's algorithm also serves this purpose in the recent textbooks by Aceto et al. [1], Taubenfeld [76], Herlihy and Shavit [29] and Kröger and Merz [37]. As we already noted at the beginning of Sect. 11, Taubenfeld's discussion about mutual exclusion using Peterson's algorithm [76] has a special significance for our approach because it helped inspire us to see the potential of time symmetry.

13.1 Expressing Peterson's algorithm in PITL

One of the main issues with modelling Peterson's algorithm in temporal logic involves the semantics of assignment statements. In imperative programming languages, when one variable is assigned, the values of others normally do not change. On the other hand, in temporal logic, a formula which only mentions the dynamic behaviour of some variables gives absolutely no indication about what happens with other variables not occurring in the formula. For example, the PTL formula $skip \wedge \big((\bigcirc p) \equiv \neg p\big)$ can be regarded as setting the next value of the propositional variable p to the negation of its current value. This tells us nothing about the behaviour of q and other propositional variables. Such a phenomenon is an instance of the **frame problem** given prominence by McCarthy and Hayes [43] (see also

Shanahan [72]). If we want variables to remain unchanged during an interval, some explicit formula or semantic mechanism for this must be in place. The simplest solution is to add a formula such as *stable* q (defined in Table 1 in Sect. 2) for each relevant variable. Hale [24] initiated the study of framing variables in ITL. Duan has also investigated this issue [13–17].

The presentation here shows one way formulas which are 2-to-1 and 1-to-$\boxed{f}$ can be used to handle framing issues.

Our illustrative formulation in temporal logic of each process P_i in Peterson's algorithm needs to ensure that the variable flag_i is always being framed or assigned using :=. We handle framing for flag_i by modelling process P_i as having *exclusive write access* to this variable. Therefore, the definitions of statements for the process P_i which do not change flag_i (i.e., all statements except the ones of the form $\mathit{flag}_i := j$) can simply include the formula $\mathit{stable}\,\mathit{flag}_i$.

On the other hand, in Peterson's algorithm the two processes must have *shared write access* to the variable *turn*. This significantly complicates framing *turn*. A process cannot simply frame *turn* by asserting *stable turn* because this would prevent the other process from changing the variable's value. However, observe that process P_0 only uses := to assign the variable *turn* the value 1, and similarly P_1 only uses := to assign *turn* the value 0. So if for example $\mathit{turn} = 0$, then only process P_0 is interested in possibly changing it to 1. We can therefore adopt the convention that if a process P_i wants to frame *turn*, the process only needs to do so between the pairs of adjacent states with $\mathit{turn} = i$ in the first one. When $\mathit{turn} = 1 - i$, the other process has the responsibility for framing. The temporal formula $\boldsymbol{\mathit{frameturn}_i}$ defined below formalises this approach in our modelling of process P_i:

$$\mathit{frameturn}_i \quad \widehat{=} \quad \Box\big((\mathit{more} \wedge \mathit{turn} = i) \supset \bigcirc\, \mathit{turn} = i\big). \tag{68}$$

Process P_i just has to include $\mathit{frameturn}_i$ instead of *stable turn* in the statements which do not change *turn* (i.e., all of the statements except $\mathit{turn} := 1 - i$). This is an acceptable solution to framing in Peterson's algorithm. For example, no logical inconsistency occurs if say process P_0 is assigning *turn* the value 1 while at the same time process P_1 is partially framing *turn* to prevent it from changing whenever it already equals 1. Also, if both processes simultaneously frame *turn*, then the combination of activities is logically equivalent to *stable turn* as expressed by the following valid PTL formula:

$$\models \quad \mathit{stable}\,\mathit{turn} \quad \equiv \quad \mathit{frameturn}_0 \;\wedge\; \mathit{frameturn}_1. \tag{69}$$

We now define another PITL formula $\boldsymbol{\mathit{nochange}_i}$. It is used as a part of statements in Peterson's algorithm which frame both flag_i and *turn*. The formula $\mathit{nochange}_i$ describes any finite or infinite period when P_0 changes neither the variable flag_i nor the variable *turn*:

$$\mathit{nochange}_i \quad \widehat{=} \quad \mathit{stable}\,\mathit{flag}_i \;\wedge\; \mathit{frameturn}_i. \tag{70}$$

We later need in Sect. 13.3 (in Lemma 13.5) the next Lemma 13.1 concerning $\mathit{nochange}_0$:

Lemma 13.1 *The formulas* $\mathit{frameturn}_i$ *and* $\mathit{nochange}_i$ *are 2-to-1 and 1-to-*$\boxed{f}$*.*

$$\begin{array}{lcl}
noop_0 & \mathrel{\widehat{=}} & nochange_0 \;\wedge\; finite \\
flag_0 := c & \mathrel{\widehat{=}} & (stable\, flag_0{}^\frown skip) \wedge fin(flag_0 = c) \\
 & & \wedge\; frameturn_0 \\
turn := 1 & \mathrel{\widehat{=}} & (frameturn_0{}^\frown skip) \wedge fin(turn = 1) \\
 & & \wedge\; stable\, flag_0 \\
\multicolumn{3}{l}{await(flag_1 = 0 \;\vee\; turn = 0) \quad \mathrel{\widehat{=}}} \\
\multicolumn{3}{l}{\quad \big(finite \equiv \Diamond(flag_1 = 0 \;\vee\; turn = 0)\big) \;\wedge\; nochange_0.}
\end{array}$$

Table 9 Semantics of individual statements in process P_0 in Peterson's algorithm

Proof We consider each of the formulas individually:

– ***frameturn$_i$*:** Observe that $frameturn_i$ (defined in (68)) has the form $\Box T$, where T is in NL^1. Therefore, Lemma 4.6 ensures that $frameturn_i$ is 2-to-1. Furthermore, $frameturn_i$ can be re-expressed to have the form $\Box(more \supset T')$, where T' is also in NL^1:

$$\models \quad frameturn_i \quad \equiv \quad \Box\big(more \supset (turn = i \supset \bigcirc\, turn = i)\big).$$

Recall from Sect. 8.1 that all such $\Box$-formulas (e.g., $stable\, flag_i$) are 1-to-$\boxed{f}$ (see the earlier Lemma 8.3 and Theorem 8.5).

– ***nochange$_i$*:** This is defined in (70) to be the conjunction of $stable\, flag_i$ and $frameturn_i$. Each of these is both 2-to-1 and 1-to-$\boxed{f}$. In addition, the 1-to-$\boxed{f}$ formulas, like the 2-to-1 formulas, are closed under conjunction (see Lemmas 4.4 and 8.4). Hence, $nochange_i$ is 2-to-1 and 1-to-$\boxed{f}$. □

An alternative and perhaps more general and intuitive approach to framing $turn$ in each P_i can employ interleaving controlled by an additional auxiliary variable. This variable determines which process has write access to $turn$. Therefore, a process has write access and is responsible for framing exactly at such times. That process can then either choose to assign a value to $turn$ or frame it. We would like in future work to look at such an approach and formally compare it with the one used here.

13.2 Semantics of one process in Peterson's algorithm

Table 9 shows the semantics of the individual statements of the concrete process P_0 in Peterson's algorithm. Note that $flag_0 := c$ is also definable as $(flag_0 \lessdot\!\sim c) \wedge frameturn_0 \wedge finite$ using the padded temporal assignment operator $\lessdot\!\sim$ (defined in Table 1 in Sect. 2). We define $await$ to terminate iff the wait condition is eventually true. Termination might not be immediate. Process P_1 has analogous definitions.

Remark 13.2 Some readers will wonder why the definition of $flag_0 := c$ requires a $skip$ subformula. This is needed so that the variable $flag_0$ remains stable except perhaps in the very last state when it might change. If we omit the $skip$, then the variable $flag_0$ will *always* be stable and *unable to change value*. This is because the definition of $stable\, flag_0$ in Table 1 in Sect. 2 specifies that the variable's value remains unchanged between *all* adjacent pairs of states. Hence, for any propositional variable p, the formula $stable\, p$ is semantically equivalent to the conjunction

$(\Box\neg p) \vee (\Box p)$. For example, the following formula concerning a change from 0 to 1 is unsatisfiable in finite intervals:

$$\neg p \;\wedge\; \mathit{stable}\, p \;\wedge\; \mathit{fin}\, p.$$

One can however dispense with the *skip* in $\mathit{flag}_0 := c$ by replacing $\mathit{stable}\, \mathit{flag}_0 {}^\frown \mathit{skip}$ with the formula $\mathit{finite} \wedge \mathit{padded}\; \mathit{flag}_0$ containing the derived PITL construct *padded*. The formula ***padded A*** is defined to keep the formula A stable except for perhaps in the last state:

$$\mathit{padded}\;A \quad \widehat{=} \quad \Box(\mathit{more} \supset \neg A) \vee \Box(\mathit{more} \supset A).$$

However, the formulas *stable A* and *padded A* are semantically equivalent in infinite intervals.

Our analysis uses a version of P_i called P_i' (previewed in Sect. 11.2.1) with the auxiliary boolean variable cs_i to track P_i's critical section:

$$\begin{aligned} P_i' \quad \widehat{=} \quad & \big(P_i\text{'s lines 1–3} \;\wedge\; \mathit{stable}\, cs_i\big); \\ & \big(P_i\text{'s line 4} \;\wedge\; cs_i \lessdot\!\sim \mathit{true}\big); \\ & \big(P_i\text{'s line 5} \;\wedge\; cs_i \lessdot\!\sim \mathit{false}\big); \\ & \big(P_i\text{'s lines 6–7} \;\wedge\; \mathit{stable}\, cs_i\big). \end{aligned} \tag{71}$$

We ultimately prove the validity of the next implication which formalises mutual exclusion for Peterson's algorithm:

$$\models \quad \bigwedge_{i\in\{0,1\}} (\mathit{pinit}_i \wedge P_i') \quad \supset \quad \Box\neg(cs_0 \wedge cs_1). \tag{72}$$

This is a concrete instance of the earlier formula (51) for abstract mutual exclusion. The state formula pinit_i for initialisation denotes the conjunction $\mathit{flag}_i = 0 \wedge \neg cs_i$ and is later formally defined as formula (76), but it was already previewed in Sect. 11.1.

13.3 Mutual exclusion for Peterson's algorithm based on the abstract one

We now consider how the properties we showed for an abstract model of a single process can be employed to reason about Peterson's algorithm. This will save us from having to do a detailed analysis specifically for Peterson's algorithm. Such an analysis would require us to first use ISEPI techniques to prove that various parts of process P_i' with suitable pre-conditions each imply some 2-to-1 formula or a related one and then to combine them to get a 2-to-1 formula for P_i'. Instead, we only need to show something weaker about Peterson's algorithm. This then ensures that a concrete instance of the 2-to-1 formula $\mathit{AbsSafe}_i$ for backward analysis holds as well. Furthermore, various other properties of the abstract algorithm automatically apply to Peterson's algorithm (e.g., formula (52) in Lemma 11.3 for multiple accesses).

Our main goal here is the validity of concrete instances for Peterson's algorithm of the Second Assumption (47) and the Third Assumption (implications (58)–(60) in Table 7 in Sect. 12). As we previously noted, the Third Assumption is meant to precisely model the informal description in (57) of an abstract process.

Let $\boldsymbol{PeteSafe_i}$ denote a concrete instance of $AbsSafe_i$. We already discussed $PeteSafe_i$ in a preliminary manner in Sects. 11.2.1 and 11.2.2 in order to motivate our use of the abstract processes Q_i, the associated 2-to-1 formula $AbsSafe_i$ and the two associated Abstract Assumptions (46) and (47) in Table 6. The concrete instance of D_i used in $PeteSafe_i$ is the following conjunction, which we denote as $\boldsymbol{E_i}$:

$$E_i \quad \widehat{=} \quad flag_i = 1 \;\wedge\; turn = 1-i \;\wedge\; \Diamond(flag_{1-i} = 0 \;\vee\; turn = i) \;\wedge\; nochange_i. \tag{73}$$

Recall that the formula $nochange_i$ previously defined in (70) specifies that P_i does not alter either of the variables $flag_i$ and $turn$. The formula E_i is based on the values of $flag_i$ and $turn$ after process P_i's lines 2–3 together with the behaviour of lines 4–5. This was already overviewed in Sect. 11.2.1 but is now summarised again. The relevant phase of process operation concerns requesting entry into the critical section, at which time $flag_i = 1$ and $turn = 1-i$, and then either *succeeding* with $flag_{1-i} = 0$ or $turn = i$ or alternatively *forever waiting in vain.* The formula E_i contains the subformula $\Diamond(flag_{1-i} = 0 \vee turn = i)$ and so deals with the case when the request is *successful.*

Lemma 13.3 *The formula E_i is 2-to-1.*

Proof This follows from E_i being the conjunction of formulas which are themselves 2-to-1 (using Lemmas 4.3 and 13.1 and then Lemma 4.4). □

Lemma 13.3 is invoked later on in Lemma 13.6's proof.

We can in principle use $noop_i$ (defined in Table 9) instead of $nochange_i$ in E_i's definition. However, the analysis with $nochange_i$ is slightly simpler because it omits the subformula *finite* in $noop_i$.

The use of the formula E_i as a concrete instance of D_i results in the concrete instance of $AbsSafe_i$ which we denote as $\boldsymbol{PeteSafe_i}$:

$$PeteSafe_i \quad \widehat{=} \quad \boxed{\text{f}}((fin\ cs_i) \supset \Diamond E_i). \tag{74}$$

Another possibility for E_i in $PeteSafe_i$ is the weak chop of P_i's lines 2–5 in Sect. 11's Fig. 1. We can denote this portion of P_i as $\boldsymbol{P_{i,2-5}}$.

The formula $PeteSafe_i$ is 2-to-1 because it is a concrete instance of $AbsSafe_i$, and we therefore have the next instance of the valid implication (64) in Table 8 in Sect. 12:

$$\models \quad PeteSafe_i; PeteSafe_i \quad \supset \quad PeteSafe_i \tag{75}$$

In addition to formulas $PeteSafe_i$ and E_i, we also define $\boldsymbol{pinit_i}$, $\boldsymbol{ptest_i}$ and $\boldsymbol{ptest'_i}$ to each be a state formula as described below:

$$pinit_i \quad \widehat{=} \quad flag_i = 0 \;\wedge\; \neg cs_i \tag{76}$$

$$ptest_i \quad \widehat{=} \quad flag_i = 1 \;\wedge\; turn = 1-i \qquad \text{(See line 3 of } P_i \text{ in Fig. 1)} \tag{77}$$

$$ptest'_i \quad \widehat{=} \quad flag_{1-i} = 0 \;\vee\; turn = i \qquad \text{(See line 4 of } P_i \text{ in Fig. 1).} \tag{78}$$

Therefore, the following equivalence relating the formula E_i (defined in (73)) with $ptest_i$, $ptest'_i$, and $nochange_i$ is valid:

$$\models \quad E_i \quad \equiv \quad ptest_i \;\wedge\; \Diamond ptest'_i \;\wedge\; nochange_i. \tag{79}$$

Part $\boldsymbol{P'_{0,-}}$	Pre-condition $\boldsymbol{pre_{0,-}}$	Post-condition $\boldsymbol{post_{0,-}}$
$\boldsymbol{P'_{0,1-3}}$	$\mathit{flag}_0 = 0 \;\wedge\; \neg cs_0$	$\mathit{flag}_0 = 1 \;\wedge\; \mathit{turn} = 1 \;\wedge\; \neg cs_0$
$\boldsymbol{P'_{0,4}}$	$\mathit{flag}_0 = 1 \;\wedge\; \mathit{turn} = 1 \;\wedge\; \neg cs_0$	$\mathit{flag}_0 = 1 \;\wedge\; cs_0$
$\boldsymbol{P'_{0,5}}$	$\mathit{flag}_0 = 1 \;\wedge\; cs_0$	$\mathit{flag}_0 = 1 \;\wedge\; \neg cs_0$
$\boldsymbol{P'_{0,6-7}}$	$\mathit{flag}_0 = 1 \;\wedge\; \neg cs_0$	$\mathit{flag}_0 = 0 \;\wedge\; \neg cs_0.$

Table 10 Pre- and post-conditions for parts of $\boldsymbol{P'_0}$

We need concrete instances of the formulas init_i (found in the First Assumption (46) in Table 6 and Third Assumption in Table 7) and D'_i (found in the Third Assumption in Table 7) which are suitable for Peterson's algorithm. Let us take init_i to be pinit_i and D'_i to be $\mathit{nochange}_i$. Here is a summary of the various abstract formulas and corresponding concrete instances:

Abstract formula:	init_i	D_i	D'_i	$\mathit{AbsSafe}_i$
Concrete formula:	pinit_i	E_i	$\mathit{nochange}_i$	$\mathit{PeteSafe}_i.$

Alternatively, D_i can be $P_{i,2-5}$ (the weak chop of P_i's lines 2–5) or even $P_{i,2-4}$ and D'_i can be noop_i.

In addition, we use a concrete version S_i of the formula R_i defined in the Third Assumption in Table 7. The two formulas are given below to facilitate comparison:

$$\begin{aligned}\text{Abstract formula } R_i\colon\ & \Box(\mathit{more} \supset \neg cs_i) \;\wedge\; (\mathit{inf} \vee \Diamond D_i)\\ \text{Concrete formula } S_i\colon\ & \Box(\mathit{more} \supset \neg cs_i) \;\wedge\; (\mathit{inf} \vee \Diamond E_i)\end{aligned}$$

We now turn to showing that Peterson's algorithm indeed obeys the requirements imposed on the abstract algorithm to guarantee mutual exclusion. Our presentation first considers the Third Assumption in Table 7, which by Lemma 12.1 implies the First Assumption (46), and then deals with the Second Assumption (47). These suffice to show that *all* of the mutual exclusion properties we established for the abstract algorithm also apply to Peterson's concrete one.

Owing to symmetry, our analysis only needs to consider process P_0 in Peterson's algorithm. For clarity, we typeset in boldface references to the version $\boldsymbol{P'_0}$, which was defined in (71) in Sect. 13.2 and includes the behaviour of cs_0. Let us first re-express the formula $\boldsymbol{P'_0}$ as the semantically equivalent formula $\boldsymbol{P'_{0,1-3}}; \boldsymbol{P'_{0,4}}; \boldsymbol{P'_{0,5}}; \boldsymbol{P'_{0,6-7}}$ and look at the behaviour of the four primary sequential parts $\boldsymbol{P'_{0,1-3}}$, $\boldsymbol{P'_{0,4}}$, $\boldsymbol{P'_{0,5}}$ and $\boldsymbol{P'_{0,6-7}}$. Our analysis shows that each of these combined with a suitable pre-condition implies a corresponding part in the formula given below:

$$(\Box\neg cs_i); S_i; (cs_i \wedge \mathit{nochange}_i); \Box\neg cs_i.$$

This is a concrete instance of a subformula of the Third Assumption's first formula (58). Every primary sequential part with its pre-condition furthermore also implies the associated post-condition in the final state when there is one.

State formulas for the pre- and post-conditions for the parts of $\boldsymbol{P'_0}$ are fairly straightforward. Table 10 shows one possible approach. For example, $\mathit{pre}_{0,4}$ refers to the pre-condition $\mathit{flag}_0 = 1 \wedge \mathit{turn} = 1 \wedge \neg cs_0$ for part $\boldsymbol{P'_{0,4}}$. Observe that normally the post-condition for each part is actually the pre-condition of the next one. In the case of $\boldsymbol{P'_{0,6-7}}$, which is the last part, the post-condition can be taken to be the pre-condition $\mathit{pre}_{0,1-3}$ for $\boldsymbol{P'_{0,1-3}}$. We therefore only need to refer to the formulas for pre-conditions and do not actually need separate names for the post-conditions.

Note that $\mathit{pre}_{0,1-3}$ is identical to pinit_0 (i.e., both denote $\mathit{flag}_0 = 0 \;\wedge\; \neg cs_0$; see (76)).

Theorem 13.4 *The concrete instances of all of the Third Assumption's three implications* (58)–(60) *for our version of Peterson's algorithm are valid.*

The proof is deferred until after we first state and prove Lemmas 13.5–13.7 for the concrete instances of the Third Assumption's three implications (58)–(60) for Peterson's algorithm. Since the proof of Lemma 13.7 for the first one (58) is the most complicated, we save it for last.

Lemma 13.5 (Validity of instance of Third Assumption's implication (59)**)** *The next concrete instance of the abstract algorithm's implication $D'_i \supset \boxed{f} D'_i$ is valid:*

$$\models \quad \mathit{nochange}_i \;\supset\; \boxed{f}\,\mathit{nochange}_i.$$

Proof This follows immediately from the earlier Lemma 13.1 in Sect. 13.1, thus ensuring that $\mathit{nochange}_i$ is 1-to-$\boxed{f}$. □

Lemma 13.6 (Validity of instance of Third Assumption's implication (60)**)** *The next concrete instance of the abstract algorithm's implication $(D_i; D'_i) \supset D_i$ is valid:*

$$\models \quad E_i; \mathit{nochange}_i \quad\supset\quad E_i. \tag{80}$$

Proof Recall from Lemma 13.3 that E_i is 2-to-1. The proof of the validity of (80) involves a routine use of the ISEPI technique of ***E****xtending rightward* such a 2-to-1 formula. We employ the equivalence (79) to express E_i as the conjunction $\mathit{ptest}_i \wedge \Diamond \mathit{ptest}'_i \wedge \mathit{nochange}_i$. Here is chain of valid implications which make use of the fact that $\mathit{nochange}_i$ is 2-to-1 as well (Lemma 13.1):

$$\begin{array}{ll} E_i; \mathit{nochange}_i & \\ \quad\supset & (\mathit{ptest}_i \wedge \Diamond \mathit{ptest}'_i \wedge \mathit{nochange}_i); \mathit{nochange}_i \\ \quad\supset & \mathit{ptest}_i \;\wedge\; \Diamond \mathit{ptest}'_i \;\wedge\; (\mathit{nochange}_i; \mathit{nochange}_i) \\ \quad\supset & \mathit{ptest}_i \;\wedge\; \Diamond \mathit{ptest}'_i \;\wedge\; \mathit{nochange}_i \\ \quad\supset & E_i. \end{array}$$

Hence, E_i can indeed be ***E****xtended rightward* with $\mathit{nochange}_i$, and so implication (80) is in fact valid. □

Lemma 13.7 (Validity of instance of Third Assumption's implication (58)**)** *For each process $\boldsymbol{P'_i}$, the following concrete instance of the Third Assumption's implication* (58) *is valid:*

$$\models \mathit{pinit}_i \;\wedge\; \boldsymbol{P'_i} \quad\supset\quad \big((\Box\neg cs_i); S_i; (cs_i \wedge \mathit{nochange}_i); \Box\neg cs_i\big) \;\wedge\; \mathit{fin}\,\mathit{pinit}_i. \tag{81}$$

Proof We will only deal with $\boldsymbol{P'_0}$, but the proof easily generalises to $\boldsymbol{P'_1}$. State formulas used for the lines' pre- and post-conditions are found in the previously presented Table 10. Each of the four main parts of P'_0 in (71), that is $\boldsymbol{P'_{0,1-3}}$, $\boldsymbol{P'_{0,4}}$, $\boldsymbol{P'_{0,5}}$ and $\boldsymbol{P'_{0,6-7}}$, contributes one of the chop operands in implication (81)'s subformula $(\Box\neg cs_0); S_0; \mathit{nochange}_i; \Box\neg cs_0$. Here are the associated implications, which are shortly proven to be valid:

$$\models \quad \mathit{pre}_{0,1-3} \ \wedge\ \boldsymbol{P'_{0,1-3}} \quad\supset\quad \Box\neg cs_0 \ \wedge\ \mathit{fin}\, \mathit{pre}_{0,4} \tag{82}$$

$$\models \quad \mathit{pre}_{0,4} \ \wedge\ \boldsymbol{P'_{0,4}} \quad\supset\quad S_0 \ \wedge\ \mathit{fin}\, \mathit{pre}_{0,5} \tag{83}$$

$$\models \quad \mathit{pre}_{0,5} \ \wedge\ \boldsymbol{P'_{0,5}} \quad\supset\quad cs_0 \wedge \mathit{nochange}_i \ \wedge\ \mathit{fin}\, \mathit{pre}_{0,6-7} \tag{84}$$

$$\models \quad \mathit{pre}_{0,6-7} \ \wedge\ \boldsymbol{P'_{0,6-7}} \quad\supset\quad \Box\neg cs_0 \ \wedge\ \mathit{fin}\, \mathit{pre}_{0,1-3}. \tag{85}$$

We structure the rest of Lemma 13.7's proof as four steps. Let us first look at a summary of them:

- **Step 1, case for $\boldsymbol{P'_{0,1-3}}$ and $\boldsymbol{P'_{0,6-7}}$:** We show the validity of the associated implications (82) and (85).
- **Step 2, case for $\boldsymbol{P'_{0,4}}$:** We show the validity of the associated implication (83).
- **Step 3, case for $\boldsymbol{P'_{0,5}}$:** We show the validity of the associated implication (84).
- **Step 4, case for $\boldsymbol{P'_0}$:** We show the validity of the associated implication (81).

Step 1, case for $\boldsymbol{P'_{0,1-3}}$ and $\boldsymbol{P'_{0,6-7}}$ to show the validity of implications (82) **and** (85)**:** Validity readily follows from the associated pre-conditions which set cs_0 to equal *false* together with the temporal formulas in the following parts of our definition (71) of $\boldsymbol{P'_0}$ corresponding to $\boldsymbol{P'_{0,1-3}}$ and $\boldsymbol{P'_{0,6-7}}$:

$$\boldsymbol{P'_{0,1-3}}\colon \big(P_i\text{'s lines 1–3} \ \wedge\ \mathit{stable}\ cs_i\big) \qquad \boldsymbol{P'_{0,6-7}}\colon \big(P_i\text{'s lines 6–7} \ \wedge\ \mathit{stable}\ cs_i\big).$$

Below are versions of the two implications (82) and (85) with the various formulas replaced by their definitions for easier checking:

$$\begin{aligned}\models\quad & \mathit{flag}_0 = 0 \ \wedge\ \neg cs_0 \ \wedge\ (\mathit{noop}_0; \mathit{flag}_0 := 1; \mathit{turn} := 1) \ \wedge\ \mathit{stable}\ cs_i \\ & \quad\supset\quad \Box\neg cs_0 \ \wedge\ \mathit{fin}(\mathit{flag}_0 = 1 \ \wedge\ \mathit{turn} = 1 \ \wedge\ \neg cs_0) \\ \models\quad & \mathit{flag}_0 = 1 \ \wedge\ \neg cs_0 \ \wedge\ (\mathit{flag}_0 := 0; \mathit{noop}_0) \ \wedge\ \mathit{stable}\ cs_i \\ & \quad\supset\quad \Box\neg cs_0 \ \wedge\ \mathit{fin}(\mathit{flag}_0 = 0 \ \wedge\ \neg cs_0).\end{aligned}$$

Step 2, case for $\boldsymbol{P'_{0,4}}$ to show the validity of implication (83)**:** Recall that the Third Assumption's definition of R_i in Table 7 is the conjunction $\Box(\mathit{more} \supset \neg cs_i) \wedge (\mathit{inf} \vee \Diamond D_i)$, so S_i is the concrete instance $\Box(\mathit{more} \supset \neg cs_i) \wedge (\mathit{inf} \vee \Diamond E_i)$. Here is an expanded version of implication (83):

$$\begin{aligned}\models\ & \mathit{flag}_0 = 1 \ \wedge\ \mathit{turn} = 1 \ \wedge\ \neg cs_0 \ \wedge\ \mathit{await}(\mathit{flag}_1 = 0 \ \vee\ \mathit{turn} = 0) \ \wedge\ cs_0 \lessdot\!\!\sim \mathit{true} \\ & \supset\quad \Box(\mathit{more} \supset \neg cs_0) \ \wedge\ (\mathit{inf} \vee \Diamond E_0) \ \wedge\ \mathit{fin}(\mathit{flag}_0 = 1 \ \wedge\ cs_0).\end{aligned} \tag{86}$$

From $\neg cs_0$ and $cs_0 \lessdot\!\!\sim \mathit{true}$ readily follows $\Box(\mathit{more} \supset \neg cs_0)$. The main remaining portion of the proof of implication (86)'s validity involves showing that (86)'s antecedent implies the consequent's subformula $\mathit{inf} \vee \Diamond E_0$. Recall from (73) that E_i has the following definition:

$$E_i \quad\widehat{=}\quad \mathit{flag}_i = 1 \ \wedge\ \mathit{turn} = 1 - i \ \wedge\ \Diamond(\mathit{flag}_{1-i} = 0 \ \vee\ \mathit{turn} = i) \ \wedge\ \mathit{nochange}_i.$$

Below is a proof first showing that the pre-condition $pre_{0,4}$ and $\boldsymbol{P'_{0,4}}$ together imply $inf \vee E_0$, from which readily follows that they imply $inf \vee \Diamond E_0$. For conciseness in the proof, we use $ptest_0$ and $ptest'_0$ to denote the state formulas $flag_0 = 1 \wedge turn = 1$ and $flag_1 = 0 \vee turn = 0$, as previously defined in (77) and (78), respectively.

1	$\models await(ptest'_0) \equiv$ $(finite \equiv \Diamond ptest'_0) \wedge nochange_0$	Def. of *await*
2	$\models (finite \equiv \Diamond ptest'_0) \supset inf \vee \Diamond ptest'_0$	PTL
3	$\models ptest_0 \wedge await(ptest'_0)$ $\supset ptest_0 \wedge (inf \vee \Diamond ptest'_0) \wedge nochange_0$	1, 2, Prop.
4	$\models ptest_0 \wedge (inf \vee \Diamond ptest'_0) \wedge nochange_0$ $\supset inf \vee (ptest_0 \wedge \Diamond ptest'_0 \wedge nochange_0)$	Prop.
5	$\models ptest_0 \wedge await(ptest'_0)$ $\supset inf \vee (ptest_0 \wedge \Diamond ptest'_0 \wedge nochange_0)$	3, 4, Prop.
6	$\models ptest_0 \wedge await(ptest'_0) \supset inf \vee E_0$	5, Def. of E_0
7	$\models inf \vee E_0 \supset inf \vee \Diamond E_0$	PTL
8	$\models ptest_0 \wedge await(ptest'_0) \supset inf \vee \Diamond E_0$	6, 7, Prop.

Step 3, case for $\boldsymbol{P'_{0,5}}$ to show the validity of implication (84)**:** This is fairly straightforward from the definitions of $noop_0$ and $cs_0 \Leftarrow\!\sim false$. Here is a version of implication (84) with $pre_{0,5}$, $\boldsymbol{P'_{0,5}}$ and $pre_{0,6-7}$ replaced by their definitions:

$$\models \begin{array}{l} flag_0 = 1 \;\wedge\; cs_0 \;\wedge\; noop_0 \;\wedge\; cs_0 \Leftarrow\!\sim false \\ \quad\supset\quad cs_0 \;\wedge\; nochange_0 \;\wedge\; fin(flag_0 = 1 \;\wedge\; \neg cs_0). \end{array}$$

Step 4, case for $\boldsymbol{P'_0}$ to show the validity of implication (81)**:** The formulas $pinit_0$ and $pre_{0,1-3}$ are identical (i.e., both denote $flag_0 = 0 \wedge \neg cs_0$; see (76)). Consequently, the initial process state ensures that $pre_{0,1-3}$ is true. The four valid implications (82)–(85) for $\boldsymbol{P'_{0,1-3}}$, $\boldsymbol{P'_{0,4}}$, $\boldsymbol{P'_{0,5}}$ and $\boldsymbol{P'_{0,6-7}}$ can then be sequentially combined to obtain the validity of the implication (81) for $\boldsymbol{P'_0}$. □

Recall that Theorem 13.4 states that concrete instances of the Third Assumption's three implications (58)–(60) (shown in Table 7 in Sect. 12) for our version of Peterson's algorithm are valid. The proof of Theorem 13.4 now readily follows:

Proof (Theorem 13.4) The previous Lemmas 13.5–13.7 together establish that the concrete instances of all three implications (58)–(60) are indeed valid. □

We now use Theorem 13.4 to show that Peterson's algorithm has suitable instances of the First Assumption:

Lemma 13.8 *The next concrete instance of the First Assumption* (46) *for each process $\boldsymbol{P'_i}$ in our version of Peterson's algorithm is valid:*

$$\models \quad pinit_i \;\wedge\; \boldsymbol{P'_i} \quad\supset\quad PeteSafe_i \;\wedge\; fin\, pinit_i. \tag{87}$$

Proof Lemma 12.1 yields from the Third Assumption the First Assumption (46). In addition, Theorem 13.4 demonstrates that each process $\boldsymbol{P'_i}$ in Peterson's algorithm fulfils an associated concrete instance of the Third Assumption. The combination of these then guarantees that each $\boldsymbol{P'_i}$ also fulfils the associated concrete instance (87) of the First Assumption. □

We also need the next Lemma 13.9 concerning the Second Assumption (47):

Lemma 13.9 *The following concrete instance of the Second Assumption* (47) *for Peterson's algorithm is valid:*

$$\models \quad \Diamond E_0 \;\wedge\; \Diamond E_1 \quad\supset\quad \mathit{inf}. \tag{88}$$

Recall that the Second Assumption (47) was needed in Sect. 11.2.3 to show mutual exclusion exclusion for the abstract algorithm. Consequently, implication (88), as an instance of the Second Assumption, encapsulates in a concise way a key aspect of Peterson's algorithm, and it focuses on a central mechanism used to ensure mutual exclusion. Therefore, the proof of the validity of implication (88) has a special significance in the understanding of how Peterson's algorithm works.

Proof (Lemma 13.9) Our proof of the validity of implication (88) actually shows the stronger result that $\Diamond E_0 \wedge \Diamond E_1$ is unsatisfiable. Simple temporal reasoning allows us to establish this by a case analysis which demonstrates that each of the following two formulas is unsatisfiable:

$$E_0 \;\wedge\; \Diamond E_1 \qquad E_1 \;\wedge\; \Diamond E_0. \tag{89}$$

This is because if $\Diamond E_0 \wedge \Diamond E_1$ were to be satisfiable, then there would be some suffix subinterval satisfying one of the two formulas $E_0 \wedge \Diamond E_1$ or $E_1 \wedge \Diamond E_0$ in (89). Owing to the symmetry involved, we only need to consider the first of these here. Let us use the valid equivalence (79) to re-express $E_0 \wedge \Diamond E_1$ in terms of ptest_i, ptest'_i, and $\mathit{nochange}_i$:

$$\mathit{ptest}_0 \;\wedge\; \Diamond \mathit{ptest}'_0 \;\wedge\; \mathit{nochange}_0 \;\wedge\; \Diamond\big(\mathit{ptest}_1 \wedge \Diamond \mathit{ptest}'_1 \wedge \mathit{nochange}_1\big). \tag{90}$$

Our analysis can ignore the conjunct $\Diamond \mathit{ptest}'_0$. When we replace the remaining state formulas ptest_0, ptest_1 and ptest'_i by their respective definitions given in (77) and (78), the slightly shortened version of formula (90) without $\Diamond \mathit{ptest}'_0$ becomes the following:

$$\begin{aligned} &\mathit{flag}_0 = 1 \;\wedge\; \mathit{turn} = 1 \;\wedge\; \mathit{nochange}_0 \\ &\quad \wedge\; \Diamond\big(\mathit{flag}_1 = 1 \;\wedge\; \mathit{turn} = 0 \wedge \Diamond(\mathit{flag}_0 = 0 \;\vee\; \mathit{turn} = 1) \wedge \mathit{nochange}_1\big). \end{aligned} \tag{91}$$

The variable flag_0 always equals 1 because of the effect of $\mathit{stable}\,\mathit{flag}_0$ in the definition of $\mathit{nochange}_0$ (see (70)). Therefore, the subformula $\mathit{flag}_0 = 0 \vee \mathit{turn} = 1$ in (91) can be reduced to $\mathit{turn} = 1$, which we underline below:

$$\begin{aligned} &\mathit{flag}_0 = 1 \;\wedge\; \mathit{turn} = 1 \;\wedge\; \mathit{nochange}_0 \\ &\quad \wedge\; \Diamond\big(\mathit{flag}_1 = 1 \;\wedge\; \mathit{turn} = 0 \wedge \Diamond(\underline{\mathit{turn} = 1}) \wedge \mathit{nochange}_1\big). \end{aligned} \tag{92}$$

Observe that the formulas $\mathit{nochange}_0$ and $\mathit{nochange}_1$, as defined in (70), are both conjunctions of $\Box$-formulas. It follows that if $\mathit{nochange}_0$ and $\mathit{nochange}_1$ are true for an interval, then they are also true for all suffix subintervals (i.e., they are **1-to-$\Box$ formulas**):

$$\models \quad \mathit{nochange}_i \quad\supset\quad \Box \mathit{nochange}_i.$$

Now consider the suffix subinterval starting with the state where the subformula $\mathit{flag}_1 = 1 \wedge \mathit{turn} = 0$ in (92) is true. In that subinterval, the formulas $\mathit{nochange}_0$ and $\mathit{nochange}_1$ are *both* true, so the two formulas $\mathit{frameturn}_0$ and $\mathit{frameturn}_1$ in their definitions are as well. Therefore, the variable *turn* must remain stable from then on (as we previously noted with valid equivalence (69) in Sect. 13.1). This behaviour concerning $nochange_0$, $nochange_1$ and *turn* is expressed by the following valid PTL formula:

$$\models \quad nochange_0 \ \wedge\ nochange_1 \quad \supset \quad stable\ turn.$$

However, the eventual stability of *turn* in formula (92) is contradicted by the conjunction $turn = 0 \wedge \Diamond(turn = 1)$ found within the outer $\Diamond$-formula. Consequently, the formula (92) itself is in fact unsatisfiable. It then follows from this that formula (91) is likewise unsatisfiable and so is the previous formula $E_0 \wedge \Diamond E_1$ in (89). Symmetry ensures that the formula $E_1 \ \wedge\ \Diamond E_0$ is unsatisfiable as well. This all demonstrates that the conjunction $\Diamond E_0 \wedge \Diamond E_1$ is unsatisfiable and so implies anything, including the formula *inf*. Consequently, implication (88) is indeed valid.

$\Box$

At this stage, we have obtained for Peterson's algorithm concrete instances of the Third Assumption (Theorem 13.4) , then the First Assumption (Lemma 13.8), and finally the Second Assumption (Lemma 13.9). Consequently, from the concrete instances (87) and (88) of the First Assumption (46) and Second Assumption (47), respectively, *all* of the mutual exclusion properties discussed for abstract processes in Sect. 11 can be carried over to Peterson's algorithm.

14 Use of time symmetry to reduce some PITL formulas to PTL

We now briefly discuss at an exploratory level how our new techniques of time symmetry and reflections introduced in Sect. 5 can provide a theoretical basis for transforming some PITL formulas to the computationally more tractable formalism PTL. The main contribution of this section is to show how to combine some ideas from our earlier work in [55] with the new concept of reflection and reasoning about prefix subintervals. In particular, we will first reduce a PITL safety property concerning backward analysis to a PTL formula involving suffix subintervals together with the temporal operators $\Box$ and *until* . This kind of reduction can then also be done on a formula about a system and an associated safety property. One potential benefit is that we can extend the application of some existing decision procedures and tool-support for PTL to handle suitable PITL formulas as well. Some computational aspects of PTL are surveyed by Kröger and Merz [37] and Fisher [19], who also provide further references to the significant literature on the subject. In [55] we discuss an implemented decision procedure for PTL with both finite and infinite time which has connections with the theory of PITL. We only mention this because some of the techniques presented in [55] are later on adapted when we reduce PITL formulas to PTL.

A key observation here is that a reflection of a formula with $\boxed{\mathrm{f}}$-subformulas contains $\Box$-subformulas. The later can be easier to reduce to PTL because $\Box$ is itself a PTL construct, whereas $\boxed{\mathrm{f}}$ is not. Recall that $\mathrm{PTL}^{\mathrm{U}}$ is the version of PTL with strong until defined earlier in Sect. 3.3. Let us use the formula $PeteSafe_0$ defined in (74) to illustrate obtaining from a $\boxed{\mathrm{f}}$-formula a reflection in $\mathrm{PTL}^{\mathrm{U}}$. In order to derive a $\mathrm{PTL}^{\mathrm{U}}$ formula which reflects $PeteSafe_0$, our reduction to $\mathrm{PTL}^{\mathrm{U}}$

first obtains the next interval-oriented way to re-express $\mathit{PeteSafe}_0$:

$$\boxed{\scriptstyle f}\Big((\mathit{fin}\ cs_0) \supset \Diamond\big((\mathit{empty} \wedge \mathit{ptest}_0); \mathit{nochange}_0; \\ (\mathit{empty} \wedge \mathit{ptest}'_0); \mathit{nochange}_0\big)\Big). \tag{93}$$

Here the PTL formula E_0 (defined in (73)) has been replaced by a PITL formula with weak chops which is semantically equivalent to E_0 in finite intervals. We now reflect (93):

$$\Box\Big(cs_0 \ \supset\ \mathop{\Diamond\kern-0.62em{\scriptscriptstyle f}\kern0.3em}\big(\mathit{nochange}^r_0; (\mathit{empty} \wedge \mathit{ptest}'_0); \\ \mathit{nochange}^r_0; (\mathit{empty} \wedge \mathit{ptest}_0)\big)\Big) \tag{94}$$

We then re-express the $\mathop{\Diamond\kern-0.62em{\scriptscriptstyle f}\kern0.3em}$-subformula in $\mathrm{PTL^U}$:

$$Y\ \mathit{until}\ \big(\mathit{ptest}'_0 \wedge (Y\ \mathit{until}\ \mathit{ptest}_0)\big), \tag{95}$$

where Y acts like $\mathit{nochange}^r_0$ on pairs of adjacent states:

$$(\bigcirc\mathit{flag}_0) = \mathit{flag}_0 \ \wedge\ \big((\bigcirc\, \mathit{turn} = 0) \supset \mathit{turn} = 0\big).$$

Finally, we can take the $\mathrm{PTL^U}$ reflection of $\mathit{PeteSafe}_0$ to be (94) with the $\mathop{\Diamond\kern-0.62em{\scriptscriptstyle f}\kern0.3em}$-subformula replaced by (95). Note the reduction to $\mathrm{PTL^U}$ of the reflection of a $\boxed{\scriptstyle f}$-formula containing chop-star might require auxiliary variables. This is because PITL with chop-star (which can express regular and omega-regular languages [77]) is much more expressive than $\mathrm{PTL^U}$. However, this is not always an issue as our example demonstrates. See Kröger and Merz [37] for a discussion of the operator *until* and the expressiveness of temporal logics containing it. It would appear that reductions from PITL to $\mathrm{PTL^U}$ could be automated for a range of syntactic classes of formulas.

To further illustrate the potential of reflection, let us now consider how to check the validity of a formula $(w \wedge \mathit{Sys}) \supset (\boxed{\scriptstyle f}A \wedge \mathit{fin}\, w')$, for some system Sys expressed in PITL. For finite-time analysis, this has the reflection $((\mathit{fin}\, w) \wedge \mathit{Sys}^r) \supset (\Box A^r \wedge w')$. We can reduce Sys^r to some PTL formula X with auxiliary variables and test finite-time satisfiability of $(\mathit{fin}\, w) \wedge X \wedge \neg(X' \wedge w')$, where X' is a reflection of $\boxed{\scriptstyle f}A$ expressed in $\mathrm{PTL^U}$ as described above for the example $\mathit{PeteSafe}_0$.

For infinite time, we can first reduce Sys to a PTL formula with auxiliary variables or an omega automaton [37,77]. As we show in [55], these can be represented in PTL by a low-level **transition configuration** of the form below:

$$\Box T \ \wedge\ \mathit{init} \ \wedge\ \Box\, \Diamond^+ L, \tag{96}$$

where T is an NL^1 formula, *init* is a state formula, $\Diamond^+ L$ abbreviates $\bigcirc \Diamond L$ (**strict** $\Diamond$), and L is a finite conjunction of implications each of the form $w \supset \Diamond w'$. As shown in [55], the transition configuration has ultimately periodic models and is equivalent to the next formula in infinite time:

$$(X'' \wedge \mathit{init})^\frown\big(X'' \wedge L \wedge (\mathbf{V} \leftarrow \mathbf{V})\big)^\star, \tag{97}$$

where X'' denotes $\Box(\mathit{more} \supset T)$ and $\mathbf{V} \leftarrow \mathbf{V}$ is the conjunction of temporal assignments $v \leftarrow v$ for each variable v in the transition configuration. Note that in [55], the chop-omega operator (A^ω) is used instead of strong chop-star $(A^\star)$. However, the two operators have identical semantics in infinite intervals. Testing for infinite-time validity of $(w \wedge \mathit{Sys}) \supset (\boxed{f}A \wedge \mathit{fin}\, w')$ is reducible to checking infinite-time unsatisfiability of $\mathit{Sys} \wedge w \wedge \neg\boxed{f}A$. Here $\mathit{fin}\, w'$ is trivially true for infinite time and ignored. We then replace Sys by (97) to obtain the formula below:

$$(X'' \wedge \mathit{init})^\frown\big(X'' \wedge L \wedge (\mathbf{V} \leftarrow \mathbf{V})\big)^{\star} \wedge w \wedge \neg\boxed{f}A.$$

This is equivalent to a variant with w in the chop's left side:

$$(X'' \wedge \mathit{init} \wedge w)^\frown\big(X'' \wedge L \wedge (\mathbf{V}\leftarrow\mathbf{V})\big)^{\star} \wedge \neg\boxed{f}A. \tag{98}$$

The next semantic inference rule (related to (36)) reduces testing unsatisfiability for (98) to *finite-time* unsatisfiability:

$$\models \mathit{finite} \wedge (B_1{}^\frown B_2^{\star}) \supset \boxed{f}B_3 \quad \Rightarrow \quad \models (B_1{}^\frown B_2^{\star}) \supset \boxed{f}B_3,$$

where the B_is can be any formulas. More precisely, it follows from this that if the conjunction $(B_1{}^\frown B_2^{\star}) \wedge \neg\boxed{f}B_3$ is unsatisfiable, then it is unsatisfiable in finite time. Observe that (98) has this form. In order to do the testing for finite time, we can first reflect (98) and reduce it to a PTL formula with more auxiliary variables. For example, if $\boxed{f}A$ is the formula $\mathit{PeteSafe}_0$ we reflected above and reduced to $\mathrm{PTL}^{\mathrm{U}}$, then this $\mathrm{PTL}^{\mathrm{U}}$ reflection can be used.

The transition configuration (96) is only meant for analysing infinite-time behaviour. However, a simplified transition configuration of the form shown below can analogously be used for checking finite-time validity of $(w \wedge \mathit{Sys}) \supset (\boxed{f}A \wedge \mathit{fin}\, w')$:

$$\Box T \;\wedge\; \mathit{init} \;\wedge\; \mathit{finite}.$$

We would like to see these rather experimental ideas implemented and also to have this approach compared with others, such as one based on a reduction of the implication $(w \wedge \mathit{Sys}) \supset (\boxed{f}A \wedge \mathit{fin}\, w')$ to a suitable formula with A instead of $\boxed{f}A$. Various formulas in our analysis of Peterson's algorithm could be used as an initial test.

15 Discussion

We now touch upon a number of topics with relevance to our framework based on ITL, 2-to-1 formulas and time symmetry.

15.1 Summary of formulas closed under conjunction and box

For the convenience of readers, Table 11 lists the classes of formulas closed under conjunction and $\Box$ which we have looked at. It also mentions where they are described and some of their uses with suitable formulas. However, note that the "almost" $*$-to-1 formulas are not a proper class.

We plan in future work to discuss some other classes of formulas which are closed under conjunction and $\Box$ and have potential applications. One example is the 1-to-$\boxed{i}$ formulas, that is, any formula A for which the implication $\models A \supset \boxed{i}A$ is valid. Recall that the operator $\boxed{i}$ (defined in Table 1 in Sect. 2) examines all prefix subintervals. In contrast, the operator $\boxed{f}$ only examines prefix subintervals having

Class of formulas	Where defined	Some uses
2-to-1 Formulas	Def. 4.1	ISEPI ***S**equential combining*
$*$-to-1 Formulas	Def. 10.1	ISEPI ***I**teration*
$+$-to-1 Formulas	Def. 10.2	ISEPI ***I**teration*
(“Almost” $*$-to-1 Formulas	Lemma 10.11	ISEPI ***I**teration*)
1-to-$\boxed{f}$ Formulas	Def. 8.2	ISEPI ***E**xtend rightward*
1-to-$\Box$ Formulas	Remark 8.6	Import formula into $\Diamond$

Table 11 Various classes of formulas closed under conjunction and box

finite length. We are studying whether the operator $\boxed{i}$ and its associated class of 1-to-$\boxed{i}$ formulas can be used instead of $\boxed{f}$ and 1-to-$\boxed{f}$ formulas in practice. For example, $\boxed{f}((\mathit{fin}\, w) \supset \Diamond B)$ is semantically equivalent to the $\boxed{i}$-formula $\boxed{i}((\mathit{sfin}\, w) \supset \Diamond B)$. Here $\boldsymbol{\mathit{sfin}\, w}$ is a strong version of *fin* derivable as $\Diamond(\mathit{empty} \wedge w)$ and also expressible as $\mathit{finite} \wedge \mathit{fin}\, w$. Our general experience is that weak interval operators can sometimes be more convenient in applications involving compositionally. More evidence one way or the other still needs to be collected.

We presented in previous sections various results which relate some of the classes. For example, Theorem 10.7 gives a sufficient condition for a 2-to-1 formula to also be $+$-to-1. Similarly, Lemma 10.11 concerns 2-to-1 formulas which are “almost” $*$-to-1. It seems worthwhile to further investigate interrelationships between various classes. We report some new results in [58].

15.2 Exogenous and endogenous frameworks

Our compositional way of reasoning about concurrency in ITL using 2-to-1 formulas contrasts with the better known and much more widely used one based on point-based temporal logic that Pnueli [65] and others have quite successfully advocated. In particular, the point-based approach does not represent or reason about a program directly in the logic but requires it to be first translated into a state-transition system with many labels (as was also done earlier by Floyd [20]). Temporal logic is used to reason about these. Pnueli already in his first publication about temporal logic over thirty five years ago describes this as being *endogenous* [65]:

> Another point that is worth mentioning is that the [Endogenous] approach taken here can be classified together with Floyd’s [20], ... By that [the term Endogenous] we mean that we immerse ourselves in a single program which we regard as the universe, and concentrate on possible developments within that universe. Characteristic of this approach is the first phase which translates the programming features into general rules of behavior which we later logically analyze.

An ITL-based analysis, on the other hand, is much closer to what Pnueli [65] refers to as being *exogenous* when he compares the two categories:

> These [proponents of Exogenous systems such as Hoare [30]] suggest a uniform formalism which deals in formulas whose constituents are both logical assertions and program segments, and can express very rich relations

between programs and assertions. We will be the first to admit the many advantages of Exogenous systems over Endogenous systems. These include among others:

a. The uniform formalism is more elegant and universal, richer in expressibility, no need for the two-phase process of Endogenous systems.
b. Endogenous systems live within a single program. There is no way to compare two programs such as proving equivalence or inclusion.
c. Endogenous systems assume the program to be rigidly given, Exogenous systems provide tools and guidance for *constructing* a correct system rather than just analyse an existent one.

Against these advantages Endogenous system can offer the following single line of defense: When the going is tough, and we are interested in proving a single intricate and difficult program, we do not care about generality, uniformity or equivalence. It is then advantageous to work with a fixed context rather than carry a varying context with each statement. Under these conditions, Endogenous systems attempt to equip the prover with the strongest possible tools to formalize his intuitive thinking and ease his way to a rigorous proof.

We do not believe that this is the place for a detailed, meaningful assessment of the merits of the (endogenous) point-based and (exogenous) interval-based temporal frameworks, particularly since ours is certainly much more experimental and less applied.

It seems appropriate to quote below the related discussion by Harel et al. in their comparison of **Dynamic Logic (DL)** [27,28] with point-based temporal logic since the succinctly expressed points concerning compositionality equally apply here:

> There are two main approaches to modal logics of programs: the *exogenous* approach, exemplified by Dynamic Logic and its precursor Hoare Logic [30], and the *endogenous* approach, exemplified by Temporal Logic and its precursor, the invariant assertions method of Floyd [20]. A logic is *exogenous* if its programs are explicit in the language. Syntactically, a Dynamic Logic program is a well-formed expression built inductively from primitive programs using a small set of program operators. Semantically, a program is interpreted as its input/output relation. The relation denoted by a compound program is determined by the relations denoted by its parts. This aspect of *compositionality* allows analysis by structural induction. The importance of compositionality is discussed by van Emde Boas [80]. In Temporal Logic, the program is fixed and is considered part of the structure over which the logic is interpreted. The current location in the program during execution is stored in a special variable for that purpose, called the *program counter*, and is part of the state along with the values of the program variables. Instead of program operators, there are temporal operators that describe how the program variables, including the program counter, change with time. Thus Temporal Logic sacrifices compositionality for a less restricted formalism.

Readers should be able to readily discern that the explanation of Harel et al. gives the impression that temporal logic as a whole is somehow intrinsically limited to being endogenous. The authors do not mention research exploring exogenous uses of temporal logics to reason about imperative program behaviour. However several earlier publications on this subject by us and others were already available at the time (e.g., [13, 16, 23, 24, 47–51]). Most of these appeared significantly before the summary appeared. Unlike Dynamic Logic, this ITL-based work does not have separate notations for programs and formulas. Our range of new and fundamental mathematical results about 2-to-1 formulas and time symmetry are a direct continuation of the research on the exogenous use of ITL to express imperative programming constructs. This is a topic we have been pursuing since the 80s.

More recent work by Duan et al. [17] and the KIV theorem prover group [7] concerns exogenous uses of variants of ITL for concurrent algorithms.

15.3 2-to-1 formulas and the assumption of discrete time

Our central Theorem 4.5 states that 2-to-1 formulas are closed under the temporal operator $\Box$. Observe that the proof there requires that time is linear but does not at all depend on it being discrete. The theorem is even applicable to a restricted version of PITL consisting of conventional propositional logic with the sole addition of the temporal operator weak chop. Now $\Box$ is the only other temporal operator needed to formalise basic $\Box$-closure of 2-to-1 formulas. It is not hard to derive $\Box$ from weak chop (as described in our earlier publications [48–51]):

$$\mathit{inf} \mathrel{\widehat{=}} \mathit{true}; \mathit{false} \qquad \mathit{finite} \mathrel{\widehat{=}} \neg \mathit{inf} \qquad \Diamond A \mathrel{\widehat{=}} \mathit{finite}; A \qquad \Box A \mathrel{\widehat{=}} \neg\Diamond\neg A.$$

So Theorem 4.5 seems quite basic in the theory of temporal logic.

We can alternatively take strong chop as a primitive to obtain 2-to-1 formulas using Lemma 4.2's second characterisation of them (i.e., $\models (A^\frown A) \supset A$). The PTL temporal operator $\Box$ is then derivable as shown in Table 1 in Sect. 2 in order to formalise $\Box$-closure of 2-to-1 formulas.

Our comments here about Theorem 4.5 not requiring discrete time also apply to the analogous Lemma 10.5 concerning the closure of +-to-1 formulas under $\Box$.

If we take *skip* and either weak or strong chop as the two temporal primitives, then the derived operator *until* defined in Sect. 3.3 (and used to express 2-to-1 formulas in Sects. 5 and 14) is expressible without chop-star by means of the following semantic equivalence:

$$\models \quad T\,\mathit{until}\ A \quad \equiv \quad \big(\mathit{finite} \wedge \Box(\mathit{more} \supset T)\big); A$$

This uses the PTL subformula $\Box(\mathit{more} \supset T)$ instead of the PITL subformula $(\mathit{skip} \wedge T)^\star$ in the original definition of $T\,\mathit{until}\ A$ in Sect. 3.3. The two subformulas are semantically equivalent because of the valid PITL equivalence below (we formally state and prove this in [55, Theorem 5.4]):

$$\models \quad \Box(\mathit{more} \supset T) \quad \equiv \quad (\mathit{skip} \wedge T)^\star.$$

Note that our technique for defining *until* as a derived operator using *skip* assumes discrete time. However, the second definition of *until* can be made to work without discrete time if the left operand is limited to being a state formula (e.g., $p\,\mathit{until}\ \Box q$). We can simply take either *more* or *empty* to be a primitive operator. Alternatively, if chop-star is taken to be a primitive, then we first derive *empty* from chop-star as $\mathit{false}^\star$ and then derive *more* (which is normally defined using *skip*) using $\neg \mathit{empty}$.

15.4 Empty intervals

We have for about thirty years used the adjective "empty" to describe one-state intervals in ITL. Some readers will surely find this convention a bit puzzling because, in contrast, the empty word in regular languages has no letters at all. Let us now examine the choice of terminology. This also helps explain the behaviour of chop-star with one-state intervals, which is a further source of confusion.

In language theory, the empty word is the unique word with no letters at all. However, since the time of our early work on ITL [45–47], we have alway let the derived construct *empty* (defined in Table 1 in Sect. 2) denote the test for one-state intervals, which are also known in ITL as **empty intervals**. In fact, intervals in ITL and PTL with finite time always have at least one state, so there is normally no ambiguity about the meaning of the word "empty" in these logics.

Another reason why the term "empty" seems reasonable is because one-state intervals in fact play a role in ITL quite similar to empty words in regular languages and the standard finite-state automata associated with them. For example, some of our proofs of axiomatic completeness for versions of ITL [52, 53, 57] use such automata to encode ITL formulas, but the operation of the automata is modified so that they always examine at least one letter. Such a letter represents both an individual state and a one-state interval. The appropriateness of using "empty" for one-state intervals can also be clearly seen by means of a comparison of Kleene star with chop-star's semantics on finite intervals:

- **Standard definition of Kleene star on a regular language L:** Define L^0 to be the singleton set $\{\epsilon\}$ containing just the empty word ϵ. For each $k \geq 0$, inductively define L^{k+1} to be the set of finite words $\{\alpha\beta : \alpha \in L, \beta \in L^k\}$, where $\alpha\beta$ is the usual string concatenation of words α and β. The language L^* is defined to be the infinitary union of these sets: $\bigcup_{k\geq 0} L^k$.
- **Semantics of PITL's chop-star for finite intervals:** For any PITL formula A, we can analogously define A^0 to be the formula *empty* and for each $k \geq 0$, the formula A^{k+1} to be $A^\frown A^k$. For each $k \geq 0$, let S_k denote the set of finite intervals which satisfy the formula A^k and let S' denote the set of finite intervals which satisfy the chop-star formula $A^\star$ (as defined in Sect. 2). Then the two sets $\bigcup_{k\geq 0} S^k$ and S' can be shown to be equal.

Observe that the set obtained from the application of Kleene star to a regular language, even the empty one $\{\}$ with no words in it, always contains the empty word ϵ. This is because the language L^0 equals $\{\epsilon\}$ for every L, so L^* also includes ϵ as an element. Similarly, if one applies chop-star to a PITL formula A, the result $A^\star$ is satisfied by all one-state (empty) intervals, even if A itself is unsatisfiable. The formula $\mathit{false}^\star$ therefore provides a natural alternative way to express *empty* using just the boolean formula *false* combined with the temporal operator chop-star.

Duan [13–15, 17] and Bowman and Thompson [10] follow our convention of using *empty*, although Duan recently abbreviates it as ε [18, 83]. We should point out that some other naming conventions for the formula *empty* nevertheless also exist. For example, Paech uses the construct L_0 [61]. This follows a convention found in some earlier work by others on process logics [26, 62]. The formula $\ell = 0$ is favoured in the Duration Calculus [59, 84, 85], where the special construct ℓ

equals interval length. The formula $\ell = 0$ can be abbreviated as $[\![\ ]\!]$. The KIV group use the construct *last* to specify one-state intervals [7].

15.5 Star-to-1 formulas and chop-star fixpoints

The $*$-to-1 formulas defined in the beginning of Sect. 10 (i.e., $\models A^* \supset A$ in Definition 10.1) are identical to the ones called **chop-star fixpoints** in our earlier work on compositional reasoning in ITL [48–51]. A chop-star fixpoint is any formula A for which the equivalence $A \equiv A^*$ is valid. Now for any PITL formula A, the implication $A \supset A^*$ is valid. Hence, A is $*$-to-1 iff the equivalence $A \equiv A^*$ is valid. We present an analysis of $*$-to-1 formulas which relates them to other classes of formulas in recent work [58] that further explores the theory of 2-to-1 formulas.

16 Related work

We now consider relevant research by others and limit our coverage to the categories below:

– Mirror images
– Early proposals for using temporal logic with past time to reason about concurrency
– Interval-based approaches for analysing mutual exclusion

More information about other formal ways to analyse mutual exclusion, including extensive bibliographies, can be found in the various recent textbooks we already cited at the beginning of Sect. 13 when justifying our choice of Peterson's algorithm to illustrate time symmetry.

16.1 Mirror images

Time reversal and reflections are related to *mirror images* (see Prior [67]) used with temporal logics to obtain a rule for past-time operators from an analogous one for future-time operators by means of time symmetry. Analyses of conventional temporal logics for computer science typically cannot directly exploit mirror images because the time models are intentionally *asymmetric* with an infinite future and either no past or a bounded one. That has severely limited the application of mirror images. Nevertheless, Furia and Spoletini [21] and Reynolds [69] have recently applied mirror images to symmetric time models (e.g., bounded past and future). This demonstrates ongoing interest in mirror images and associated techniques.

16.2 Early applications of temporal logic with past time to concurrency

Our presentation already mentioned in Sect. 6 the work by Pnueli [66] and Lichtenstein, Pnueli and Zuck [41] in the mid 80s which formalises safety properties using temporal formulas of the form $\Box(w \supset X)$, where the only temporal operators in X are past-time ones such as those described in Sect. 5.1. Therefore, X can just concern past states and the current state, but not future ones. Only Pnueli [66] specifically discusses mutual exclusion and Peterson's algorithm. One motivation for using past time is to assist in doing backward analysis about what must have preceded certain events. We pointed in Sect. 6 out that the formula $\Box(w \supset X)$ bears a certain resemblance to our class of 2-to-1 PITL formulas having the form $\boxed{\text{f}}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ and showed how some instances can be formally related in a semantic sense. One example of this given in Sect. 6 concerns the PITL formula (16) and PTL$^-$ formula (17) which we reproduce below for the convenience of readers:

$$\boxed{\text{f}}\big((\mathit{fin}\, p) \supset \Diamond(\mathit{skip} \wedge q)\big) \qquad \mathit{first} \;\;\wedge\;\; \Box(p \supset \circleddash q).$$

Pnueli's main justification given for past time is that point-based temporal logic without past-time constructs imposes a more *global* view of the system behaviour. In contrast, past-time constructs help to *modularly* specify and analyse the behaviour of an individual process.

Interestingly, around the same time as Pnueli, both Barringer and Kuiper [3, 4] and Koymans, Vytopil and de Roever [36] similarly suggest the use of past-time constructs for reasoning about concurrency. They do not discuss ones of the form $\Box(w \supset X)$. However, Barringer, Kuiper and Pnueli in the slightly later joint paper [5] mention a couple of formulas of this kind and also examine mutual exclusion and Peterson's algorithm.

The straightforward definitions of satisfiability and validity we use for PTL$^-$ in Sect. 5.1 correspond to the so-called *floating framework* of PTL with past time. However, Manna and Pnueli propose another approach called the *anchored framework* [42] (also discussed by Lichtenstein and Pnueli in [40]) which they argue is superior. In this framework, satisfiability and validity only examine pairs of the form $(\sigma, 0)$. There exist ways to go between the two conventions, but we will not delve into this here and instead simply assume the more traditional floating interpretation.

16.3 Other interval-based analyses of mutual exclusion

We now mention some interval-based work involving algorithms for mutual exclusion and the related topic of lock-free data structures. The only previously published analysis of Peterson's algorithm in some ITL variant seems to be the one by Pei and Xu [63] which uses the *Discrete Time Duration Calculus* [25,84]. Verification is performed using model checking with the popular SPIN tool [31] and is global rather than modular in the sense of Pnueli [66] (as we briefly discussed in Sect. 16.2).

Projection Temporal Logic is an ITL extension with operators for temporal granularities and framing [13–15, 17]. Duan [13, 14] expresses Dekker's mutual exclusion algorithm (first published by Dijkstra in [12]) in Projection Temporal Logic but without any formal analysis. Yang, Duan and Ma [82] have applied Projection Temporal Logic to the analysis of an mutual exclusion example involving a counter and described earlier by Biere et al. [8]. The interactive theorem prover PVS [60] provides tool support for a global proof in Pnueli's sense involving the combined behaviour of two concurrent processes. Consequently, no compositional properties involving the correctness of the individual processes are given. Duan, Zhang and

Koutny [18] investigate axiomatic completeness for propositional Projection Temporal Logic and illustrate their framework by summarising the global analysis of another mutual exclusion example. In principle, Projection Temporal Logic supports past-time constructs, so a modular analysis, at least in Pnueli's sense, seems feasible. However, the case studies of mutual exclusion in [18, 82] are formalised in a version of the logic where the only past-time construct, the operator $\ominus$ ("previous"), seems intended solely for framing variables. Our techniques involving time symmetry and compositional formulas which are closed under conjunction and the temporal operator $\Box$ might also be applicable to analysis involving Projection Temporal Logic because it supports basic ITL operators.

The KIV interactive theorem prover group [7, 79] has combined a variant of ITL with the *rely-guarantee* paradigm [34, 35] of Jones to verify lock-free algorithms. However, they have not yet looked at mutual exclusion. Moreover, the lack of much published literature on applying the quite established rely-guarantee approach to mutual exclusion (e.g., Stark [74] and see also the related work of Stølen [75] and Collette [11]) suggests that the framework is not particularly well suited for it. The textbook by de Roever et al. [70] presents a rely-guarantee example involving mutual exclusion and is a comprehensive source of information about compositional reasoning based on rely-guarantee conditions as well as other similar work.

We take this opportunity to also mention a class of ITL formulas which Siewe et al. [73] and Janicke et al. [33] use for describing access control policies. Such formulas have the form given below:

$$\boxed{\scriptstyle f}\big((\Diamond B) \supset \mathit{fin}\, w\big). \tag{99}$$

Their syntax makes them similar to the 2-to-1 formula $\boxed{\scriptstyle f}\big((\mathit{fin}\, w) \supset \Diamond B\big)$ we first discussed in Sect. 6. However, the variant (99) is not necessarily 2-to-1. For example, consider any three-state interval σ. It has exactly two two-state subintervals. Each of these subintervals trivially satisfies the next $\boxed{\scriptstyle f}$-formula:

$$\boxed{\scriptstyle f}\Big(\big(\Diamond(\mathit{skip}^\frown \mathit{skip})\big) \;\supset\; \mathit{fin}\, \mathit{false}\Big). \tag{100}$$

This is because a one- or two-state interval does not satisfy the implication's left operand $\Diamond(\mathit{skip}^\frown \mathit{skip})$, which is only true for intervals with three or more states. Hence, the implication's right operand $\mathit{fin}\, \mathit{false}$ is ignored. Now let A denote the $\boxed{\scriptstyle f}$-formula (100). Our reasoning so far about the subintervals ensures that the three-state interval σ satisfies the chop formula $A; A$. Nevertheless, σ fails to satisfy A because the left subformula $\Diamond(\mathit{skip}^\frown \mathit{skip})$ of the implication in A is true in σ, but the right subformula $\mathit{fin}\, \mathit{false}$ is not. Hence, σ does not satisfy the implication $(A; A) \supset A$, so the formula (100) is not 2-to-1.

Conclusions and Further Work

We believe that our results about interval-based compositional reasoning using 2-to-1 formulas and time symmetry are promising. The various compositional classes of formulas described here which are closed under conjunction and the temporal operator $\Box$ seem quite intriguing owing to their simple mathematical features and natural connections with PTL. The approach therefore appears worthy of further study. Moreover, perhaps the application of the compositional classes and time symmetry can even somewhat narrow the currently perceived wide practical gap between PITL and PTL and help increase combined use of the two formalisms. Possible connections could also be explored involving temporal logics with the same expressiveness as PITL but lower computational complexity such as *Regular Linear Temporal Logic* proposed by Leucker and Sánchez [39, 71]. Incidentally, PITL itself contains a natural, equally expressive sublogic of this sort called *Fusion Logic* [54, 55], which has some tool support.

Ideally, we would also like to see a comparative analysis encompassing a number of suitable benchmark applications, range of formalisms and models of concurrency such as interleaving and true concurrency. It furthermore seems appropriate to evaluate the tradeoffs between analyses involving concrete algorithms and more abstract ones. The compositional details required in our analysis of Peterson's algorithm certainly suggest to us that abstraction can be quite beneficial. We have clearly focused our attention on modular techniques here, but the nature of both global and modular ones deserves further investigation.

Our future research plans include using 2-to-1 formulas and time symmetry in a calculus of sequential and parallel composition based on Hoare triples having assertions expressed in ITL. Implementations of decision procedures for PITL using time symmetry and reductions to point-based temporal logic are also envisioned.

We end our discussion here by noting that we believe that the basic mathematical concepts described here enrich the body of knowledge about intervals and temporal logics, no matter what the ultimate practical implications might be. They include some elementary and exciting properties about compositionality and time symmetry which turned out with the hindsight of several decades to be quite elusive and so until now were completely overlooked and unexplored. It also seems remarkable that most of them only involve the subset of PITL with just the temporal operators chop and *skip*, but not chop-star. Perhaps similar treasures still remain hidden, waiting to be discovered. We believe that the further systematic and scientific study and application of the compositional techniques we already presented in our earlier work [48–51], together with our interval-oriented analysis of conventional point-based linear time temporal [55] and new completeness proof for PITL with infinite time [57], could help in the exploration. This view is supported by the fact that the material in these publications played a crucial part in leading us to uncovering the results we have described here.

Acknowledgements We would like to thank Antonio Cau, Amin El-kustaban, Helge Janicke, Maciej Koutny, Sven Schewe, Xiaoxiao Yang and anonymous referees for comments. Shirley Craig's outstanding library services deserve special mention.

References

1. Aceto, L., Ingólfsdóttir, A., Larsen, K.G., Srba, J.: Reactive Systems: Modelling, Specification and Verification. Cambridge University Press (2007)
2. Balser, M., Bäumler, S., Knapp, A., Reif, W., Thums, A.: Interactive verification of UML state machines. In: J. Davies, W. Schulte, M. Barnett (eds.) Proc. 6th International Conference on Formal Engineering Methods (ICFEM 2004), *LNCS*, vol. 3308, pp. 434–448. Springer-Verlag (2004)
3. Barringer, H., Kuiper, R.: Hierarchical development of concurrent systems in a temporal logic framework. In: S.D. Brookes, A.W. Roscoe, G. Winskel (eds.) Seminar on Concurrency, *LNCS*, vol. 197, pp. 35–61. Springer-Verlag (1985)

4. Barringer, H., Kuiper, R.: Towards the hierarchical, temporal logic, specification of concurrent systems. In: B. Denvir, W. Harwood, M. Jackson, M. Wray (eds.) The Analysis of Concurrent Systems, *LNCS*, vol. 207, pp. 157–183. Springer-Verlag (1985)
5. Barringer, H., Kuiper, R., Pnueli, A.: A really abstract concurrent model and its temporal logic. In: Proc. 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'86), pp. 173–183. ACM (1986)
6. Bäumler, S., Balser, M., Nafz, F., Reif, W., Schellhorn, G.: Interactive verification of concurrent systems using symbolic execution. AI Communications **23**(2–3), 285–307 (2010)
7. Bäumler, S., Schellhorn, G., Tofan, B., Reif, W.: Proving linearizability with temporal logic. Formal Aspects of Computing **23**(1), 91–112 (2011)
8. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers **58**, 117–148 (2003)
9. Bowman, H., Cameron, H., King, P., Thompson, S.: Mexitl: Multimedia in Executable Interval Temporal Logic. Formal Methods in Systems Design **22**(1), 5–38 (2003)
10. Bowman, H., Thompson, S.J.: A decision procedure and complete axiomatization of finite Interval Temporal Logic with projection. Journal of Logic and Computation **13**(2), 195–239 (2003)
11. Collette, P.: Composition of assumption-commitment specifications in a UNITY style. Science of Computer Programming **23**(2-3), 107–125 (1994)
12. Dijkstra, E.W.: Cooperating sequential processes. In: F. Genuys (ed.) Programming Languages: NATO Advanced Study Institute, pp. 43–112. Academic Press (1968)
13. Duan, Z.: An extended interval temporal logic and a framing technique for temporal logic programming. Ph.D. thesis, Dept. of Computing Science, University of Newcastle Upon Tyne (1996). Technical report 556, later published as [14]
14. Duan, Z.: Temporal Logic and Temporal Logic Programming. Science Press, Beijing, China (2005). Published version of [13]
15. Duan, Z., Koutny, M.: A framed temporal logic programming language. Journal of Computer Science and Technology **19**(3), 341–351 (2004)
16. Duan, Z., Koutny, M., Holt, C.: Projection in temporal logic programming. In: F. Pfenning (ed.) Proc. of Logic Programming and Automated Reasoning (LPAR '94), *LNCS*, vol. 822, pp. 333–344. Springer-Verlag, Berlin (1994)
17. Duan, Z., Yang, X., Koutny, M.: Framed temporal logic programming. Science of Computer Programming **70**(1), 31–61 (2008)
18. Duan, Z., Zhang, N., Koutny, M.: A complete axiomatization of propositional projection temporal logic. Theor. Comp. Sci. (2012). DOI `10.1016/j.tcs.2012.01.026`
19. Fisher, M.: An Introduction to Practical Formal Methods Using Temporal Logic. John Wiley & Sons (2011)
20. Floyd, R.W.: Assigning meanings to programs. In: J.T. Schwartz (ed.) Proc. AMS Symp. on Applied Mathematics 19, pp. 19–32. American Mathematical Society, Providence, Rhode Island, USA (1967)
21. Furia, C.A., Spoletini, P.: Tomorrow and all our yesterdays: MTL satisfiability over the integers. In: J.S. Fitzgerald, A.E. Haxthausen, H. Yenigün (eds.) 5th International Colloquium on Theoretical Aspects of Computing (ICTAC 2008), *LNCS*, vol. 5160, pp. 126–140. Springer-Verlag (2008)
22. Gómez, R., Bowman, H.: PITL2MONA: Implementing a decision procedure for propositional Interval Temporal Logic. Journal of Applied Non-Classical Logics **14**(1–2), 105–148 (2004). Special issue on Interval Temporal Logics and Duration Calculi. V. Goranko and A. Montanari, guest editors
23. Hale, R.: Temporal logic programming. In: A. Galton (ed.) Temporal Logics and Their Applications, pp. 91–119. Academic Press, London (1987)
24. Hale, R.W.S.: Programming in temporal logic. Ph.D. thesis, Computer Laboratory, Cambridge University, Cambridge, England (1988). Appeared in 1989 as Technical report 173
25. Hansen, M.R., Zhou Chaochen: Duration calculus: Logical foundations. Formal Aspects of Computing **9**(3), 283–330 (1997)
26. Harel, D., Kozen, D., Parikh, R.: Process Logic: Expressiveness, decidability, completeness. Journal of Computer and System Sciences **25**(2), 144–170 (1982)
27. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press, Cambridge, Mass. (2000)
28. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. In: D. Gabbay, F. Guenthner (eds.) Handbook of Philosophical Logic, vol. 4, 2nd edn., pp. 99–217. Kluwer Academic Publishers, Dordrecht (2002)

29. Herlihy, M., Shavit, N.: The Art of Multiprocessor Programming. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2008)
30. Hoare, C.A.R.: An axiomatic basis for computer programming. Communications of the ACM **12**(10), 576–580,583 (1969)
31. Holzmann, G.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley Professional (2003)
32. Interval Temporal Logic web pages. `http://www.tech.dmu.ac.uk/STRL/ITL/`
33. Janicke, H., Cau, A., Siewe, F., Zedan, H., Jones, K.: A compositional event & time-based policy model. In: Proceedings of POLICY2006, London, Ontario, Canada, pp. 173–182. IEEE Computer Society Press (2006)
34. Jones, C.B.: Specification and design of (parallel) programs. In: R.E.A. Mason (ed.) Proc. IFIP Congress '83, pp. 321–332. North Holland Publishing Co., Amsterdam (1983)
35. Jones, C.B.: Tentative steps toward a development method for interfering programs. ACM Transactions on Programming Languages and Systems **5**(4), 596–619 (1983)
36. Koymans, R., Vytopil, J., de Roever, W.P.: Real-time programming and asynchronous message passing. In: Proceedings of the Second Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC'83), pp. 187–197 (1983)
37. Kröger, F., Merz, S.: Temporal Logic and State Systems. Texts in Theoretical Computer Science (An EATCS Series). Springer-Verlag (2008)
38. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Professional (2002)
39. Leucker, M., Sánchez, C.: Regular Linear Temporal Logic. In: C.B. Jones, Z. Liu, J. Woodcock (eds.) Proc. 4th International Colloquium on Theoretical Aspects of Computing (ICTAC'07), Macau, China, *LNCS*, vol. 4711, pp. 291–305. Springer-Verlag (2007)
40. Lichtenstein, O., Pnueli, A.: Propositional temporal logics: Decidability and completeness. Logic Journal of the IGPL **8**(1), 55–85 (2000)
41. Lichtenstein, O., Pnueli, A., Zuck, L.: The glory of the past. In: R. Parikh, et al. (eds.) Logics of Programs, *LNCS*, vol. 193, pp. 196–218. Springer-Verlag, Berlin (1985)
42. Manna, Z., Pnueli, A.: The anchored version of the temporal framework. In: J.W.D. Bakker, W.P. de Roever, G. Rozenberg (eds.) Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency (REX Workshop 1988), *LNCS*, vol. 354, pp. 201–284. Springer-Verlag (1989)
43. McCarthy, J., Hayes, P.J.: Some philosophical problems from the standpoint of artificial intelligence. In: D. Michie, B. Meltzer (eds.) Machine Intelligence 4, pp. 463–502. Edinburgh University Press, Edinburgh (1969). Reprinted in [81, 431–450]
44. Mo, D., Wang, X., Duan, Z.: Asynchronous communication in MSVL. In: S. Qin, Z. Qiu (eds.) 13th Int'l Conf. on Formal Engineering Methods (ICFEM 2011), *LNCS*, vol. 6991, pp. 82–97. Springer-Verlag (2011)
45. Moszkowski, B.: Reasoning about digital circuits. Ph.D. thesis, Department of Computer Science, Stanford University (1983). Technical report STAN–CS–83–970
46. Moszkowski, B.: A temporal logic for multilevel reasoning about hardware. Computer **18**, 10–19 (1985)
47. Moszkowski, B.: Executing Temporal Logic Programs. Cambridge University Press, Cambridge, England (1986)
48. Moszkowski, B.: Some very compositional temporal properties. In: E.R. Olderog (ed.) Programming Concepts, Methods and Calculi (PROCOMET'94), *IFIP Transactions*, vol. A-56, pp. 307–326. IFIP, Elsevier Science B.V. (North–Holland) (1994)
49. Moszkowski, B.: Compositional reasoning about projected and infinite time. In: Proc. 1st IEEE Int'l Conf. on Engineering of Complex Computer Systems (ICECCS'95), pp. 238–245. IEEE Computer Society Press (1995)
50. Moszkowski, B.: Using temporal fixpoints to compositionally reason about liveness. In: He Jifeng, J. Cooke, P. Wallis (eds.) BCS-FACS 7th Refinement Workshop, electronic Workshops in Computing. BCS-FACS, Springer-Verlag and British Computer Society, London (1996)
51. Moszkowski, B.: Compositional reasoning using Interval Temporal Logic and Tempura. In: W.P. de Roever, H. Langmaack, A. Pnueli (eds.) Compositionality: The Significant Difference, *LNCS*, vol. 1536, pp. 439–464. Springer-Verlag, Berlin (1998)
52. Moszkowski, B.: An automata-theoretic completeness proof for Interval Temporal Logic (extended abstract). In: U. Montanari, J. Rolim, E. Welzl (eds.) Proc. 27th Int'l. Colloquium on Automata, Languages and Programming (ICALP 2000), *LNCS*, vol. 1853, pp. 223–234. Springer-Verlag, Geneva, Switzerland (2000)

53. Moszkowski, B.: A complete axiomatization of Interval Temporal Logic with infinite time (extended abstract). In: Proc. 15th Ann. IEEE Symp. on Logic in Computer Science (LICS 2000), pp. 242–251. IEEE Computer Society Press (2000)
54. Moszkowski, B.: A hierarchical completeness proof for Propositional Interval Temporal Logic with finite time. Journal of Applied Non-Classical Logics **14**(1–2), 55–104 (2004). Special issue on Interval Temporal Logics and Duration Calculi. V. Goranko and A. Montanari, guest editors.
55. Moszkowski, B.: Using temporal logic to analyse temporal logic: A hierarchical approach based on intervals. Journal of Logic and Computation **17**(2), 333–409 (2007)
56. Moszkowski, B.: Compositional reasoning using intervals and time reversal. In: 18th Int'l Symp. on Temporal Representation and Reasoning (TIME 2011), pp. 107–114. IEEE Computer Society (2011)
57. Moszkowski, B.: A complete axiom system for propositional Interval Temporal Logic with infinite time. Logical Methods in Computer Science **8**(3:10), 1–56 (2012)
58. Moszkowski, B.: Interconnections between classes of sequentially compositional temporal formulas. Inf. Process. Lett. **113**(9), 350–353 (2013)
59. Olderog, E.R., Dierks, H.: Real-Time Systems: Formal Specification and Automatic Verification. Cambridge University Press, Cambridge, England (2008)
60. Owre, S., Shankar, N.: A brief overview of PVS. In: O.A. Mohamed, C. Muñoz, S. Tahar (eds.) 21st International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2008), *LNCS*, vol. 5170, pp. 22–27. Springer-Verlag (2008)
61. Paech, B.: Gentzen-systems for propositional temporal logics. In: E. Börger, H.K. Büning, M.M. Richter (eds.) Proceedings of the 2nd Workshop on Computer Science Logic (CSL'88), *LNCS*, vol. 385, pp. 240–253. Springer-Verlag (1989)
62. Parikh, R., Chandra, A.K., Halpern, J.Y., Meyer, A.R.: Equations between regular terms and an application to process logic. SIAM Journal on Computing **14**(4), 935–942 (1985)
63. Pei Yu, Xu Qiwen: Checking interval based properties for reactive systems. In: B. Steffen, G. Levi (eds.) Verification, Model Checking, and Abstract Interpretation, *LNCS*, vol. 2937, pp. 51–75. Springer-Verlag (2004)
64. Peterson, G.L.: Myths about the mutual exclusion problem. Inf. Process. Lett. **12**(3), 115–116 (1981)
65. Pnueli, A.: The temporal logic of programs. In: Proc. 18th Ann. IEEE Symp. on the Foundation of Computer Science (FOCS), pp. 46–57. IEEE Computer Society Press (1977)
66. Pnueli, A.: In transition from global to modular temporal reasoning about programs. In: K.R. Apt (ed.) Logics and Models of Concurrent Systems, *NATO ASI Series F*, vol. 13, pp. 123–144. Springer-Verlag (1985)
67. Prior, A.: Past, Present and Future. Oxford Univ. Press, London (1967)
68. Reif, W., Schellhorn, G., Stenzel, K., Balser, M.: Structured specifications and interactive proofs with KIV. In: W. Bibel, P.H. Schmitt (eds.) Automated Deduction – A Basis for Applications, Volume II: Systems and Implementation Techniques, pp. 13–39. Kluwer Academic Publishers, Dordrecht (1998)
69. Reynolds, M.: A tableau for Until and Since over linear time. In: 18th Int'l Symp. on Temporal Representation and Reasoning (TIME 2011), pp. 41–48. IEEE Computer Society (2011)
70. de Roever, W.P., de Boer, F., Hanneman, U., Hooman, J., Lakhnech, Y., Poel, M., Zwiers, J.: Concurrency Verification: Introduction to Compositional and Noncompositional Methods. No. 54 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (2001)
71. Sánchez, C., Leucker, M.: Regular Linear Temporal Logic with past. In: 11th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI 2010), *LNCS*, vol. 5944, pp. 295–311. Springer-Verlag (2010)
72. Shanahan, M.: Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia. MIT Press (1997)
73. Siewe, F., Cau, A., Zedan, H.: A compositional framework for access control policies enforcement. In: M. Backes, D. Basin, M. Waidner (eds.) ACM Workshop on Formal Methods in Security Engineering (FMSE'03), pp. 32–42. ACM Press, Washington, DC (2003)
74. Stark, E.W.: A proof technique for rely/guarantee properties. In: Proceedings of the 5th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1985), *LNCS*, vol. 206, pp. 369–391. Springer-Verlag (1985)
75. Stølen, K.: A method for the development of totally correct shared-state parallel programs. In: CONCUR 1991, *LNCS*, vol. 527, pp. 510–525. Springer-Verlag (1991)

76. Taubenfeld, G.: Synchronization Algorithms and Concurrent Programming. Pearson/Prentice Hall (2006)
77. Thomas, W.: Automata on infinite objects. In: J. van Leeuwen (ed.) Handbook of Theoretical Computer Science, vol. B: Formal Models and Semantics, chap. 4, pp. 133–191. Elsevier/MIT Press, Amsterdam (1990)
78. Thums, A., Schellhorn, G., Ortmeier, F., Reif, W.: Interactive verification of Statecharts. In: H. Ehrig, W. Damm, J. Desel, M. Große-Rhode, W. Reif, E. Schnieder, E. Westkämper (eds.) SoftSpez Final Report, *LNCS*, vol. 3147, pp. 355–373. Springer-Verlag (2004)
79. Tofan, B., Bäumler, S., Schellhorn, G., Reif, W.: Temporal logic verification of lock-freedom. In: Proc. MPC 2010, Springer LNCS 6120, pp. 377–396 (2010)
80. van Emde Boas, P.: The connection between Modal Logic and Algorithmic Logic. In: 7th Symposium on Mathematical Foundations of Computer Science (MFCS 1978), *lncs*, vol. 64, pp. 1–15. springer (1978)
81. Webber, L., Nilsson, N.J. (eds.): Readings in Artificial Intelligence. Tioga Publishing Co., Palo Alto, California (1981)
82. Yang, X., Duan, Z., Ma, Q.: Axiomatic semantics of projection temporal logic programs. Mathematical Structures in Computer Science **20**(5), 865–914 (2010)
83. Zhang, N., Duan, Z., Tian, C.: A cylinder computation model for many-core parallel computing. Theor. Comp. Sci. (2012). DOI `10.1016/j.tcs.2012.02.011`
84. Zhou Chaochen, Hansen, M.R.: Duration Calculus: A Formal Approach to Real-Time Systems. Monographs in Theoretical Computer Science (An EATCS series). Springer-Verlag (2004)
85. Zhou Chaochen, Hoare, C.A.R., Ravn, A.P.: A calculus of durations. Inf. Process. Lett. **40**(5), 269–276 (1991)

This author-produced version was formatted on 26 July 2013.