URL: http://www.elsevier.nl/locate/entcs/volume7.html 22 pages

Specification in CTL+Past, Verification in CTL

F. Laroussinie, Ph. Schnoebelen

Lab. Specification and Verification
ENS de Cachan & CNRS URA 2236
61, av. Pdt Wilson; 94235 Cachan Cedex; FRANCE

Abstract

We describe $PCTL$, a temporal logic extending CTL with connectives allowing to refer to the past of a current state. This incorporates the new N, "From Now On", combinator we recently introduced.

$PCTL$ has branching future but determined, finite and cumulative past. We argue this is the right choice for a semantical framework, and show this through an extensive example.

Finally, we demonstrate how a translation-based approach allows model-checking specification written in $\mathbf{N}CTL$, a fragment of $PCTL$.

1 Introduction

Temporal Logic. Following Pnueli's pioneering work, the *temporal logic (TL) framework* has long been recognized as a fundamental approach to the formal specification and verification of reactive systems [20,5]. TL allows precise and concise statements of complex behavioral properties. Additionally, it supports the very successful *model-checking technology* that allows large and complex (finite) systems to be verified automatically [3,4,21].

Still, TL has its well-known limitations. Here we are concerned with its *limitations in expressive power*, both in a practical and in a theoretical sense. On the theoretical side, it is well-known that not all interesting behavioral properties can be expressed in the most commonly used temporal logics. On the practical side, it is well-known that not all expressible properties can be expressed in a simple and natural way, so that specifications are often hard to read and error-prone. A typical situation is that some temporal properties are more easily written in first-order logic over time points, or in an automata-theoretic framework, than in temporal logic.

Past-time. Ever since [18] it has been known that allowing both past-time and future-time constructs makes TL specification easier and more natural: the English sentence "*if a crash occurs, then necessarily a mistake took place* *earlier*" is directly rendered by $\Box(crash \Rightarrow \Diamond^{-1} mistake)$. If we don't allow past-time constructs, we may end up with the clumsier $\neg\exists(\neg mistake \mathsf{U}\, crash)$.

Today there exists a huge body of literature where a variety of TL's with past are used to specify systems (less frequently to verify them and even less frequently to model-check them). Surprisingly, these proposals use quite different semantics for past, and the reasons behind the semantical choices are not discussed in depth.

Model-checking with Past. Only a few papers (e.g. [25,11,13]) propose model-checking algorithms for a TL with past. None of the widely available model-checking tools supports past-time constructs.

Translation between logics. Instead of building new model-checking tools for TL with past, we suggest an alternative, so-called *translation-based, approach* [16,17]: larger logics are translated into CTL (or related logics), so that the existing model-checkers, e.g. SMV [22], can be used with no adaptation at all. Contrasting its many advantages, the main drawback of this approach is that the diagnostic a model-checker sometimes provides refers to its input formula, i.e. the translated formula and not the original formula written by a human specifier.

Translations between past-and-future logics into pure-future logics have been known since [8]. They were used to argue that past-time does not add theoretical expressivity. They were not suggested as an actual practical approach to the model-checking problem for extended logics.

Our contribution. In this paper, we extend our previous results [16] in several directions : we prove a translation theorem for $\mathbf{N}CTL$, a fragment of $PCTL$ (i.e. $CTL + Past$) that extends the $CTL + \mathsf{F}^{-1}$ solved in [16] and we show that the translation is correct even in a framework with fairness.

By necessity, $\mathbf{N}CTL$ only permits a restricted use of the *Since* modality. We show, through an extensive example (the well-known Lift example [10]) that these restrictions are not too drastic in practice. Indeed, we only isolated the $\mathbf{N}CTL$ fragment as a by-product of writing our Lift specification in $PCTL$. This unexpected development was a good example of practical studies suggesting hard theoretical results.

Also, because the differences between semantic frameworks for Past are not much discussed in the literature, we take some time discussing them and classifying the different proposals we found.

Plan of the paper. We assume familiarity with CTL. Section 2 gives the syntax and semantics of $PCTL$. The semantical framework for past-time is discussed in section 3 where the main related works are categorized. Section 4 gives the lift specification. Section 5 presents the translation-based approach before section 6 defines $\mathbf{N}CTL$ and gives the translation theorem.

2 $PCTL$, or CTL+Past

Syntactically, the $PCTL$ logic we define is the $CTL + \mathsf{S} + \mathsf{X}^{-1} + \mathsf{N}$ of [16]. It inherits the syntactic restrictions of CTL (no nesting of linear-time combinators under the scope of a path quantifier) for the future-time part. Semantically, this logic is interpreted into Kripke structures with fairness while [16] only used structures without fairness.

2.1 Syntax

We assume a given non-empty finite set $Prop = \{a, b, \ldots\}$ of *atomic propositions*. $PCTL$ formulas are given by the following grammar:

$$\varphi, \psi \ ::= \ \neg\varphi \mid \varphi \wedge \psi \mid \mathsf{EX}\varphi \mid \mathsf{E}\varphi\mathsf{U}\psi \mid \mathsf{A}\varphi\mathsf{U}\psi \mid \mathsf{X}^{-1}\varphi \mid \varphi\mathsf{S}\psi \mid \mathsf{N}\varphi \mid a \mid b \mid \ldots$$

Here, the well-known future-only CTL logic is enriched with past-time constructs X^{-1} ("Previous"), S ("Since") and N ("From now on").

Standard abbreviations include $\top, \bot, \varphi \vee \psi, \varphi \Rightarrow \psi, \ldots$ as well as

$$\mathsf{EF}\varphi \stackrel{\text{def}}{=} \mathsf{E}\top\mathsf{U}\varphi \qquad \mathsf{EG}\varphi \stackrel{\text{def}}{=} \neg\mathsf{AF}\neg\varphi \qquad \mathsf{AX}\varphi \stackrel{\text{def}}{=} \neg\mathsf{EX}\neg\varphi$$
$$\mathsf{AF}\varphi \stackrel{\text{def}}{=} \mathsf{A}\top\mathsf{U}\varphi \qquad \mathsf{AG}\varphi \stackrel{\text{def}}{=} \neg\mathsf{EF}\neg\varphi \qquad \mathsf{F}^{-1}\varphi \stackrel{\text{def}}{=} \top\mathsf{S}\varphi$$

2.2 Semantics

$PCTL$ formulas are interpreted over *histories* (that is, a current state with a past) in Kripke structures with fairness constraints. Formally,

Definition 2.1 A fair Kripke structure (a "FKS") is a tuple $S = \langle Q_S, R_S, l_S, I_S, \Phi_S\rangle$ where

- $Q_S = \{q_1, \ldots\}$ is a non-empty set of *states*,
- $R_S \subseteq Q_S \times Q_S$ is a total *transition relation*,
- $l_S : Q_S \to 2^{Prop}$ labels every state with the propositions it satisfies,
- $I_S \subseteq Q_S$ is a set of *initial states*,
- Φ_S is a *fairness constraint* (see below).

In the rest of the paper, we drop the "S" subscript in our notations whenever no ambiguity will arise.

A *computation* in a FKS is an infinite sequence $q_0 q_1 \ldots$ s.t. $(q_i, q_{i+1}) \in R$ for all $i = 0, 1, \ldots$. Because R is total, any state can be the starting point of a computation. We use $\pi, \ldots$ to denote computations. As usual, $\pi(i)$ (resp. π^i) denotes the i-th state, q_i (resp. i-th suffix: $q_i q_{i+1}, \ldots$).

A *fair computation* in an FKS is a computation satisfying the fairness constraint, which is just some way of telling fair from unfair computations. Formally,

Definition 2.2 A *fairness constraint* (for S) is a predicate Φ on S-computations satisfying the following properties:
1. *fairness only depends on the "end" of a computation:* for all π and suffix π^n, $\Phi(\pi)$ iff $\Phi(\pi^n)$,
2. *any finite behaviour can be continued in a way ensuring fairness:* for all $\pi = q_0 q_1 \ldots$, for all $n \geq 0$, there exists a fair π' starting with $q_0 q_1 \ldots q_n$.

In practice, fairness constraints are always given through some precise mechanism (e.g. infinitely repeated states). We let $\Pi_S(q)$ denote the set of fair computations starting from q, and write $\Pi(S)$ for the union of all $\Pi_S(q)$.

An *history* is a **non-empty** finite sequence $q_0q_1\ldots q_n$ s.t. $(q_i, q_{i+1}) \in R$ for all $i < n$. We use $\sigma, \ldots$ to denote histories. Histories are prefix of computations. Given $i \geq 0$ and $\pi = q_0q_1\ldots$, we let $\pi_{|i}$ denote the i-th prefix of π, i.e. the history $q_0\ldots q_i$. By extension, we write $\Pi(\sigma)$ for the set of all fair computations starting from σ.

The intuition is that an history $\sigma = q_0q_1\ldots q_n$ denotes a current state q_n of some computation still in process, with the additional information that the past of this computation has been σ. From this history, the system can proceed to a next state q_{n+1} and then the past will be $\sigma' = q_0\ldots q_nq_{n+1}$. Any state q is a history (where the past is empty) by itself.

Figure 1 defines when a history σ, in some FKS S, satisfies a formula φ, written $\sigma \models_S \varphi$, by induction over the structure of φ.

$\sigma \models a$	iff $a \in l(q_n)$,
$\sigma \models \neg\varphi$	iff $\sigma \not\models \varphi$,
$\sigma \models \varphi \wedge \psi$	iff $\sigma \models \varphi$ and $\sigma \models \psi$,
$\sigma \models \mathsf{EX}\varphi$	iff there exists $\pi \in \Pi(\sigma)$ s.t. $\pi_{\|n+1} \models \varphi$,
$\sigma \models \mathsf{E}\varphi\mathsf{U}\psi$	iff there exists $\pi \in \Pi(\sigma)$ and $k \geq 0$
	s.t. $\pi_{\|n+k} \models \psi$ and $\pi_{\|n+i} \models \varphi$ for all $0 \leq i < k$,
$\sigma \models \mathsf{A}\varphi\mathsf{U}\psi$	iff for all $\pi \in \Pi(\sigma)$ there exists a $k \geq 0$
	s.t. $\pi_{\|n+k} \models \psi$ and $\pi_{\|n+i} \models \varphi$ for all $0 \leq i < k$,
$\sigma \models \mathsf{X}^{-1}\varphi$	iff $n > 0$ and $\sigma' \models \varphi$ (where $\sigma' = q_0\ldots q_{n-1}$),
$\sigma \models \varphi\mathsf{S}\psi$	iff there exists $k \leq n$ s.t. $\sigma_{\|k} \models \psi$ and
	$\sigma_{\|i} \models \varphi$ for all $k < i \leq n$,
$\sigma \models \mathsf{N}\varphi$	iff $q_n \models \varphi$.
when σ is $q_0, \ldots, q_n$.	

Fig. 1. Semantics of *PCTL*

Then satisfaction can be defined over fair Kripke structures through

$$S \models \varphi \overset{\text{def}}{\Leftrightarrow} \pi_{|0} \models \varphi \text{ for all } \pi \in \Pi_S(I_S)$$

adopting the anchored-view of satisfaction [19] common in TL specifications [5].

The semantics we just gave justifies the usual reading of combinators as $\mathsf{EF}\varphi$: "it is possible to have φ in the future"; $\mathsf{AF}\varphi$: "φ will occur in any future"; $\mathsf{EG}\varphi$: "it is possible to have φ holding permanently"; $\mathsf{AG}\varphi$: "φ will always hold"; $\mathsf{F}^{-1}\varphi$: "φ held at some time in the past"; $\varphi \mathsf{S} \psi$: "φ held at some time in the past, and ψ has been holding ever since".

2.3 N*, or "From now on"*

The N combinator was introduced in [16]. $\mathsf{N}\varphi$ reads "from now on, φ holds", or "starting anew from the current state, φ holds". Assume we want to state that any *crash* in the future is preceded by an earlier *mistake*. This can be written in *PCTL* as $\mathsf{AG}(\mathit{crash} \Rightarrow \mathsf{F}^{-1}\mathit{mistake})$.

Assume we now want to state that *after a proper reset is done*, any crash is preceded by an earlier mistake. Then $\mathsf{AG}[\mathit{reset} \Rightarrow \mathsf{AG}(\mathit{crash} \Rightarrow \mathsf{F}^{-1}\mathit{mistake})]$ will not do, because it allows the earlier mistake to occur before the reset is done ! This is a situation where we do not want to consider what happened before, and the right way to formally express our requirement is with $\mathsf{AG}[\mathit{reset} \Rightarrow \mathsf{NAG}(\mathit{crash} \Rightarrow \mathsf{F}^{-1}\mathit{mistake})]$ (see [16] for more details).

3 The difference between past and future

There exists several different ways to add past-time constructs to a pure-future temporal logic. Many proposals choose to view past and future as symmetric concepts. This gives rise to more uniform definitions. We choose to view Past and Future as having different properties. This view is motivated by considerations on what is the behavior of a non-deterministic reactive system, and what are the kind of properties we want to express about it.

The key points behind our choice are

1. **Past is determined.** We consider that, at any time along any computation, there is a completely fixed linear history of all events which already took place. This is in contrast with the branching view of Future where different possible continuations are considered.
2. **Past is finite.** A run of a system always has a starting point. This is in contrast with the usual view of Future where we do not require that all behaviors eventually terminate.
3. **Past is cumulative.** Whenever the system performs some steps and advances in time, its history becomes richer and longer. At termination (if ever), the past of the system is the whole computation.

We believe point **1.** is the most crucial. Logicians call it the *Ockhamist past* [31]. Some proposals (e.g. [24]) consider a non-determined past, also called "branching past", most typically through a clause like

$$q \models \mathsf{EX}^{-1} f \text{ iff there exists a } q' \, R \, q \text{ s.t. } q' \models f$$

(then making past potentially infinite.) We believe such a clause is often motivated by a concern for symmetry between past and future. Additionally, this allows the same efficient model-checking procedures. But such an "EX^{-1}" combinator is not very meaningful in terms of computations. It really expresses properties of a graph of states, and not of a behavioral tree. Indeed, the resulting logic is not compatible with bisimulation equivalence while our $PCTL$ is.

Point **2.** is less crucial because it is often possible (but clumsy) to write formulas in such a way that they only apply to behaviors having a definite starting point, much as we can express termination. However, we believe such a fundamental idea as "behaviors have a starting point" is better embedded into the semantic model. (Observe that "past is finite" is independent from the anchored notion of satisfaction.)

Point **3.** has its pros and cons (but the issue is only meaningful when past is determined). In [16], we explicitly asked whether we need a cumulative or a non-cumulative past when specifying reactive systems. Our answer was that most often a cumulative past is better suited, and we introduced the N combinator to deal with the few cases where a forgetful view of past is preferable. Observe that the combination of both views is only possible in a basic model with cumulative past.

Figure 2 classifies the different treatments of past in the literature. [14] is an important paper: it proposes extensions of CTL and of CTL^*, with a branching and with an Ockhamist past. Then it compares these extensions in term of expressive power, complexity, ... Basically, their Ockhamist past is like our proposal (from [16]) but without N. The paper does not give any indication of how its branching-past would be used for expressing natural behavioral properties of reactive systems, lending additional support to our views.

	Structure of past						
	Determined	Non-determ.	Finite	Infinite	Cumulative	Non-cumul.	
[8], [12], [18], [1], [32], [7], [28], [20], [23]	•		•		•		Linear time temporal logics
[24], [29], [26], [27]		•		•			non-Ockhamist past
[14]		•	•				
[31], [30]	•			•	•		Ockhamist past
[9], [2]	•		•			•	
[15], [14]	•		•		•		
[16]	•		•		•	•	

Fig. 2. The semantics of past in the literature

4 Specification of a lift system

We use the classical example of a lift system (from [1,10]) to experiment with the $PCTL$ logic. We want to see whether temporal specifications are clearer and closer to our intuitions when written in $PCTL$. This example has been chosen because it is rich and realistic but still easy to understand.

Our background hypothesis are:

- The lift services n floors numbered $1, \ldots, n$.
- There is a lift-door at each floor, with a call-button and an indicator light telling whether the cabin is called.
- In the cabin there are n send-buttons, one per floor, and n indicator lights.

4.1 Informal specification

The informal specification we have in mind gathers several properties we list (by order of importance) in Figure 3.

P1. Safe doors: A floor door is never open if the cabin is not present at the given floor.
P2. Indicator lights: The indicator lights correctly reflect the current requests.
P3. Service: All requests are eventually satisfied.
P4. Smart service: The cabin only services the requested floors and does not move when it has no request.
P5. Diligent service: The cabin does not pass by a floor for which it has a request without servicing it.
P6. Direct movements: The cabin always moves directly from previous to next serviced floor.
P7. Priorities: The cabin services in priority requests that do not imply a change of direction (upward or downward).

Fig. 3. An informal lift specification

P1-3 are sufficient to guarantee a correct and useful behavior (admittedly not too smart). The remaining properties can be seen as describing a notion of optimized behavior. Of course, this is still very informal and the whole point of the exercise is to now write all this down, using a formal logical language.

At any given time, some parameters of the system are observable. The specification will only refer to these parameters (and their evolution through time). We assume they are:

- a floor door is *open* or *closed*,
- a button is *pressed* or *depressed*,
- an indicator light is *on* or *off*,
- the cabin is *present* at floor i, or it is *absent*.

4.2 Atomic propositions

Formally, the assumption we made about the observable parameters just means that we consider a set $Prop$ of atomic propositions consisting of:

- $Open\_Door_i$ $(i = 1, \ldots, n)$, true if the floor door at floor i is open,
- $Call_i$ (resp. $Send_i$) $(i = 1, \ldots, n)$, true if the call-button at floor i (resp. send-button for i) is pressed,
- $Call\_Light_i$ (resp. $Send\_Light_i$) $(i = 1, \ldots, n)$, true if the indicator light for the i-th call- (resp. send-) button is on,
- At_i $(i = 1, \ldots, n)$, true if the cabin is at floor i.

4.3 The formal specification

4.3.1 P1. Safe doors

This leaves no room for interpretation : $\bigwedge_{i=1}^{n} \mathsf{AG}(Open\_Door_i \Rightarrow At_i)$ (S1)

4.3.2 P2. Indicator lights

This has to be interpreted. We choose to express that each time a button is pressed, there is a corresponding request that has to be memorized until fulfillment (if ever). A request for floor i is satisfied when the lift is servicing floor i, i.e. present at floor i with its door open. We introduce the corresponding abbreviation:

$$Servicing_i \stackrel{\text{def}}{=} At_i \wedge Open\_Door_i \quad (i = 1, \ldots, n) \tag{D1}$$

We decompose the intuition into several component. First, when a button is pressed, the corresponding indicator light is turned on:

$$\bigwedge_{i=1}^{n} \mathsf{AG}[Call_i \Rightarrow (Servicing_i \vee Call\_Light_i)] \tag{S2.1}$$

$$\bigwedge_{i=1}^{n} \mathsf{AG}[Send_i \Rightarrow (Servicing_i \vee Send\_Light_i)] \tag{S2.2}$$

Then, lights on stay on until the corresponding request is fulfilled (if ever). For this we use W, the “weak until” (also “unless”), defined by

$$\mathsf{E}\varphi\mathsf{W}\psi \stackrel{\text{def}}{=} \neg\mathsf{A}(\neg\psi)\mathsf{U}(\neg\varphi \wedge \neg\psi) \qquad \mathsf{A}\varphi\mathsf{W}\psi \stackrel{\text{def}}{=} \neg\mathsf{E}(\neg\psi)\mathsf{U}(\neg\varphi \wedge \neg\psi)$$

and write

$$\bigwedge_{i=1}^{n} \mathsf{AG}[Call\_Light_i \Rightarrow \mathsf{A}\,Call\_Light_i \mathsf{W} Servicing_i] \tag{S2.3}$$

$$\bigwedge_{i=1}^{n} \mathsf{AG}[Send\_Light_i \Rightarrow \mathsf{A}\,Send\_Light_i \mathsf{W} Servicing_i] \tag{S2.4}$$

Then, lights are turned off when the request is fulfilled:

$$\bigwedge_{i=1}^{n} \mathsf{AG}[\mathit{Servicing}_i \;\Rightarrow\; (\neg \mathit{Call\_Light}_i \wedge \neg \mathit{Send\_Light}_i)] \tag{S2.5}$$

There only remains to state that the lights are only turned on when necessary. For this, we can write that, whenever a light is on, then a corresponding request has been made *before*. However, something like $\mathsf{AG}(\mathit{Call\_Light}_i \;\Rightarrow\; \mathsf{F}^{-1}\mathit{Call}_i)$ does not work because it allows one early call to account for all future turnings on of the indicator light. Rather, we mean

$$\bigwedge_{i=1}^{n} \mathsf{AG}[\mathit{Call\_Light}_i \;\Rightarrow\; (\mathit{Call\_Light}_i \mathsf{S} \mathit{Call}_i)] \tag{S2.6}$$

$$\bigwedge_{i=1}^{n} \mathsf{AG}[\mathit{Send\_Light}_i \;\Rightarrow\; (\mathit{Send\_Light}_i \mathsf{S} \mathit{Send}_i)] \tag{S2.7}$$

An alternative possibility would have been to use N combinator, suited to this kind of situation, and state:

$$\bigwedge_{i=1}^{n} \mathsf{AG}[\neg \mathit{Call\_Light}_i \;\Rightarrow\; \mathsf{NAG}(\mathit{Call\_Light}_i \;\Rightarrow\; \mathsf{F}^{-1}\mathit{Call}_i)] \tag{S2.6'}$$

$$\bigwedge_{i=1}^{n} \mathsf{AG}[\neg \mathit{Send\_Light}_i \;\Rightarrow\; \mathsf{NAG}(\mathit{Send\_Light}_i \;\Rightarrow\; \mathsf{F}^{-1}\mathit{Send}_i)] \tag{S2.7'}$$

(Observe that (S2.6-7) and (S2.6'-7') are not equivalent when considered in isolation.)

We could choose to summarize all this stating "*an indicator light is on* **iff** *there exists a (corresponding) pending request*".

$$\bigwedge_{i=1}^{n} \mathsf{AG}\left[\begin{array}{l} \big(\mathit{Call\_Light}_i \;\Leftrightarrow\; \neg \mathit{Servicing}_i \,\mathsf{S}\, (\mathit{Call}_i \wedge \neg \mathit{Servicing}_i)\big) \\ \wedge\big(\mathit{Send\_Light}_i \;\Leftrightarrow\; \neg \mathit{Servicing}_i \,\mathsf{S}\, (\mathit{Send}_i \wedge \neg \mathit{Servicing}_i)\big) \end{array}\right] \tag{S2'}$$

4.3.3 P3. Service

We choose the more logical approach and express this in terms of pressed buttons, rather than indicator lights.

$$\bigwedge_{i=1}^{n} \mathsf{AG}[\mathit{Request}_i \;\Rightarrow\; \mathsf{AF}\mathit{Servicing}_i] \tag{S3}$$

where $$\mathit{Request}_i \stackrel{\text{def}}{=} \mathit{Call}_i \vee \mathit{Send}_i \quad (i = 1, \ldots, n) \tag{D2}$$

4.3.4 P4. Smart service

This is better stated in terms of indicator lights. We introduce the abbreviations

$$\mathit{Pending\_Request}_i \stackrel{\text{def}}{=} \mathit{Call\_Light}_i \vee \mathit{Send\_Light}_i \quad (i = 1, \ldots, n) \tag{D3}$$

$$\mathit{Some\_Pending\_Request} \stackrel{\text{def}}{=} \bigvee_{i=1}^{n} \mathit{Pending\_Request}_i \tag{D4}$$

and can now write that a floor is only serviced if there is a pending request for it

$$\bigwedge_{i=1}^{n} \mathsf{AG}\left[Servicing_i \;\Rightarrow\; (Servicing_i \,\mathsf{S}\, Pending\_Request_i) \right] \tag{S4.1}$$

and that the cabin is motionless unless there is some request

$$\bigwedge_{i=1}^{n} \mathsf{AG}\left(At_i \;\Rightarrow\; \left[\mathsf{A}\, At_i \,\mathsf{W}\, Some\_Pending\_Request\right] \right) \tag{S4.2}$$

Observe that the cabin needs not always be at some floor. We complete (S4.2) with

$$\mathsf{AG}\left(Between\_Floors \;\Rightarrow\; \left[\mathsf{A}\, Between\_Floors \,\mathsf{W}\, Some\_Pending\_Request\right] \right) \tag{S4.3}$$

where

$$Between\_Floors \stackrel{\text{def}}{=} \bigwedge_{i=1}^{n} \neg At_i \tag{D5}$$

4.3.5 P5. Diligent service

We formalize "diligent service" as forbidding situations where

(i) the cabin was servicing some floor i,
(ii) then it moved and went to service some other floor j,
(iii) therefore passing by some intermediary floor k,
(iv) but *this ignored a pending request* for k.

This is a complex behavioral notion. We need to express a notion of "passing by a given floor" while we have no observable parameter telling us whether the cabin is moving or not, whether it is moving up or down, ... Furthermore, we have to choose between two possible interpretations of "ignoring a pending request for k": (i) the request already exists when the cabin starts moving, or (ii) the request exists when the cabin actually is at floor k.

The second interpretation is easy to specify with

$$Not\_Servicing \stackrel{\text{def}}{=} \bigwedge_{i=1}^{n} \neg Servicing_i \tag{D6}$$

$$\bigwedge_{k=1}^{n} \mathsf{AG}\Big[\left(At_k \wedge Pending\_Request_k \right) \;\Rightarrow\; \mathsf{A}\; Not\_Servicing \,\mathsf{W}\, Servicing_k \Big] \tag{S5'}$$

but we prefer the first interpretation which we see as more realistic. It requires to refer to the moment where we leave the previously serviced floor. We shall use the following abbreviations:

$$At_j From_i \stackrel{\text{def}}{=} Servicing_j \;\wedge\; (Servicing_j \vee Not\_Servicing) \,\mathsf{S}\, Servicing_i \qquad (i, j = 1, \ldots, n) \tag{D7}$$

and write $(i,j) \stackrel{\text{def}}{=} \{k \mid i < k < j \text{ or } j < k < i\}$ for the set of intermediary floors between i and j. Now "diligent servicing" can be stated

$$\bigwedge_{i=1}^{n} \bigwedge_{\substack{j=1 \\ j \neq i}}^{n} \mathsf{AG}\ At_j From_i \Rightarrow \begin{bmatrix} \big(Servicing_j \vee Not\_Servicing\big) \\ \quad \mathsf{S}\big(Servicing_i \wedge \bigwedge_{k \in (i,j)} \neg Request_k\big) \end{bmatrix} \tag{S5}$$

4.3.6 P6. Direct movements

We understand this property in terms of positions "At_i" rather than in terms of services "$Servicing_i$". Basically, we require that whenever the cabin is at some time at floor i, later at floor j, and finally at floor k, then (1) j lies between i and k, or (2) this is because the lift went to *service* a floor not between i and k.

This is easily stated if we use the N combinator to mark the moment where the cabin is "initially" at i.

$$\bigwedge_{i,k=1}^{n} \bigwedge_{j \notin (i,k)} \mathsf{AG}\Big[\mathsf{N} At_i \ \Rightarrow\ \mathsf{AG}\Big(At_k \wedge \mathsf{F}^{-1} At_j \ \Rightarrow\ \mathsf{F}^{-1} \bigvee_{l \notin (i,k)} Servicing_l\Big)\Big] \tag{S6}$$

4.3.7 P7. Priorities

We need to express when the cabin is going upward (resp. downward). Intuitively, the cabin is *going up* (resp. *down*) at all times between a (strictly) earlier moment when it is at floor $i-1$ (resp. $i+1$) and a later moment when it is at floor i.

$$Up \stackrel{\text{def}}{=} \bigvee_{i=2}^{n} \Big[\Big(\big(At_i \vee Between\_Floors\big)\mathsf{S} At_{i-1}\Big) \ \wedge\ \mathsf{A} Between\_Floors\, \mathsf{U} At_i\Big] \tag{D8}$$

$$Down \stackrel{\text{def}}{=} \bigvee_{i=1}^{n-1} \Big[\Big(\big(At_i \vee Between\_Floors\big)\mathsf{S} At_{i+1}\Big) \ \wedge\ \mathsf{A} Between\_Floors\, \mathsf{U} At_i\Big] \tag{D9}$$

Now, we can state that if the cabin services some floor i, and is coming from a higher floor (i.e. is *going down*), and there exists a request for a lower floor j, then the next serviced floor will not be a higher floor k. We also require a similar property when the cabin is going up.

$$\mathsf{AG} \bigwedge_{i=1}^{n} \begin{bmatrix} \Big(Servicing_i \wedge Down \wedge \big(\bigvee_{j<i} Pending\_Request_j\big)\Big) \\ \quad \Rightarrow\ \neg\ \mathsf{E}\ \big(Servicing_i \vee Not\_Servicing\big)\ \mathsf{U}\ \big(\bigvee_{k>i} Servicing_k\big) \end{bmatrix} \tag{S7.1}$$

$$\mathsf{AG} \bigwedge_{i=1}^{n} \begin{bmatrix} \Big(Servicing_i \wedge Up \wedge \big(\bigvee_{j>i} Pending\_Request_j\big)\Big) \\ \quad \Rightarrow\ \neg\ \mathsf{E}\ \big(Servicing_i \vee Not\_Servicing\big)\ \mathsf{U}\ \big(\bigvee_{k<i} Servicing_k\big) \end{bmatrix} \tag{S7.2}$$

4.4 Some lessons to be drawn

We do not claim our informal specification from Fig. 3 reflects the reality of lift-designing. We just wanted to have a collection of easy-to-understand behavioral properties and see how we could express them in $CTL+Past$. Observe that roughly one half of the specification uses the past-time constructs. Thus our example is one more proof of the usefulness of these constructs.

Many other properties could have been considered, many variant formalizations could have been offered. Still we think the following conclusions have some general truth in them:

- It is indeed quite possible to express interesting temporal properties in a propositional temporal logic like $CTL + Past$,
- Without accompanying explanations, the resulting formulas are hard to read and can probably not be used as a documentation aid. But they can be used for verification purposes when model-checking is possible.
- They are not so hard to write, when one just sees them as a rather direct encoding of sentences spelled out in English.
- Allowing past-time constructs is convenient. It makes the specification easier to write, and easier to read.

5 Verification with past constructs

We just saw how extending CTL with some well-chosen past-time constructs equipped with the right semantics allows writing simpler and much more natural specifications.

Now, CTL is paradigmatic in the field because it allowed the development of very efficient model-checking tools that can successfully handle very large systems [3]. Thus a very important question is to know how our proposal for an extended CTL allows efficient model-checking. Indeed, other proposed extensions to CTL (typically CTL^* and the full branching-time mu-calculus) were not so successful because they lacked efficient model-checking algorithms.

We advocate a *translation-based approach* for extensions of CTL [16,17]. That is, we argue that, when possible, the most convenient way to handle extensions of CTL is to translate them back into equivalent CTL formulas, so that the finely-tuned technology of CTL model-checkers can be reused without modification. An other advantage is that the translation can be implemented once, independently of the actual model-checking tool that is used afterward.

Now the problem is to find interesting extensions for which translations exist. In [16] we showed how $CTL + \mathsf{F}^{-1} + \mathsf{N}$ could be translated into CTL. Other extensions of CTL for enhanced practical expressivity have been proposed (e.g. CTL^+ in [6] or CTL^2 in [13]) but these works did not argue for a translation-approach to model-cheking.

In the next section, we demonstrate a translation for a fragment of $PCTL$ in which our lift example can be written. We first need to define what we mean by a *correct translation*. Recall that we are interested in specification for reactive systems *starting from an initial state*. Given a specification φ using past-time constructs, we need to translate it into some φ' with only future-time constructs with the following correctness criterion:

$$\text{for any FKS } S,\ S \models \varphi \text{ iff } S \models \varphi' \qquad \text{(CC)}$$

This naturally leads us to distinguish two notions of equivalence between formulas:

Definition 5.1 (i) Two formulas f and g are *equivalent*, written $f \equiv g$, when for all histories σ in all fair Kripke structures, $\sigma \models f \quad \Leftrightarrow \quad \sigma \models g$.

(ii) Two formulas f and g are *initially equivalent*, written $f \equiv_i g$, when for all states q in all fair Kripke structures, $q \models f \quad \Leftrightarrow \quad q \models g$.

Initial equivalence, $\equiv_i$, is the equivalence we need for our translation. We have (CC) iff $\varphi \equiv_i \varphi'$. The main difficulty is that $\equiv_i$ is not substitutive: $\varphi \equiv_i \varphi'$ does not entail $\psi[\varphi] \equiv_i \psi[\varphi']$ if $\psi[.]$ is a context involving past-time constructs. That is why we also use $\equiv$, the classical equivalence for formulas, which is fully substitutive. It is stronger than initial equivalence: $f \equiv g$ entails $f \equiv_i g$ but the converse is not true, e.g. $\mathsf{X}^{-1}\top \equiv_i \bot$ (because $\mathsf{X}^{-1}\top$ doesn't hold for a starting point) but of course $\mathsf{X}^{-1}\top \not\equiv \bot$. Note that N helps understand the links between the two notions of equivalences:

$$\varphi \equiv_i \psi \text{ iff } \mathsf{N}\varphi \equiv \mathsf{N}\psi$$

Now, we can define the translation of a logic L_1 into a logic L_2:

Definition 5.2 L_1 can be (initially) translated into L_2, if for any $f_1 \in L_1$ there is a $f_2 \in L_2$ s.t. $f_1 \equiv_i f_2$.

(Of course, this is only interesting in practice if there exists an effective method for the translation.)

Section 6 studies the possibilities of translating specifications with past combinators into "pure future" specification (written in CTL).

6 A translation-based approach to model-checking $CTL+Past$

We would like to translate $PCTL$ into CTL. Unfortunately this is impossible:

Theorem 6.1 **[16]**

(i) *$CTL + \mathsf{S}$ cannot be translated into CTL.*

(ii) *$CTL + \mathsf{X}^{-1}$ cannot be translated into CTL.*

These two results are based on the following observations: (1) the formula $\mathsf{EG}(a \vee \mathsf{X}^{-1}a \vee \neg\mathsf{X}^{-1}\top)$ cannot be expressed in CTL, and (2) it is possible, by using embedded S combinators, to build a $CTL + \mathsf{S}$ formula equivalent to the

CTL^* formula $\mathsf{E}(c \vee a\mathsf{U}b)\mathsf{U}d$ which cannot be expressed in CTL.

In view of these impossibility results, one has to look for a fragment of $PCTL$ that can be translated into CTL. Indeed, we know that

Theorem 6.2 **[16]** *$CTL + \mathsf{F}^{-1} + \mathsf{N}$ can be translated into CTL.*

This result only partly helps us because our LIFT specification from section 4 was not written in the $CTL + \mathsf{F}^{-1} + \mathsf{N}$ fragment. (Additionally, [16] did not take fairness into account.)

The main theoretical result of this paper is the observation that, even if the introduction of S into CTL can push it far beyond CTL expressivity, there exists a precisely delineated fragment of $PCTL$ that (1) support the LIFT specification, and (2) can be translated into CTL. For example, notwithstanding its occurrences of S, formula (S2.6) is initially equivalent to a CTL formula:

$$\bigwedge_{i=1}^{n} \mathsf{AG}[(\neg Call\_Light_i \wedge \neg Call_i) \;\;\Rightarrow\;\; \neg\mathsf{E}(\neg Call_i)\mathsf{U}(Call\_Light_i \wedge \neg Call_i)]$$

Informally instead of specifying "when a light is on, the corresponding button has been pressed", we say "when a light is off, it will not turn on unless the button is pressed".

We now define $\mathbf{N}CTL$, the aforementioned fragment of $PCTL$:

Definition 6.3 The logic $\mathbf{N}CTL$

$$\mathbf{N}CTL \ni \varphi, \psi \;::=\; \lambda \mid \varphi \wedge \psi \mid \neg\varphi \mid \mathsf{EX}\varphi \mid \mathsf{E}\lambda\mathsf{U}\varphi \mid \lambda\mathsf{S}\mu \mid \mathsf{X}^{-1}\varphi$$

$$\lambda, \mu \;::=\; a \mid \lambda \wedge \mu \mid \neg\lambda \mid \mathsf{EX}\lambda \mid \mathsf{E}\lambda\mathsf{U}\mu \mid \mathsf{A}\lambda\mathsf{U}\mu \mid \mathsf{F}^{-1}\lambda \mid \mathsf{N}\varphi$$

Thus $\mathbf{N}CTL$ forbids occurrences of S and X^{-1} in the scope of S or $\mathsf{A\_U}$ (except if a N is in between) and in the left-hand side of $\mathsf{E\_U}$. In such contexts, only *limited* formulas λ and μ are allowed. Note that F^{-1} can be used without restriction.

Remark 6.4 Every formula used in the LIFT specification of section 4 belongs to $\mathbf{N}CTL$.

Now we have the following result:

Theorem 6.5 *$\mathbf{N}CTL$ can be (effectively) translated into CTL*

This is the main theorem. In the rest of this section, we only give the plan of its proof, relegating details into the appendix.

We say that a $PCTL$ formula is *separated* when no past combinator occurs in the scope of a future combinator. This definition, more general than

Gabbay's stricter notion [7], is what we really need. Theorem 6.5 is based on the following separation lemma:

Lemma 6.6 (Separation lemma) *Any* $\mathbf{N}CTL$ *formula is equivalent to a separated* $\mathbf{N}CTL$ *formula.*

Proof. See the appendix. □

Now the final step only requires transforming a separated formula into an initially equivalent CTL formula (this can be done easily, see the appendix). For example, we have:

$$\mathsf{E}\, a\, \mathsf{U}\, (b \wedge c \mathsf{S} d) \qquad \text{a } \mathbf{N}CTL \text{ formula}$$

$$\equiv \left.\begin{array}{l} \quad c\mathsf{S}d \;\wedge\; \mathsf{E}\,(a \wedge c)\,\mathsf{U}\,(b \wedge c) \\ \vee\; \mathsf{E}\, a\, \mathsf{U} \left(a \wedge d \wedge \mathsf{EX}\, \mathsf{E}(a \wedge c)\mathsf{U}(b \wedge c) \right) \\ \vee\; \mathsf{E}\, a\, \mathsf{U}\, (b \wedge d) \end{array}\right\} \text{a separated } \mathbf{N}CTL \text{ formula}$$

$$\equiv_i \left.\begin{array}{l} \quad \mathsf{E}\, a\, \mathsf{U} \left(a \wedge d \wedge \mathsf{EX}\, \mathsf{E}(a \wedge c)\mathsf{U}(b \wedge c) \right) \\ \vee\; \mathsf{E}\, a\, \mathsf{U}\, (b \wedge d) \end{array}\right\} \text{a } CTL \text{ formula}$$

A consequence of Theorem 6.5 is that all formulas used in the LIFT specification can be automatically translated into (initially) equivalent CTL formulas for the verification step: the specification is easier to write (and to rectify) and a model of a lift system (given as some FKS) can be verified with a standard model-checker by confronting it to the CTL translation of the specification.

Remark 6.7 Theorem 6.5 can be extended to a larger $\mathbf{N}CTL^+$ where boolean combinations of path-formulas are allowed under a path quantifier, and to an even larger $\mathbf{N}ECTL^+$, this time translating it into $ECTL^+$.

Conclusion

In this paper, we explained and motivated what is, in our opinion, the best semantical framework for temporal logics with past-time when it comes to specifying and verifying reactive systems. Today, this so-called *Ockhamist framework with finite and cumulative past* is not the most commonly used for branching-time logics, in part because the question of which semantical framework is best has not yet been much discussed.

We demonstrated the advantages of this approach by writing a specification for the classical lift system example in $PCTL$. Following our earlier translation-based approach, we showed that this $PCTL$ specification can be used effectively for model-checking purposes if one translates it into an equivalent CTL specification. This can be done thanks to a new translation theorem, extending to $\mathbf{N}CTL$ our earlier work on $CTL + \mathsf{F}^{-1}$.

An important question is the complexity of the translation : From a theoretical viewpoint, our translation algorithm may induce combinatorial explosions, even with limited temporal height [16]. As far as we know, informative lower bounds on the problem (rather than about a given translation algorithm) are not known, even in the linear-time fragment of [7]. From a practical viewpoint, what remains to be done is to implement Theorem 6.5 and see whether actual $\mathbf{N}CTL$ specifications can be translated in practice.

Directions for future work should be motivated by actual applications. Thus our plans for the near-future are to implement the translation algorithm we propose and to plug it on top of SMV and other model-checkers accepting CTL (with or without fairness). We expect this will naturally suggest ideas for improved rewriting strategy (and rules) and for enlarged logics.

References

[1] H. Barringer. Up and down the temporal way. *The Computer Journal*, 30(2):134–148, April 1987.

[2] A. Bouajjani, Y. Lakhnech, and S. Yovine. Model checking for extended timed temporal logics. In *Proc. 4th Int. Symp. Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'96), Uppsala, Sweden, Sep. 1996*, volume 1135 of *Lecture Notes in Computer Science*, pages 306–326. Springer-Verlag, 1996.

[3] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[4] E. Clarke, O. Grümberg, and D. Long. Verification tools for finite-state concurrent systems. In *A Decade of Concurrency, Proc. REX School/Symp., Noordwijkerhout, NL, June 1993*, volume 803 of *Lecture Notes in Computer Science*, pages 124–175. Springer-Verlag, 1994.

[5] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, vol. B*, chapter 16, pages 995–1072. Elsevier Science Publishers, 1990.

[6] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30(1):1–24, 1985.

[7] D. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In *Proc. Workshop Temporal Logic in Specification, Altrincham, UK, Apr. 1987*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448. Springer-Verlag, 1989.

[8] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proc. 7th ACM Symp. Principles of Programming Languages (POPL'80), Las Vegas, Nevada, Jan. 1980*, pages 163–173, 1980.

[9] T. Hafer and W. Thomas. Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In *Proc. 14th Int. Coll. Automata, Languages, and Programming (ICALP'87), Karlsruhe, FRG, July 1987*, volume 267 of *Lecture Notes in Computer Science*, pages 269–279. Springer-Verlag, 1987.

[10] R. Hale. Using temporal logic for prototyping: the design of a lift controller. In *Proc. Workshop Temporal Logic in Specification, Altrincham, UK, Apr. 1987*, volume 398 of *Lecture Notes in Computer Science*, pages 375–408. Springer-Verlag, 1989.

[11] Y. Kesten, Z. Manna, H. McGuire, and A. Pnueli. A decision algorithm for full propositional temporal logic. In *Proc. 5th Int. Conf. Computer-Aided Verification (CAV'93), Elounda, Greece, June 1993*, volume 697 of *Lecture Notes in Computer Science*, pages 97–109. Springer-Verlag, 1993.

[12] R. Koymans, J. Vytopil, and W. P. de Roever. Real-time programming and asynchronous message passing. In *Proc. 2nd ACM Symp. Principles of Distributed Computing (PODC'83), Montreal, Canada, Aug. 1983*, pages 187–197, 1983.

[13] O. Kupferman and O. Grümberg. Buy one, get one free!!! *J. Logic and Computation*, 6(4):523–539, August 1996.

[14] O. Kupferman and A. Pnueli. Once and for all. In *Proc. 10th IEEE Symp. Logic in Computer Science (LICS'95), San Diego, CA, June 1995*, pages 25–35, 1995.

[15] F. Laroussinie, S. Pinchinat, and Ph. Schnoebelen. Translations between modal logics of reactive systems. *Theoretical Computer Science*, 140(1):53–71, 1995.

[16] F. Laroussinie and Ph. Schnoebelen. A hierarchy of temporal logics with past. *Theoretical Computer Science*, 148(2):303–324, 1995.

[17] F. Laroussinie and Ph. Schnoebelen. Translations for model-checking temporal logic with past. In *Actes des 2e Journées Toulousaines sur la Formalisation des Activités Concurrentes (FAC'96), Toulouse, France, Feb. 1996*, pages 17–19, 1996.

[18] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Proc. Logics of Programs Workshop, Brooklyn, NY, June 1985*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer-Verlag, 1985.

[19] Z. Manna and A. Pnueli. The anchored version of the temporal framework. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Proc. REX School/Workshop, Noordwijkerhout, NL, May-June 1988*, volume 354 of *Lecture Notes in Computer Science*, pages 201–284. Springer-Verlag, 1989.

[20] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*, volume I: Specification. Springer-Verlag, 1992.

[21] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[22] K.L. McMillan. The SMV system, symbolic model checking - an approach. Technical Report CMU-CS-92-131, Carnegie Mellon University, 1992.

[23] L. E. Moser, P. M. Melliar-Smith, G. Kutty, and Y. S. Ramakrishna. Completeness and soundness of axiomatizations for temporal logics without next. *Fundamenta Informaticae*, 21(4):257–305, October 1994.

[24] S. S. Pinter and P. Wolper. A temporal logic for reasoning about partially ordered computations (extended abstract). In *Proc. 3rd ACM Symp. Principles of Distributed Computing, (PODC'84), Vancouver, B.C., Canada, Aug. 1984*, pages 28–37, 1984.

[25] Y. S. Ramakrishna, L. E. Moser, L. K. Dillon, P. M. Melliar-Smith, and G. Kutty. An automata-theoretic decision procedure for propositional temporal logic with Since and Until. *Fundamenta Informaticae*, 17(3):271–282, November 1992.

[26] C. Stirling. Comparing linear and branching time temporal logics. In *Proc. Workshop Temporal Logic in Specification, Altrincham, UK, Apr. 1987*, volume 398 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 1989.

[27] C. Stirling. Modal and temporal logics. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, pages 477–563. Oxford Univ. Press, 1992.

[28] M. Y. Vardi. A temporal fixpoint calculus. In *Proc. 15th ACM Symp. Principles of Programming Languages (POPL'88), San Diego, CA, Jan. 1988*, pages 250–259, 1988.

[29] P. Wolper. On the relation of programs and computations to models of temporal logic. In *Proc. Workshop Temporal Logic in Specification, Altrincham, UK, Apr. 1987*, volume 398 of *Lecture Notes in Computer Science*, pages 75–123. Springer-Verlag, 1989.

[30] A. Zanardo. Branching-time logic with quantification over branches: The point of view of modal logic. *J. Symbolic Logic*, 61(1):1–39, March 1996.

[31] A. Zanardo and J. Carmo. Ockhamist computational logic: Past-sensitive necessitation in CTL*. *J. Logic and Computation*, 3(3):249–268, 1993.

[32] L. Zuck. *Past Temporal Logic.* PhD thesis, Weizmann Institute, Rehovot, Israel, August 1986.

A Appendix: Proof of the Separation Lemma for NCTL

Recall that a separated formula is a formula in which no past-time construct occurs in the scope of future combinators.

We follow the steps of our earlier proof for the separation $CTL + \mathsf{F}^{-1} + \mathsf{N}$ in [16]: we offer a collection of rewriting rules to extract occurrences of the past combinators S, F^{-1} and X^{-1} from the scope of future combinators. The crucial point is to find a strategy for the application of the rules that ensures termination.

Our set of rules is split into two parts: those needed to extract the S's and X^{-1}'s are given in Figure A.1 and those needed to extract the F^{-1}'s are given in Figure A.2.

$$(R1)\ \mathsf{E}\Big(\varphi\mathsf{U}(\alpha \wedge \mathsf{X}^{-1}x)\Big) \equiv \alpha \wedge \mathsf{X}^{-1}x \vee \mathsf{E}\Big(\varphi\mathsf{U}(\varphi \wedge x \wedge \mathsf{EX}\alpha)\Big)$$

$$(R2)\ \mathsf{E}\Big(\varphi\mathsf{U}(\alpha \wedge \neg\mathsf{X}^{-1}x)\Big) \equiv \alpha \wedge \neg\mathsf{X}^{-1}x \vee\ \mathsf{E}\Big(\varphi\mathsf{U}(\varphi \wedge \neg x \wedge \mathsf{EX}\alpha)\Big)$$

$$(R3)\ \mathsf{EX}(\alpha \wedge \mathsf{X}^{-1}x) \equiv x \wedge \mathsf{EX}\alpha$$

$$(R4)\ \mathsf{EX}(\alpha \wedge \neg\mathsf{X}^{-1}x) \equiv \neg x \wedge \mathsf{EX}\alpha$$

$$\begin{aligned}(R5)\ & \mathsf{E}\Big(\varphi\mathsf{U}(\alpha \wedge x\mathsf{S}y)\Big) \\ & \equiv \mathsf{E}\varphi\mathsf{U}(\alpha \wedge y) \vee \mathsf{E}\varphi\mathsf{U}\Big(\varphi \wedge y \wedge \mathsf{EXE}(\varphi \wedge x)\mathsf{U}(\alpha \wedge x)\Big) \\ & \quad \vee x\mathsf{S}y \wedge \Big(\alpha \vee \mathsf{E}(\varphi \wedge x)\mathsf{U}(\alpha \wedge x)\Big)\end{aligned}$$

$$\begin{aligned}(R6)\ & \mathsf{E}\Big(\varphi\mathsf{U}(\alpha \wedge \neg(x\mathsf{S}y))\Big) \\ & \equiv \mathsf{E}\varphi\mathsf{U}(\alpha \wedge \neg x \wedge \neg y) \vee \mathsf{E}\varphi\mathsf{U}\Big(\varphi \wedge \neg x \wedge \neg y \wedge \mathsf{E}(\varphi \wedge \neg y)\mathsf{U}(\alpha \wedge \neg y)\Big) \\ & \quad \vee \neg(x\mathsf{S}y) \wedge \Big(\alpha \vee \mathsf{E}(\varphi \wedge \neg y)\mathsf{U}(\alpha \wedge \neg y)\Big)\end{aligned}$$

$$(R7)\ \mathsf{EX}(\alpha \wedge x\mathsf{S}y) \equiv \mathsf{EX}(\alpha \wedge y) \vee x\mathsf{S}y \wedge \mathsf{EX}(\alpha \wedge x)$$

$$(R8)\ \mathsf{EX}(\alpha \wedge \neg(x\mathsf{S}y)) \equiv \mathsf{EX}(\alpha \wedge \neg x \wedge \neg y) \vee\ \neg(x\mathsf{S}y) \wedge \mathsf{EX}(\alpha \wedge \neg y)$$

Fig. A.1. Rules to extract S and X^{-1} from the scope of future combinators.

$$\begin{aligned}(R9)\ \ & \mathsf{E}\Big((\mathsf{F}^{-1}x \wedge \alpha) \vee (\neg\mathsf{F}^{-1}x \wedge \beta) \vee \gamma\Big)\mathsf{U}\Big((\mathsf{F}^{-1}x \wedge \alpha') \vee (\neg\mathsf{F}^{-1}x \wedge \beta') \vee \gamma'\Big) \\ & \equiv \mathsf{F}^{-1}x\ \wedge\ \mathsf{E}(\alpha \vee \gamma)\mathsf{U}(\alpha' \vee \gamma') \\ & \quad \vee \neg\mathsf{F}^{-1}x\ \wedge\ \mathsf{E}\Big(\neg x \wedge (\beta \vee \gamma)\Big)\mathsf{U}\Big(x \wedge \mathsf{E}(\alpha \vee \gamma)\mathsf{U}(\alpha' \vee \gamma')\Big) \\ & \quad \vee \neg\mathsf{F}^{-1}x\ \wedge\ \mathsf{E}\Big(\neg x \wedge (\beta \vee \gamma)\Big)\mathsf{U}\Big(\neg x \wedge (\beta' \vee \gamma')\Big)\end{aligned}$$

$$\begin{aligned}(R10)\ & \mathsf{EG}\Big((\mathsf{F}^{-1}x \wedge \alpha) \vee (\neg\mathsf{F}^{-1}x \wedge \beta) \vee \gamma\Big) \\ & \equiv \mathsf{F}^{-1}x\ \wedge\ \mathsf{EG}(\alpha \vee \gamma) \\ & \quad \vee \neg\mathsf{F}^{-1}x\ \wedge\ \mathsf{E}\Big(\neg x \wedge (\beta \vee \gamma)\Big)\mathsf{U}\Big(x \wedge \mathsf{EG}(\alpha \vee \gamma)\Big) \\ & \quad \vee \neg\mathsf{F}^{-1}x\ \wedge\ \mathsf{EG}\Big(\neg x \wedge (\beta \vee \gamma)\Big)\end{aligned}$$

$$(R11)\ \mathsf{EX}(\alpha \wedge \mathsf{F}^{-1}x)\ \equiv\ \mathsf{EX}(\alpha \wedge x)\ \vee\ \mathsf{F}^{-1}x \wedge \mathsf{EX}\alpha$$

$$(R12)\ \mathsf{EX}(\alpha \wedge \neg\mathsf{F}^{-1}x)\ \equiv\ \neg\mathsf{F}^{-1}x \wedge \mathsf{EX}(\alpha \wedge \neg x)$$

Fig. A.2. Rules to extract F^{-1} from the scope of future combinators.

A.1 Soundness of the rules

Lemma A.1 (Soundness) *All rules in figures A.1 and A.2 are correct for FKS's, i.e. the equivalences hold for any PCTL formulas φ, x, y, α, β, γ, α', β' and γ'.*

The complete proof of Lemma A.1 is a tedious verification left to the reader. The general approach is always the same and it can be illustrated with the

$(R5)$ rule: assume $\sigma \models \mathsf{E}\,\varphi\,\mathsf{U}(\alpha \wedge x\mathsf{S}y)$. Then σ can be extended into some π s.t. in particular $\pi \models \alpha \wedge x\mathsf{S}y$. Now we distinguish three cases depending on when y is satisfied: (1) at the last moment, together with α, or (2) after σ but strictly before α holds, or (3) in the past of σ. Each case yields one term in the disjunction.

The conditions over fairness constraints (Def. 2.2) are required for Lemma A.1. They let us decompose any execution into several parts and concatenate an arbitrary prefix with a fair suffix.

A.2 Stability for $\mathbf{N}CTL$

Lemma A.2 *Rewriting any* $\mathbf{N}CTL$ *formula* φ *(resp. limited formula* λ*) by applying one of our 12 rules to a subformula always yields* φ' *in* $\mathbf{N}CTL$ *(resp. a limited formula* λ'*).*

This is because, when given $\mathbf{N}CTL$ formulas, the rules never move a S or X^{-1} in the left-hand part of some $\mathsf{E\_U}$ or inside an $\mathsf{A\_U}$ or S context.

Note that, in addition, all the usual boolean manipulation rules one uses (distributivity, disjunctive normalization, ...) are stable for $\mathbf{N}CTL$.

A.3 Separation strategy for $\mathbf{N}CTL$

The 12 rewrite rules we gave allows to extract any single occurrence of a past-time combinator from the scope of one future-time combinator. However, it is not clear that a blind application of them will always eventually separate past from future. E.g. consider (R10): using it extracts $\mathsf{F}^{-1}x$ from the scope of EG, but at the same time this (1) duplicates γ, and (2) one buries one occurrence of γ under two embedded future-time combinators. Clearly, if γ contains past-time constructs, eventual separation is not guaranteed.

We now show how a precise strategy ensures eventual separation. Our strategy clearly shows how the rules are used and why termination is ensured. We present it in a hierachical way, handling larger and larger fragments of $\mathbf{N}CTL$. We heavily use *contexts*, i.e. formulas with variables in them. The x in $f[x]$ can be replaced by any formula: we write $f[g]$ for f with g in place of x. **Note that x in $f[x]$ may stand for several occurrences of x. This is a key point in our method, used to collect copies of duplicated subformulas.**

Lemma A.3 *Let* $f[x]$ *be a* CTL *context. If* $f[x_1\mathsf{S}x_2]$ *(resp.* $f[\mathsf{X}^{-1}x_1]$, $f[\mathsf{F}^{-1}x_1]$*) is a* $\mathbf{N}CTL$ *context, then* $f[x_1\mathsf{S}x_2]$ *(resp.* $f[\mathsf{X}^{-1}x_1]$, $f[\mathsf{F}^{-1}x_1]$*) is equivalent to a separated* $\mathbf{N}CTL$ *context* $f'[x_1, x_2, x_1\mathsf{S}x_2]$ *(resp.* $f'[x_1, \mathsf{X}^{-1}x_1]$, $f'[x_1, \mathsf{F}^{-1}x_1]$*) where* $f'[x_1, x_2, x_3]$ *(resp.* $f'[x_1, x_2]$*) is a* CTL *context.*

The proof is by structural induction on $f[x]$. (Here we only consider the $x_1\mathsf{S}x_2$ and $\mathsf{F}^{-1}x_1$ cases. The $\mathsf{X}^{-1}x_1$ case is quite similar to the $\mathsf{F}^{-1}x_1$ case, and may occur in fewer contexts.) By assumption, there is no past construct in

$f[x]$. We have four basic situations:

1. $f[x]$ **is some** $\mathsf{E}\varphi[x]\mathsf{U}\psi[x]$**:** Assume $f[x_1\mathsf{S}x_2]$ is a $\mathbf{N}CTL$ context. Then x does not occur in φ, which is then a CTL formula.

By ind. hyp., $\psi[x_1\mathsf{S}x_2]$ is equivalent to some separated $\psi'[x_1, x_2, x_1\mathsf{S}x_2]$ and $f[x_1\mathsf{S}x_2] \equiv \mathsf{E}\varphi\mathsf{U}\psi'[x_1, x_2, x_1\mathsf{S}x_2]$. In ψ', $x_1\mathsf{S}x_2$ can only appear under boolean combinators because of the separation property. We group all the occurrences of $x_1\mathsf{S}x_2$ using boolean manipulations and obtain:

$$f[x_1\mathsf{S}x_2] \equiv \mathsf{E}\varphi\mathsf{U}\Big((\alpha \wedge x_1\mathsf{S}x_2) \vee (\beta \wedge \neg(x_1\mathsf{S}x_2)) \vee \gamma\Big) \tag{A.1}$$

where α, β, γ and φ are pure-future (CTL). Then we use distributivity:

$$\mathsf{E}g\mathsf{U}(h \vee h') \equiv (\mathsf{E}g\mathsf{U}h) \vee (\mathsf{E}g\mathsf{U}h')$$

Then we may use the rules from Figure A.1 and extract all occurrences of $x_1\mathsf{S}x_2$ from the scope of $\mathsf{E\_U}$.

Now consider the F^{-1} case. $f[\mathsf{F}^{-1}x_1]$ is always a $\mathbf{N}CTL$ context and then x may occur in both φ and ψ. By ind. hyp., $\varphi[\mathsf{F}^{-1}x_1]$ and $\psi[\mathsf{F}^{-1}x_1]$ are equivalent to some separated $\varphi'[x_1, \mathsf{F}^{-1}x_1]$ and $\psi'[x_1, \mathsf{F}^{-1}x_1]$. Then $f[\mathsf{F}^{-1}x_1] \equiv \mathsf{E}\varphi'[x_1, \mathsf{F}^{-1}x_1]\mathsf{U}\psi'[x_1, \mathsf{F}^{-1}x_1]$. In φ' and ψ', $\mathsf{F}^{-1}x_1$ can only appear under boolean combinators because of the separation property. We use boolean manipulations to obtain:

$$\begin{aligned} f[\mathsf{F}^{-1}x_1] \equiv & \mathsf{E}\Big((\mathsf{F}^{-1}x_1 \wedge \alpha) \vee (\neg\mathsf{F}^{-1}x_1 \wedge \beta) \vee \gamma\Big) \\ & \mathsf{U}\Big((\mathsf{F}^{-1}x_1 \wedge \alpha') \vee (\neg\mathsf{F}^{-1}x_1 \wedge \beta') \vee \gamma'\Big) \end{aligned} \tag{A.2}$$

where $\alpha, \beta, \gamma, \alpha', \beta'$ and γ' are pure-future. Then we may use the rules from Figure A.2 and extract all occurrences of $\mathsf{F}^{-1}x_1$ from the scope of $\mathsf{E\_U}$.

2. $f[x]$ **is some** $\mathsf{EX}\varphi[x]$**:** We proceed similarly, using the ind. hyp. and distributivity:

$$\mathsf{EX}(h \vee h') \equiv \mathsf{EX}h \ \vee \ \mathsf{EX}h'$$

3. $f[x]$ **is some** $\mathsf{EG}\varphi[x]$**:** Then $f[\mathsf{F}^{-1}x_1] \equiv \mathsf{EG}\big(\varphi'[x_1, \mathsf{F}^{-1}x_1]\big)$. Because of the separation assumption, w.l.o.g. we can write $\mathsf{EG}\big(\varphi'[x_1, \mathsf{F}^{-1}x_1]\big)$ under the general form

$$f[\mathsf{F}^{-1}x_1] \equiv \mathsf{EG}\Big((\alpha \wedge \mathsf{F}^{-1}x_1) \vee (\beta \wedge \neg\mathsf{F}^{-1}x_1) \vee \gamma\Big) \tag{A.3}$$

Then we only need rule (R10) since no S or X^{-1} combinator can occur in this context.

4. Remaining cases: Finally, the other cases are obvious, or can be reduced to what we saw thanks to $\mathsf{AX}h \equiv \neg\mathsf{EX}\neg h$ and $\mathsf{A}g\mathsf{U}h \equiv \neg\mathsf{EG}\neg h \ \wedge \neg\Big(\mathsf{E}\neg h\mathsf{U}\neg g \wedge \neg h\Big)$.

Lemma A.4 *Let $f[x_1, \ldots, x_n]$ be a CTL context and assume that for $i = 1, .., n$, g_i is $y_i\mathsf{S}z_i$, or $\mathsf{F}^{-1}y_i$, or $\mathsf{X}^{-1}y_i$. If $f[g_1, \ldots, g_n]$ is a $\mathbf{N}CTL$ context, then* *it is equivalent to a separated* $f'[y_1, z_1, g_1, \dots, y_n, z_n, g_n]$ *with* $f'[y_1, z_1, u_1, \dots, y_n, z_n, u_n]$ *a* CTL *context.*

Proof. By induction on n, using Lemma A.3. □

Lemma A.5 *Let* $f[x_1, \dots, x_n]$ *be a* CTL *context and* $\psi_1^-, \dots, \psi_n^-$ *be pure-past* $\mathbf{N}CTL$ *formulas without* N. *If* $f[\psi_1^-, \dots, \psi_n^-]$ *is a* $\mathbf{N}CTL$ *formula, then it is equivalent to a separated* $\mathbf{N}CTL$ *formula.*

Proof. By induction on the maximum number of nested past combinator's in the ψ_i^-'s and using Lemma A.4. □

Lemma A.6 *Let* $f[x_1, \dots, x_n]$ *be a* CTL *context and* $\psi_1, \dots, \psi_n$ *be separated* $\mathbf{N}CTL$ *formulas without* N. *If* $f[\psi_1, \dots, \psi_n]$ *is a* $\mathbf{N}CTL$ *formula, then it is equivalent to a separated* $\mathbf{N}CTL$ *formula.*

Proof. Because it is separated, a ψ_i has the form $g_i^-[\varphi_{i,1}^+, \dots, \varphi_{i,m_i}^+]$ with pure-future $\varphi_{i,j}^+$'s and a pure-past $g_i^-[x_1, \dots, x_{k_i}]$. Lemma A.5 says that $f[g_1^-[x_{1,1}, \dots, x_{1,m_1}], \dots, g_n^-[x_{n,1}, \dots, x_{n,m_n}]]$ is equivalent to a separated $f'[x_{1,1}, \dots, x_{n,m_n}]$. Then $f'[\varphi_{1,1}^+, \dots, \varphi_{n,m_n}^+]$ is separated and equivalent to $f[\psi_1, \dots, \psi_n]$. □

Lemma A.7 *Any* $\mathbf{N}CTL$ *formula is equivalent to a separated* $\mathbf{N}CTL$ *formula.*

Proof. First, Lemma A.6 and structural induction allow us to separate any $\mathbf{N}CTL$ formula without N.

Now consider a formula $\mathsf{N}\varphi$ with φ a $\mathbf{N}CTL$ formula without N. Then φ is equivalent to a separated formula $g^-[\varphi_1^+, \dots, \varphi_n^+]$ where $g^-[x_1, \dots, x_n]$ is a pure-past context and all φ_i^+'s are CTL formulas. Given this separated formula, we obtain a CTL formula equivalent to $\mathsf{N}\varphi$ by applying the following equivalences:

$$\mathsf{N}(\psi \mathsf{S} \varphi) \equiv \mathsf{N}\varphi \qquad \mathsf{N}\mathsf{X}^{-1}\psi \equiv \bot \qquad \mathsf{N}\mathsf{F}^{-1}\varphi \equiv \mathsf{N}\varphi$$

$$\mathsf{N}\neg\varphi \equiv \neg\mathsf{N}\varphi \qquad \mathsf{N}(\psi \wedge \varphi) \equiv \mathsf{N}\psi \wedge \mathsf{N}\varphi$$

$$\mathsf{N}\varphi^+ \equiv \varphi^+ \quad \text{for any pure future formula } \varphi^+$$

We conclude the proof by using induction over the number of nested N. □

Finally the proof for Theorem 6.5 is obtained by the previous elimination of N: a given $\mathbf{N}CTL$ formula φ is equivalent to a separated $\mathbf{N}CTL$ formula φ', and $\mathsf{N}\varphi'$ is equivalent to a CTL formula φ''. Finally $\varphi \equiv_i N\varphi' \equiv \varphi''$.