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 $\text{N}_\text{From Now On}$, 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 $NCTL$, 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 $\Box (\text{crash} \Rightarrow \Diamond \neg \text{mistake})$. If we don’t allow past-time constructs, we may end up with the clumsier $\exists (\neg \text{mistake} \cup \text{crash})$.

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 $NCTL$, a fragment of $PCTL$ (i.e., $CTL + \text{Past}$) that extends the $CTL + \text{F^{-1}}$ solved in [16] and we show that the translation is correct even in a framework with fairness.

By necessity, $NCTL$ only permits a restricted use of the $\text{Since}$ 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 $NCTL$ 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 $NCTL$ and gives the translation theorem.

## 2 PCTL, or $CTL + \text{Past}$

Syntactically, the $PCTL$ logic we define is the $CTL + S + X^{-1} + N$ 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:

$$\phi, \psi ::= \neg \phi \mid \phi \land \psi \mid \text{EX}\phi \mid \text{E}\phi U\psi \mid \text{A}\phi U\psi \mid \text{X}^{-1}\phi \mid \phi S\psi \mid N\phi \mid a \mid b \mid \ldots$$

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

Standard abbreviations include $T, \bot, a \lor \psi, \phi \Rightarrow \psi, \ldots$ as well as

$$\begin{align*}
\text{EF}\phi & \overset{\text{def}}{=} \text{ET}U\phi & \text{EG}\phi & \overset{\text{def}}{=} \neg\text{AF}\neg\phi & \text{AX}\phi & \overset{\text{def}}{=} \neg\text{EX}\neg\phi \\
\text{AF}\phi & \overset{\text{def}}{=} \text{AT}U\phi & \text{AG}\phi & \overset{\text{def}}{=} \neg\text{EF}\neg\phi & \text{F}^{-1}\phi & \overset{\text{def}}{=} \text{TS}\phi
\end{align*}$$

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 $S = \langle Q_S, R_S, I_S, I_S, \Phi_S \rangle$ 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,
- $I_S : Q_S \to 2^{Prop}$ labels every state with the propositions it satisfies,
- $I_S \subseteq Q_S$ is a set of initial states,
- $\Phi_S$ 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. $\pi^i$) 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 $\Phi$ on $S$-computations satisfying the following properties:

1. fairness only depends on the “end” of a computation: for all $\pi$ and suffix $\pi^n$, $\Phi(\pi)$ 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 $\pi'$ 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 $I_S(q)$ 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 $\pi$, i.e. the history $q_0\ldots q_i$. By extension, we write $\Pi(\sigma)$ for the set of all fair computations starting from $\sigma$.

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 $\sigma$. 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 $\sigma$, in some FKS $S$, satisfies a formula $\varphi$, written $\sigma \models_s \varphi$, by induction over the structure of $\varphi$.

$$\begin{align*}
\sigma &\models a \quad \text{iff } a \in l(q_n), \\
\sigma &\models \neg \varphi \quad \text{iff } \sigma \not\models \varphi, \\
\sigma &\models \varphi \land \psi \quad \text{iff } \sigma \models \varphi \text{ and } \sigma \models \psi, \\
\sigma &\models \exists X \varphi \quad \text{iff there exists } \pi \in \Pi(\sigma) \text{ s.t. } \pi_{i+1} \models \varphi, \\
\sigma &\models \exists \varphi \cup \psi \quad \text{iff there exists } \pi \in \Pi(\sigma) \text{ and } k \geq 0 \\
&\quad \text{s.t. } \pi_{i+k} \models \psi \text{ and } \pi_{i+i} \models \varphi \text{ for all } 0 \leq i < k, \\
\sigma &\models \forall X \varphi \quad \text{iff for all } \pi \in \Pi(\sigma) \text{ there exists a } k \geq 0 \\
&\quad \text{s.t. } \pi_{i+k} \models \psi \text{ and } \pi_{i+i} \models \varphi \text{ for all } 0 \leq i < k, \\
\sigma &\models \exists^{-1} \varphi \quad \text{iff } n > 0 \text{ and } \sigma' \models \varphi \text{ (where } \sigma' = q_0\ldots q_{n-1}), \\
\sigma &\models \forall S \psi \quad \text{iff there exists } k \leq n \text{ s.t. } \sigma_k \models \psi \text{ and } \\
&\quad \sigma_i \models \varphi \text{ for all } k < i \leq n, \\
\sigma &\models \forall \varphi \quad \text{iff } q_n \models \varphi.
\end{align*}$$

Fig. 1. Semantics of $PCTL$

Then satisfaction can be defined over fair Kripke structures through

$$S \models \varphi \overset{\text{def}}{=} \forall \pi \models \varphi \text{ for all } \pi \in \Pi_S(I_S)$$

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

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

2.3 $N$, or “From now on”

The $N$ combinator was introduced in [16]. $N \varphi$ reads “from now on, $\varphi$ holds”, or “starting anew from the current state, $\varphi$ 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 $AG(crash \Rightarrow F^{-1} \text{mistake})$.

Assume we now want to state that after a proper reset is done, any crash is preceded by an earlier mistake. Then $AG[\text{reset} \Rightarrow AG(crash \Rightarrow F^{-1} \text{mistake})]$ 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 $AG[\text{reset} \Rightarrow NAG(crash \Rightarrow F^{-1} \text{mistake})]$ (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

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

(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 \( \text{N} \) 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 \( \text{N} \). 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.

<table>
<thead>
<tr>
<th>Structure of past</th>
<th>Determined</th>
<th>Non-determined</th>
<th>Finite</th>
<th>Infinite</th>
<th>Cumulative</th>
<th>Non-cumulative</th>
</tr>
</thead>
<tbody>
<tr>
<td>[8], [12], [18], [1], [32], [7], [28], [20], [23]</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Linear time temporal logics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[24], [29], [26], [27]</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>non-Ockhamist past</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[14]</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ockhamist past</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[31], [30]</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[9], [2]</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[15], [14]</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[16]</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

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

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*. 

7
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:

- \(\text{Open\_Door}_i\) \((i = 1, \ldots, n)\), true if the floor door at floor \(i\) is open,
- \(\text{Call}_i\) (resp. \(\text{Send}_i\)) \((i = 1, \ldots, n)\), true if the call-button at floor \(i\) (resp. send-button for \(i\)) is pressed,
- \(\text{Call\_Light}_i\) (resp. \(\text{Send\_Light}_i\)) \((i = 1, \ldots, n)\), true if the indicator light for the \(i\)-th call- (resp. send-) button is on,
- \(\text{At}_i\) \((i = 1, \ldots, n)\), 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:

\[
\bigwedge_{i=1}^{n} \text{AG}(\text{Open\_Door}_i \Rightarrow \text{At}_i) \quad \text{(S1)}
\]

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:

\[
\text{Servicing}_i \overset{\text{def}}{=} \text{At}_i \land \text{Open\_Door}_i \quad (i = 1, \ldots, n) \quad \text{(D1)}
\]

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

\[
\bigwedge_{i=1}^{n} \text{AG}[\text{Call}_i \Rightarrow (\text{Servicing}_i \lor \text{Call\_Light}_i)] \quad \text{(S2.1)}
\]

\[
\bigwedge_{i=1}^{n} \text{AG}[\text{Send}_i \Rightarrow (\text{Servicing}_i \lor \text{Send\_Light}_i)] \quad \text{(S2.2)}
\]

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

\[
\text{E}_\varphi W_\psi \overset{\text{def}}{=} -A(\neg \varphi) U (\neg \psi) \land \neg \psi \quad \text{A}_\varphi W_\psi \overset{\text{def}}{=} -E(\neg \psi) U (\neg \psi)
\]

and write

\[
\bigwedge_{i=1}^{n} \text{AG}[\text{Call\_Light}_i \Rightarrow A\text{Call\_Light}_i W\text{Servicing}_i] \quad \text{(S2.3)}
\]

\[
\bigwedge_{i=1}^{n} \text{AG}[\text{Send\_Light}_i \Rightarrow A\text{Send\_Light}_i W\text{Servicing}_i] \quad \text{(S2.4)}
\]
Then, lights are turned off when the request is fulfilled:

\[ \bigwedge_{i=1}^{n} \text{AG}[\text{Servicing}_i \Rightarrow (\neg \text{Call}_{Light_i} \land \neg \text{Send}_{Light_i})] \quad (S2.5) \]

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 \( \text{AG}(\text{Call}_{Light_i} \Rightarrow F^{-1} \text{Call}_i) \) does not work because it allows one early call to account for all future turnings on of the indicator light. Rather, we mean

\[ \bigwedge_{i=1}^{n} \text{AG}[\text{Call}_{Light_i} \Rightarrow (\text{Call}_{Light_i} \land \lnot \text{Call}_i)] \quad (S2.6) \]
\[ \bigwedge_{i=1}^{n} \text{AG}[\text{Send}_{Light_i} \Rightarrow (\text{Send}_{Light_i} \land \lnot \text{Send}_i)] \quad (S2.7) \]

An alternative possibility would have been to use \( \text{N} \) combinator, suited to this kind of situation, and state:

\[ \bigwedge_{i=1}^{n} \text{AG}[\neg \text{Call}_{Light_i} \Rightarrow \text{NAG}(\text{Call}_{Light_i} \Rightarrow F^{-1} \text{Call}_i)] \quad (S2.6') \]
\[ \bigwedge_{i=1}^{n} \text{AG}[\neg \text{Send}_{Light_i} \Rightarrow \text{NAG}(\text{Send}_{Light_i} \Rightarrow F^{-1} \text{Send}_i)] \quad (S2.7') \]

(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”.

\[ \bigwedge_{i=1}^{n} \text{AG} \left[ (\text{Call}_{Light_i} \Leftrightarrow \neg \text{Servicing}_i \land (\text{Call}_i \land \neg \text{Servicing}_i)) \right] \land \left[ (\text{Send}_{Light_i} \Leftrightarrow \neg \text{Servicing}_i \land (\text{Send}_i \land \neg \text{Servicing}_i)) \right] \quad (S2') \]

4.3.3 P3. Service

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

\[ \bigwedge_{i=1}^{n} \text{AG}[\text{Request}_i \Rightarrow \text{AFServicing}_i] \quad (S3) \]

where \( \text{Request}_i \overset{\text{def}}{=} \text{Call}_i \lor \text{Send}_i \quad (i = 1, \ldots, n) \quad (D2) \)

4.3.4 P4. Smart service

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

\[ \text{PendingRequest}_i \overset{\text{def}}{=} \text{Call}_{Light_i} \lor \text{Send}_{Light_i} \quad (i = 1, \ldots, n) \quad (D3) \]

\[ \text{SomePendingRequest} = \bigvee_{i=1}^{n} \text{PendingRequest}_i \quad (D4) \]
and can now write that a floor is only serviced if there is a pending request for it

\[ \bigwedge_{i=1}^{n} \text{AG} \left[ \text{Servicing}_i \Rightarrow (\text{Servicing}_i S \text{Pending-Request}_i) \right] \] (S4.1)

and that the cabin is motionless unless there is some request

\[ \bigwedge_{i=1}^{n} \text{AG} \left( \text{At}_i \Rightarrow [\text{A At}_i W \text{Some-Pending-Request}] \right) \] (S4.2)

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

\[ \text{AG} \left( \text{Between-Floors} \Rightarrow [\text{A Between-Floors} W \text{Some-Pending-Request}] \right) \] (S4.3)

where \( \text{Between-Floors} \overset{\text{def}}{=} \bigwedge_{i=1}^{n} \neg \text{At}_i \) (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

\[ \text{Not-Servicing} \overset{\text{def}}{=} \bigwedge_{i=1}^{n} \neg \text{Servicing}_i \] (D6)

\[ \bigwedge_{k=1}^{n} \text{AG} \left[ (\text{At}_k \land \text{Pending-Request}_k) \Rightarrow \text{A Not-Servicing} W \text{Servicing}_k \right] \] (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:

\[ \text{At}_j \text{From}_i \overset{\text{def}}{=} \text{Servicing}_j \land (\text{Servicing}_j \lor \text{Not-Servicing}) S \text{Servicing}_i \]

\[ (i, j = 1, \ldots, n) \] (D7)
and write \((i, j) \triangleq \{k \mid i < k < j\text{ or } j < k < i\}\) for the set of intermediary floors between \(i\) and \(j\). Now “diligent servicing” can be stated

\[
\bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} \text{AG } At_{j} \text{From}_{i} \Rightarrow \left( (\text{Servicing}_{j} \lor \text{Not-Servicing}) \right) \Rightarrow \left( (\text{Servicing}_{i} \land \bigwedge_{k \in (i, j)} \neg \text{Request}_{k}) \right)
\]  

(S5)

4.3.6 P6. Direct movements
We understand this property in terms of positions “\(At_{i}\)” rather than in terms of services “\(\text{Servicing}_{i}\)”. 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 \(i\).

\[
\bigwedge_{i,k=1}^{n} \bigwedge_{j \notin (i,k)} \text{AG} \left( \neg \text{At}_{i} \Rightarrow \text{AG} \left( \text{At}_{k} \land F^{-1} \text{At}_{j} \Rightarrow F^{-1} \bigvee_{i \notin (i,k)} \text{Servicing}_{i} \right) \right)
\]  

(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 \(i\).

\[
U_{p} \triangleq \bigvee_{i=2}^{n} \left( (\text{At}_{i} \lor \text{Between-Floors}) \text{SA}_{i-1} \right) \land \text{ABetween-Floors} \cup \text{At}_{i}
\]  

(D8)

\[
Down \triangleq \bigvee_{i=1}^{n-1} \left( (\text{At}_{i} \lor \text{Between-Floors}) \text{SA}_{i+1} \right) \land \text{ABetween-Floors} \cup \text{At}_{i}
\]  

(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.

\[
\text{AG} \bigwedge_{i=1}^{n} \left[ (\text{Servicing}_{i} \land \text{Down} \land (\bigvee_{j<i} \text{Pending-Request}_{j})) \right] \Rightarrow \neg \text{E} \left( (\text{Servicing}_{i} \lor \text{Not-Servicing}) \cup (\bigvee_{k<i} \text{Servicing}_{k}) \right)
\]  

(S7.1)

\[
\text{AG} \bigwedge_{i=1}^{n} \left[ (\text{Servicing}_{i} \land U_{p} \land (\bigvee_{j>i} \text{Pending-Request}_{j})) \right] \Rightarrow \neg \text{E} \left( (\text{Servicing}_{i} \lor \text{Not-Servicing}) \cup (\bigvee_{k>i} \text{Servicing}_{k}) \right)
\]  

(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 \( CTL + F^{-1} + N \) 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-checking.

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 $\varphi$ using past-time constructs, we need to translate it into some $\varphi'$ with only future-time constructs with the following correctness criterion:

$$\text{for any FKS } S, S \models \varphi \iff S \models \varphi'$$  \hspace{1cm} (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 $\sigma$ in all fair Kripke structures, $\sigma \models f \iff \sigma \models g$.

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

*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. $X^{-1}T \equiv_i \perp$ (because $X^{-1}T$ doesn’t hold for a starting point) but of course $X^{-1}T \neq \perp$. Note that $\mathbb{N}$ helps understand the links between the two notions of equivalences:

$$\varphi \equiv_i \psi \iff \mathbb{N}\varphi \equiv \mathbb{N}\psi$$

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) $CTL + S$ cannot be translated into $CTL$.

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

These two results are based on the following observations: (1) the formula $\text{EG}(a \lor X^{-1}a \lor \neg X^{-1}T)$ cannot be expressed in $CTL$, and (2) it is possible, by using embedded $S$ combinators, to build a $CTL + S$ formula equivalent to the
\(\text{CTL}^*\) formula \(E(c \lor aUb)Ud\) which cannot be expressed in \(\text{CTL}\).

In view of these impossibility results, one has to look for a fragment of \(P\text{CTL}\) that can be translated into \(\text{CTL}\). Indeed, we know that

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

This result only partly helps us because our LIFT specification from section 4 was not written in the \(\text{CTL} + F^{-1} + N\) 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 \(\text{CTL}\) can push it far beyond \(\text{CTL}\) expressivity, there exists a precisely delineated fragment of \(P\text{CTL}\) that (1) support the LIFT specification, and (2) can be translated into \(\text{CTL}\). For example, notwithstanding its occurrences of \(S\), formula (S2.6) is initially equivalent to a \(\text{CTL}\) formula:

\[
\bigwedge_{i=1}^{n} AG[\neg Call_{Light_i} \land \neg Call_i] \Rightarrow \neg E(\neg Call_i)U(Call_{Light_i} \land \neg Call_i]
\]

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 \(\text{NCTL}\), the aforementioned fragment of \(P\text{CTL}\):

**Definition 6.3** The logic \(\text{NCTL}\)

\[
\text{NCTL} \ni \varphi, \psi ::= \lambda \mid \varphi \lor \psi \mid \neg \varphi \mid E\varphi \mid E\lambda U \varphi \mid \lambda S \mu \mid X^{-1} \varphi
\]

\[
\lambda, \mu ::= a \mid \lambda \land \mu \mid \neg \lambda \mid EX \lambda \mid E\lambda U \mu \mid A\lambda U \mu \mid F^{-1} \lambda \mid N \varphi
\]

Thus \(\text{NCTL}\) forbids occurrences of \(S\) and \(X^{-1}\) in the scope of \(S\) or \(A\land U\) (except if a \(N\) is in between) and in the left-hand side of \(E\land U\). In such contexts, only limited formulas \(\lambda\) and \(\mu\) 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 \(\text{NCTL}\).

Now we have the following result:

**Theorem 6.5** \(\text{NCTL}\) can be (effectively) translated into \(\text{CTL}\)

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 \(P\text{CTL}\) 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 NCTL formula is equivalent to a separated NCTL 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:

\[
\begin{align*}
E a \mathbin{U} (b \land cSd) & \quad \text{a NCTL formula} \\
\equiv \quad & \quad (cSd \land E(a \land c) \mathbin{U} (b \land c) \\
& \quad \equiv \quad E a \mathbin{U} (a \land d \land EX E(a \land c)U(b \land c)) \\
& \quad \lor E a \mathbin{U} (b \land d) \\
\equiv \quad & \quad E a \mathbin{U} (a \land d \land EX E(a \land c)U(b \land c)) \\
& \quad \lor E a \mathbin{U} (b \land d) \\
& \quad \text{a separated NCTL formula} \\
& \quad \text{a CTL formula}
\end{align*}
\]

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 NCTL+ where boolean combinations of path-formulas are allowed under a path quantifier, and to an even larger NECTL+, 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 equiva-
lent $CTL$ specification. This can be done thanks to a new translation theorem, extending to $NCTL$ our earlier work on $CTL + F^{-1}$.

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 $NCTL$ 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


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 combinator.

We follow the steps of our earlier proof for the separation $CTL + F^{-1} + N$ 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) \( E(\varphi U(\alpha \land X^{-1}x)) \equiv \alpha \land X^{-1}x \lor E(\varphi U(\varphi \land x \land EX\alpha)) \)

(R2) \( E(\varphi U(\alpha \land \neg X^{-1}x)) \equiv \alpha \land \neg X^{-1}x \lor E(\varphi U(\varphi \land \neg x \land EX\alpha)) \)

(R3) \( EX(\alpha \land X^{-1}x) \equiv x \land EX\alpha \)

(R4) \( EX(\alpha \land \neg X^{-1}x) \equiv \neg x \land EX\alpha \)

(R5) \( E(\varphi U(\alpha \land xSy)) \)
\[ \equiv E\varphi U(\alpha \land y) \lor E\varphi U(\varphi \land y \land EX(\varphi \land x)U(\alpha \land x)) \]
\[ \land xSy \land (\alpha \lor E(\varphi \land x)U(\alpha \land x)) \]

(R6) \( E(\varphi U(\alpha \land \neg(xSy))) \)
\[ \equiv E\varphi U(\alpha \land \neg x \land \neg y) \lor E\varphi U(\varphi \land \neg x \land \neg y \land E(\varphi \land \neg y)U(\alpha \land \neg y)) \]
\[ \land \neg(xSy) \land (\alpha \lor E(\varphi \land \neg y)U(\alpha \land \neg y)) \]

(R7) \( EX(\alpha \land xSy) \equiv EX(\alpha \land y) \lor xSy \land EX(\alpha \land x) \)

(R8) \( EX(\alpha \land \neg(xSy)) \equiv EX(\alpha \land \neg x \land \neg y) \lor (xSy) \land EX(\alpha \land \neg y) \)

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

(R9) \( E((F^{-1}x \land \alpha) \lor (\neg F^{-1}x \land \beta) \lor \gamma)U((F^{-1}x \land \alpha') \lor (\neg F^{-1}x \land \beta') \lor \gamma') \)
\[ \equiv F^{-1}x \land E(\alpha \lor \gamma)U(\alpha' \lor \gamma') \]
\[ \lor \neg F^{-1}x \land E(\neg x \land (\beta \lor \gamma))U(x \land E(\alpha \lor \gamma)U(\alpha' \lor \gamma')) \]
\[ \lor \neg F^{-1}x \land E(\neg x \land (\beta \lor \gamma))U(\neg x \land (\beta' \lor \gamma')) \]

(R10) \( EG((F^{-1}x \land \alpha) \lor (\neg F^{-1}x \land \beta) \lor \gamma) \)
\[ \equiv F^{-1}x \land EG(\alpha \lor \gamma) \]
\[ \lor \neg F^{-1}x \land E(\neg x \land (\beta \lor \gamma))U(x \land EG(\alpha \lor \gamma)) \]
\[ \lor \neg F^{-1}x \land EG(\neg x \land (\beta \lor \gamma)) \]

(R11) \( EX(\alpha \land F^{-1}x) \equiv EX(\alpha \land x) \lor F^{-1}x \land EX\alpha \)

(R12) \( EX(\alpha \land \neg F^{-1}x) \equiv \neg F^{-1}x \land EX(\alpha \land \neg x) \)

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 \( \varphi, x, y, \alpha, \beta, \gamma, \alpha', \beta' \) and \( \gamma' \).

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
Lemma A.3 Let $f[x]$ be a CTL context. If $f[x_1 x_2]$, $f'[x_1, x_2, x_1 x_2]$ (resp. $f[x_1]$, $f'[x_1, x_1 x_2]$) is a separated NCTL context, then $f[x_1 x_2]$, $f'[x_1, x_1 x_2]$ (resp. $f[x_1]$, $f'[x_1, x_1 x_2]$) is equivalent to a separated NCTL context $f'[x_1, x_2, x_1 S x_2]$ (resp. $f'[x_1, x_1 x_2]$), 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 $x_1 S x_2$ and $F^{-1} x_1$ cases. The $X^{-1} x_1$ case is quite similar to the $F^{-1} x_1$ 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 \(E\varphi[x]U\psi[x]\):** Assume \(f[x_1sx_2]\) is a \(\mathbf{NCTL}\) context. Then \(x\) does not occur in \(\varphi\), which is then a \(\text{CTL}\) formula.

   By ind. hyp., \(\psi[x_1sx_2]\) is equivalent to some separated \(\psi'[x_1, x_2, x_1sx_2]\) and \(f[x_1sx_2] \equiv E\varphi U \psi'[x_1, x_2, x_1sx_2]\). In \(\psi'\), \(x_1sx_2\) can only appear under boolean combinators because of the separation property. We group all the occurrences of \(x_1sx_2\) using boolean manipulations and obtain:

   \[
   f[x_1sx_2] \equiv E\varphi U \left((\alpha \land x_1sx_2) \lor (\beta \land \neg(x_1sx_2)) \lor \gamma\right) \tag{A.1}
   \]

   where \(\alpha, \beta, \gamma\) and \(\varphi\) are pure-future (\(\text{CTL}\)). Then we use distributivity:

   \[
   EqU(h \lor h') \equiv (EqUh) \lor (EqUh')
   \]

   Then we may use the rules from Figure A.1 and extract all occurrences of \(x_1sx_2\) from the scope of \(E.U\).

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

   \[
   f[F^{-1}x_1] \equiv E\left((F^{-1}x_1 \land \alpha) \lor (\neg F^{-1}x_1 \land \beta) \lor \gamma\right) \tag{A.2}
   \]

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

2. **\(f[x]\) is some \(EX\varphi[x]\):** We proceed similarly, using the ind. hyp. and distributivity:

   \[
   EX(h \lor h') \equiv EXh \lor EXh'
   \]

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

   \[
   f[F^{-1}x_1] \equiv EG \left((\alpha \land F^{-1}x_1) \lor (\beta \land \neg F^{-1}x_1) \lor \gamma\right) \tag{A.3}
   \]

   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 \(AXh \equiv \neg EXh\) and \(AgUh \equiv \neg EG\neg h \land \neg \left(E\neg hUg \land \neg h\right)\).

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


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

Proof. By induction on the maximum number of nested past combinator’s in the $\psi_i$’s and using Lemma A.4. □

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

Proof. Because it is separated, a $\psi_i$ has the form $g_i^- \left[ \varphi_{i,1}^+, \ldots, \varphi_{i,m_i}^+ \right]$ with pure-future $\varphi_{i,j}^+$’s and a pure-past $g_i^- [x_1, \ldots, x_k]$. Lemma A.5 says that $f[g_i^- [x_{1,1}, \ldots, x_{1,m_{1},i}], \ldots, g_i^- [x_{n,1}, \ldots, x_{n,m_{n},i}]]$ is equivalent to a separated $f'[x_{1,1}, \ldots, x_{n,m_{n}}]$. Then $f'[\varphi_{i,1}^+, \ldots, \varphi_{i,m_i}^+]$ is separated and equivalent to $f[\psi_1, \ldots, \psi_n]$. □

Lemma A.7 Any NCTL formula is equivalent to a separated NCTL formula.

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

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

\[ \mathbb{N}(\psi S\varphi) \equiv \mathbb{N}\varphi \quad \mathbb{N}X^{-1}\psi \equiv \bot \quad \mathbb{N}F^{-1}\varphi \equiv \mathbb{N}\varphi \]

\[ \mathbb{N} \neg \varphi \equiv \neg \mathbb{N}\varphi \quad \mathbb{N}(\psi \land \varphi) \equiv \mathbb{N}\psi \land \mathbb{N}\varphi \]

\[ \mathbb{N}\varphi^+ \equiv \varphi^+ \text{ for any pure future formula } \varphi^+ \]

We conclude the proof by using induction over the number of nested $\mathbb{N}$. □

Finally the proof for Theorem 6.5 is obtained by the previous elimination of $\mathbb{N}$: a given NCTL formula $\varphi$ is equivalent to a separated NCTL formula $\varphi'$, and $\mathbb{N}\varphi'$ is equivalent to a CTL formula $\varphi''$. Finally $\varphi \equiv_{i\cdot} \mathbb{N}\varphi' \equiv \varphi''$. 

22