Specification in CTL+Past for Verification in CTL

F. Laroussinie and Ph. Schnoebelen
Lab. Specification and Verification
ENS de Cachan & CNRS URA 2236
61, av. Pdt Wilson; 94235 Cachan Cedex; FRANCE
email: {fl,phs}@lsv.ens-cachan.fr

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.

We investigate the feasibility of verification with $PCTL$ and 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 [Pnueli, 1977], the *temporal logic (TL) framework* [Manna and Pnueli, 1992, Emerson, 1990] has long been recognized as a fundamental approach to the formal specification and verification of reactive systems. 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 [Burch et al., 1992, Clarke et al., 1994, McMillan, 1993].

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, not all interesting behavioral properties can be expressed in the most commonly used temporal logics. On the practical side, 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 [Lichtenstein et al., 1985] it has been known that allowing both past-time and future-time constructs makes TL specification easier and more natural: with past-time the English sentence "*if a crash occurs, then a mistake necessarily 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 $\neg\exists(\neg mistake \mathsf{U} crash)$ (using the "until" combinator).

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.

Our contribution. In this article, we describe $PCTL$, our version of $CTL+Past$ and discuss its pros and cons for the specification and verification of reactive systems. $PCTL$ is a branching-time logic with Ockamist Past: we spend some time discussing the different semantic frameworks for Past we found in the literature, classifying the existing proposals, and explaining the reasons behind our choice.

We study the feasibility of verification problems for $PCTL$ and show that model-checking is PSPACE-complete, even for very restricted fragments of $PCTL$. This reuses and extends [Kupferman and Pnueli, 1995, Demri and Schnoebelen, 1998].

We advocate a *translation-based* approach for the verification of $CTL + Past$ and present a translation algorithm from $NCTL$ (a carefully delineated fragment of $PCTL$) into CTL. This result extends expressivity results about branching time logics with past presented in [Laroussinie and Schnoebelen, 1995]. By necessity, $NCTL$ only permits a restricted use of the past-time modalities. We show, through an extensive example (the well-known Lift example [Barringer, 1987, Hale, 1989]) 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 $PCTL$.

Related work. • Our $PCTL$ logic was already proposed in [Laroussinie and Schnoebelen, 1995] where a first translation algorithm is proposed. [Kupferman and Pnueli, 1995] proposes extensions of CTL and CTL^* with two different kinds of Past (see section 3). Their CTL_{lp} actually

is our $PCTL$ without its "Now" combinator. They are concerned with verification and expressivity but not with translation issues.
• Only a few papers (e.g. [Ramakrishna et al., 1992, Kesten et al., 1993, Kupferman and Grumberg, 1996, Raviv, 1997]) propose model-checking algorithms (actual algorithms and not decidability proofs) for a temporal logic with past. None of the widely available model-checking tools supports past-time constructs.
• Translations between past-and-future logics into pure-future logics are known since [Gabbay et al., 1980]. They were used to argue that past-time does not add theoretical expressivity. Only in [Laroussinie and Schnoebelen, 1995] they were suggested as an actual practical approach to the model-checking problem for extended logics.
• Another translation-based approach, reducing LTL model-checking to CTL model-checking, is proposed in [Clarke et al., 1997].

Plan of the article. We assume familiarity with the temporal logics used in specification and verification of reactive systems and often refer to [Emerson, 1990]. 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 discusses the complexity of satisfiability and model-checking for $PCTL$, leading to our proposal, in section 5, of a translation-based approach for verification problems in $PCTL$. Section 6 proves our main theorem, where we translate $NCTL$ into CTL. The article closes with an appendix describing the Lift example, a comprehensive case study of specification with $PCTL$ and $NCTL$.

2 $PCTL$, or CTL+Past

Syntactically, the $PCTL$ logic we define is the $CTL+\mathsf{S}+\mathsf{X}^{-1}+\mathsf{N}$ of [Laroussinie and Schnoebelen, 1995]. 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 [Laroussinie and Schnoebelen, 1995] only used structures without fairness.

2.1 Syntax

We assume a given non-empty finite set $Prop = \{a, b, \ldots\}$ of *atomic propositions*. $PCTL$ formulae 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

$$\begin{array}{lllllllll} \mathsf{EF}\varphi & \stackrel{\text{def}}{=} & \mathsf{E}\top\mathsf{U}\varphi & \quad \mathsf{EG}\varphi & \stackrel{\text{def}}{=} & \neg\mathsf{AF}\neg\varphi & \quad \mathsf{AX}\varphi & \stackrel{\text{def}}{=} & \neg\mathsf{EX}\neg\varphi \\ \mathsf{AF}\varphi & \stackrel{\text{def}}{=} & \mathsf{A}\top\mathsf{U}\varphi & \quad \mathsf{AG}\varphi & \stackrel{\text{def}}{=} & \neg\mathsf{EF}\neg\varphi & \quad \mathsf{F}^{-1}\varphi & \stackrel{\text{def}}{=} & \top\mathsf{S}\varphi \end{array}$$

2.2 Semantics

We use Kripke structures with abstract fairness conditions as our underlying model [Hennessy and Stirling, 1985]. This is a classical approach to the semantics of reactive systems, and it applies to most current implementations, e.g. the SMV model-checker [McMillan, 1993].

PCTL formulae are interpreted over *histories* (that is, a current state with a past) in fair Kripke structures. 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 arises.

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 finite sequence of R-related states can be continued into 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 a way of telling fair computations from unfair ones. Formally,

Definition 2.2. A *fairness constraint* (for S) is a predicate Φ on S-computations satisfying the following properties:
(suffix-closure) *fairness only depends on the "end" of a computation:* for all π and suffix π^n, $\Phi(\pi)$ iff $\Phi(\pi^n)$,
(density) *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 $q_0 q_1 \ldots q_n$.

We let $\Pi_S(q)$ denote the set of fair computations starting from q, and let $\Pi_S(I_S)$ (resp. $\Pi(S)$) denote the union of all $\Pi_S(q)$ for $q \in I_S$ (resp. for $q \in Q_S$).

In practice, fairness constraints are often described in a finite way through some simple mechanism (e.g. repeated states à la Büchi, or the requirement that possible transitions are chosen according to some fair scheduling scheme [Manna and Pnueli, 1992]). A special case is the *trivial fairness constraint* where all computations are considered fair: we speak of KS without fairness constraint.

A *history* is a **non-empty** finite sequence $q_0 q_1 \ldots q_n$ s.t. $(q_i, q_{i+1}) \in R$ for all $i < n$. We use $\sigma, \ldots$ to denote histories. We write $|\sigma|$ for the number of steps in history σ: $|q_0 q_1 \ldots q_n| \stackrel{\text{def}}{=} n$.

Histories are prefixes of computations. Given $i \geq 0$ and $\pi = q_0 q_1 \ldots$, 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 with σ.

The intuition is that a history $\sigma = q_0 q_1 \ldots q_n$ denotes a "current state", q_n, of some computation still in progress, with the additional information that the past of this computation has been $\sigma_{|n-1}$. From this history, the system can proceed to a next state q_{n+1} and then the past will be $\sigma' = q_0 \ldots q_n q_{n+1}$. Any state q is a history by itself (that is, a history reduced to the present state q, with an empty past).

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.

Let σ be $q_0, \ldots, q_n$, we define :

$\sigma \models a$	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. $\pi_{\vert n+1} \models \varphi$
$\sigma \models \mathsf{E}\varphi\mathsf{U}\psi$	iff	there exists $\pi \in \Pi(\sigma)$ and $k \geq 0$ s.t. $\pi_{\vert n+k} \models \psi$ and $\pi_{\vert n+i} \models \varphi$ 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. $\pi_{\vert n+k} \models \psi$ and $\pi_{\vert n+i} \models \varphi$ 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. $\sigma_{\vert k} \models \psi$ and $\sigma_{\vert i} \models \varphi$ for all $k < i \leq n$
$\sigma \models \mathsf{N}\varphi$	iff	$q_n \models \varphi$

Figure 1: Semantics of $PCTL$

Then satisfaction can be defined over fair Kripke structures through

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

adopting the anchored-view of satisfaction [Manna and Pnueli, 1989] common in temporal specifications [Emerson, 1990].

The semantics we have just given 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"; $\varphi\mathsf{S}\psi$: "ψ held at some time in the past, and φ has been holding ever since".

When past-time is involved, it is natural to distinguish between two different notions of equivalence between formulae:

Definition 2.3.

1. Two formulae φ and ψ are *(globally) equivalent*, written $\varphi \equiv \psi$, when for all histories σ in all fair Kripke structures, $\sigma \models \varphi \;\Leftrightarrow\; \sigma \models \psi$.
2. Two formulae φ and ψ are *initially equivalent*, written $\varphi \equiv_i \psi$, when for all states q in all fair Kripke structures, $q \models \varphi \;\Leftrightarrow\; q \models \psi$.

Clearly $\equiv_i$ is weaker than $\equiv$: $\varphi \equiv \varphi'$ implies $\varphi \equiv_i \varphi'$ but the converse is not true in general. For example $\mathsf{X}^{-1}\top \equiv_i \bot$ (because $\mathsf{X}^{-1}\top$ doesn't hold for a starting point) but of course $\mathsf{X}^{-1}\top \not\equiv \bot$.

Initial equivalence, $\equiv_i$, is the natural equivalence when comparing specifications of FKS's:

Proposition 2.4. *$\varphi \equiv_i \varphi'$ iff for all FKS S: $S \models \varphi \;\Leftrightarrow\; S \models \varphi'$.*

One difficulty is that $\equiv_i$ is not substitutive: if $\psi[.]$ is some context, then $\varphi \equiv_i \varphi'$ does not entail $\psi[\varphi] \equiv_i \psi[\varphi']$ (for example : $\mathsf{EXX}^{-1}\top \not\equiv_i \mathsf{EX}\bot$). Thus, in many situations where we have to transform subformulae, we use global equivalence, $\equiv$, (which is substitutive) as the equivalence criterion.

2.3 N, or "From now on"

$PCTL$ without N is the CTL_{lp} logic of [Kupferman and Pnueli, 1995]. The N combinator was introduced in [Laroussinie and Schnoebelen, 1995]. $\mathsf{N}\varphi$ reads "from now on, φ holds", or "starting anew from the current state, φ holds". Here we present some explanations and motivations regarding this new combinator.

Assume we want to state that any *crash* in the future is preceded by an earlier *mistake*. This can be written in $PCTL$ as $\mathsf{AG}(crash \;\Rightarrow\; \mathsf{F}^{-1}mistake)$. Assume we now want to state that *after a proper reset is done*, any crash is preceded by an earlier mistake. Then $\mathsf{AG}[reset \;\Rightarrow\; \mathsf{AG}(crash \;\Rightarrow\; \mathsf{F}^{-1}mistake)]$ 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 combination of N and F^{-1} gives a natural way to express such a property with $\mathsf{AG}[reset \Rightarrow \mathsf{NAG}(crash \;\Rightarrow\; \mathsf{F}^{-1}mistake)]$ (see [Laroussinie and Schnoebelen, 1995] for more details).

A more theoretical reason for the introduction of N is that it is the natural bridge between initial and global equivalence:

Proposition 2.5. [Laroussinie and Schnoebelen, 1995] *For all φ, ψ in $PCTL$, $\varphi \equiv_i \psi$ iff $\mathsf{N}\varphi \equiv \mathsf{N}\psi$.*

N is only meaningful when past-time is allowed, and indeed if φ is a pure-future formula, then $\mathsf{N}\varphi \equiv \varphi$.

N enjoys some additional distributivity and elimination properties:

Proposition 2.6. [Laroussinie and Schnoebelen, 1995] *For all φ, ψ in $PCTL$:*

$$\begin{aligned} \mathsf{N}(\varphi \wedge \psi) &\equiv \mathsf{N}\varphi \wedge \mathsf{N}\psi, & \mathsf{NX}^{-1}\varphi &\equiv \bot, \\ \mathsf{N}\neg\varphi &\equiv \neg\mathsf{N}\varphi, & \mathsf{N}(\varphi\mathsf{S}\psi) &\equiv \mathsf{N}\psi, \\ \mathsf{NN}\varphi &\equiv \mathsf{N}\varphi, & \mathsf{NF}^{-1}\varphi &\equiv \mathsf{N}\varphi. \end{aligned}$$

3 The difference between past and future

There are 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 the behavior of a non-deterministic reactive system is, and what kinds 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 have already taken 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* [Zanardo and Carmo, 1993]. Some proposals (e.g. [Pinter and Wolper, 1984]) consider a non-determined past, also called "branching past", most typically through a definition like

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

(thus making past potentially infinite). We believe such a definition 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 formulae 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 [Laroussinie and Schnoebelen, 1995], 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. [Kupferman and Pnueli, 1995] is an important paper: it proposes CTL_{bp} and CTL_{lp}, two extensions of CTL with a branching and, respectively, with a linear (Ockhamist) past. CTL_{lp} is in fact our $PCTL$ from [Laroussinie and Schnoebelen, 1995] , without its "Now" combinator. In [Kupferman and Pnueli, 1995] the introduction of CTL_{bp} is motivated by theoretical problems (the comparison of its expressivity with CTL_{lp}). No example of smooth and natural properties relying on branching past is given, which only confirms our views that branching past is awkward for behavioral specifications.

4 $CTL + Past$ and verification

When it comes to verification, $PCTL$ has its pros and cons. On the positive side, writing temporal specifications is made easier in practice with $PCTL$ (see appendix). On the negative side, some verification problems become harder.

	Structure of past						
	Determined	Non-determ.	Finite	Infinite	Cumulative	Non-cumul.	
GPSS80, KVd83, LPZ85, Bar87, Zuc86, Gab89, Var88, MP92, MMKR94	•		•		•		Linear time
PW84, Wol89, Sti89, Sti92		•		•			Branching time
KP95		•	•				non-Ockhamist past
ZC93, Zan96	•			•	•		
HT87, BLY96	•		•			•	Branching time
LPS95, KP95	•		•		•		Ockhamist past
LS95	•		•		•	•	

Figure 2: The semantics of past in the literature

In this section we give precise complexity measures for the satisfiability and model-checking problems when $PCTL$ is concerned. We state these results for Kripke Structures *without fairness constraint* because any complexity result requires that we choose a description mechanism for fairness constraints. We simply picked the simplest choice, which also is the most informative when we look at *lower bounds* for complexity. Our results without fairness constraints easily extend to the more common kind of constraints such as repeated states *à la Buchi* or (weakly and strongly) fair transitions as in [Manna and Pnueli, 1992].

4.1 Satisfiability

The *satisfiability problem* asks whether a given temporal formula φ is satisfiable, that is, whether there exists a KS S s.t. $S \models \varphi$. *Finite satisfiability* asks whether a formula φ is satisfiable in a finite KS. Because $PCTL$ has the finite model property (see below) satisfiability and finite satisfiability coincide for this logic.

Theorem 4.1. *The satisfiability problem for $PCTL$ is EXPTIME-complete.*

Proof. Because $PCTL$ contains CTL for which satisfiability is already EXPTIME-complete, there only remains to prove membership in EXPTIME. This is our Prop. 4.4 (see below). □

The rest of this section proves membership in EXPTIME, our Proposition 4.4, by reusing and extending a similar proof for CTL_{lp} [Kupferman and Pnueli, 1995], and we show a finite model property for $PCTL$.

In a standard way, CTL, CTL_{lp} and $PCTL$ can be extended to $QCTL$, $QCTL_{lp}$ and $QPCTL$ where quantification over auxiliary propositions is allowed. $EQCTL$, $EQCTL_{lp}$ and $EQPCTL$ are the extensions where only outermost existential quantifications are allowed: e.g. $\exists p \exists q (\mathsf{A} a \mathsf{U} (p \wedge b \mathsf{S} q))$ is an $EQPCTL$ formula but $\forall p(\mathsf{A} a \mathsf{U} p)$ or $\mathsf{A} a \mathsf{U} \exists p.p$ are not.

When dealing with $EQPCTL$ it is sometimes useful to only consider acyclic Kripke structures, i.e. KS's where the underlying directed graph (Q_S, R_S) is a forest. In such acyclic Kripke structures, a state cannot appear twice in an execution, so that the state-valuations in an execution are independent and can be enriched (for new propositional variables introduced by quantification) independently. Observe that acyclic KS's are necessarily infinite because R_S is total. We write $\equiv^\star$ to denote equivalence restricted to acyclic structures: $\varphi \equiv^\star \psi$ if for all σ in all acyclic structures $\sigma \models \varphi$ iff $\sigma \models \psi$. Similarly, $\equiv_i^\star$ denotes initial equivalence over acyclic structures.

Lemma 4.2. **[Kupferman and Pnueli, 1995]** *For any $EQCTL_{lp}$ formula $\exists p_1 \ldots \exists p_n.\varphi$, one can build in linear-time a CTL formula $\bar{\varphi}$ and a set of new propositional variables $p'_1, \ldots, p'_k$ s.t. $(\exists p_1 \ldots \exists p_n.\varphi) \equiv_i^\star (\exists p'_1 \ldots \exists p'_k \exists p_1 \ldots \exists p_n.\bar{\varphi})$.*

The idea behind Lemma 4.2 is a standard renaming technique. We replace past-time subformulae in φ by new propositions and introduce auxiliary conjuncts defining the new propositions. Consider our earlier example $\exists p \exists q(\mathsf{A}a\mathsf{U}(p \wedge b\mathsf{S}q))$. Here $b\mathsf{S}q$ is replaced by a new proposition p' and a constraint on p'. Then

$$\exists p \exists q(\mathsf{A}a\mathsf{U}(p \wedge b\mathsf{S}q)) \equiv_i^\star \exists p' \exists p \exists q. \left(\begin{array}{ll} & (\mathsf{A}a\mathsf{U}(p \wedge p')) \\ \wedge & (p' \Leftrightarrow q) \\ \wedge & \mathsf{AG}(p' \Rightarrow \mathsf{AX}(p' \Leftrightarrow b \vee q)) \\ \wedge & \mathsf{AG}(\neg p' \Rightarrow \mathsf{AX}(p' \Leftrightarrow q)) \end{array} \right)$$

The constraint states that (1) at the root, we have $b\mathsf{S}q$ iff we have q, (2) every time we have $b\mathsf{S}q$ then the next states have $b\mathsf{S}q$ iff they have b or q, and (3) every time we do not have $b\mathsf{S}q$, the next states have $b\mathsf{S}q$ iff they have q.

Lemma 4.3. *For any φ in $EQPCTL$, one can build in linear-time an $EQCTL_{lp}$ formula $\tilde{\varphi}$ s.t. $\varphi \equiv^\star \tilde{\varphi}$.*

Proof. By induction over the number of N operators. If φ has no N then it is an $EQCTL_{lp}$ formula and we are done. Otherwise we can pick one of the innermost occurences of N and write φ under the form $\varphi[\mathsf{N}\psi]$ where ψ has no N. Lemma 4.2 gives a $\bar{\psi}$ in CTL and some fresh propositions $p'_1, \ldots, p'_k$ s.t. $\psi \equiv_i^\star \exists p'_1 \ldots \exists p'_k.\bar{\psi}$. Then, because $\bar{\psi}$ is pure-future, we have $\mathsf{N}\psi \equiv^\star \mathsf{N}(\exists p'_1 \ldots \exists p'_k.\bar{\psi}) \equiv \exists p'_1 \ldots \exists p'_k.\bar{\psi}$. Then we can use another fresh proposition $p'_{\bar{\psi}}$ to characterize states verifying $\bar{\psi}$ and we have : $\varphi[\mathsf{N}\psi] \equiv^\star \exists p'_1 \ldots \exists p'_k.\exists p'_{\bar{\psi}}(\varphi[p'_{\bar{\psi}}] \wedge \mathsf{AG}(p'_{\bar{\psi}} \Leftrightarrow \bar{\psi}))$. We now have one less N and apply the induction hypothesis on the $\varphi[p'_{\bar{\psi}}]$ part (which is smaller than ψ, hence the linear-time complexity). $\square$

Proposition 4.4. *Satisfiability for PCTL is in EXPTIME.*

Proof. Combining Lemmas 4.2 and 4.3 we see that for any φ in $EQPCTL$ there exists a $\exists p'_1 \ldots \exists p'_k \bar{\varphi}$ in $EQCTL$ s.t. $\varphi \equiv_i^\star \exists p'_1 \ldots \exists p'_k \bar{\varphi}$. Then φ is satisfiable iff $\exists p'_1 \ldots \exists p'_k \bar{\varphi}$ is satisfiable iff $\bar{\varphi}$ is satisfiable. Because $\bar{\varphi}$ is in CTL, because satisfiability for CTL is in EXPTIME, and because the construction of $\bar{\varphi}$ from φ is in linear-time, we deduce that satisfiability for $EQPTCL$, and hence for $PCTL$, is in EXPTIME. $\square$

Finally, we can observe that $\equiv_i$ and $\equiv_i^\star$ coincide for $EQPCTL$ formulae as a corollary of

Proposition 4.5. *$EQPCTL$ (and hence $PCTL$) have the finite model property.*

Proof. Assume $\exists p_1 \ldots \exists p_n.\varphi$ is satisfied by some infinite KS S. Then φ is satisfied by some infinite KS S' obtained from S by enriching the labeling function l_S.

Now there exists a ψ in CTL^* s.t. $\varphi \equiv_i \psi$ because $PCTL$ formulae can be expressed in CTL^* modulo initial equivalence [Laroussinie and Schnoebelen, 1995]. $S' \models \varphi$ entails $S' \models \psi$ (hence ψ is satisfiable). CTL^* has the finite model property so that there exists S'', a finite KS with $S'' \models \psi$ and $S'' \models \varphi$. By forgetting part of $l_{S''}$ we get a finite KS where $\exists p_1 \ldots \exists p_n.\varphi$ is satisfied. $\square$

4.2 Model-checking

The *model-checking problem* asks whether a given temporal formula φ holds in a given finite KS S, i.e. whether $S \models \varphi$. For CTL it is well-known that model-checking can be done in time $O(|S|\,.\,|\varphi|)$ and hence is in P.

Adding past-time constructs makes model-checking more complex. In fact very little needs to be added as we show. We follow [Emerson, 1990] and write $B(\mathsf{H}_1, \mathsf{H}_2, \ldots)$ for the branching-time logic where only the temporal combinators $\mathsf{H}_1, \mathsf{H}_2, \ldots$ are allowed (and where all future-time combinators are immediately under a path quantifier). So that CTL is $B(\mathsf{X}, \mathsf{U})$, CTL_{lp} is $B(\mathsf{X}, \mathsf{U}, \mathsf{X}^{-1}, \mathsf{S})$, and $PCTL$ is $B(\mathsf{X}, \mathsf{U}, \mathsf{X}^{-1}, \mathsf{S}, \mathsf{N})$. Furthermore, we write $B_r(\mathsf{H}_1, \mathsf{H}_2, \ldots)$ for the fragment of $B(\mathsf{H}_1, \mathsf{H}_2, \ldots)$ where one only allows formulae with no future-time combinator occuring in the scope of a past-time combinator. (Such restrictions have been used in linear-time logics, e.g. in [Manna and Pnueli, 1990] where it is proved that any $L(\mathsf{X}, \mathsf{U}, \mathsf{X}^{-1}, \mathsf{S})$-property can be expressed by a formula in $L_r(\mathsf{F}, \mathsf{X}^{-1}, \mathsf{S})$ with height at most 2 for the nesting of F's.)

Lemma 4.6. *Model-checking for PCTL is PSPACE-easy.*

Proof. We reduce model-checking for $PCTL$ to model-checking for CTL_{lp} which is in PSPACE (this was conjectured in [Kupferman and Pnueli, 1995], has later been proved by Kupferman, Pnueli and Vardi [Kupferman, 1998], and has not yet been published).

Consider φ in $PCTL$ and some KS S. We use induction over the number of N's in φ. If there are none, φ is a CTL_{lp} formula. Otherwise we can write φ in the form $\varphi[\mathsf{N}\psi]$ where ψ has no N. Because $\mathsf{N}\psi$ is a state formula (see [Laroussinie and Schnoebelen, 1995]) it is possible to consider it as a new proposition p and label every state of S accordingly: $p \in l(s) \overset{\text{def}}{\Leftrightarrow} S, s \models \psi$. Checking $\varphi[\mathsf{N}\psi]$ in S has been reduced to checking $\varphi[p]$ in a modified S. $\square$

Theorem 4.7. *Model-checking for* $B_r(\mathsf{F}^{-1}, \mathsf{F})$*,* $B_r(\mathsf{F}^{-1}, \mathsf{X})$*,* $B_r(\mathsf{X}^{-1}, \mathsf{F})$ *and* $B_r(\mathsf{X}^{-1}, \mathsf{X})$ *is PSPACE-complete.*

Proof. PSPACE-easiness is proved with lemma 4.6. For PSPACE-hardness we build on ideas from [Demri and Schnoebelen, 1998] and exhibit a simple transformation of QBF (Quantified Boolean Formulae) into our model-checking problems.

Let P an instance of QBF. W.l.o.g. we assume P has the form :

$$P \equiv \forall x_1 \, \exists x_2 \forall x_3 \ldots \exists x_n \overbrace{\bigwedge_{i=1}^{m} \bigvee_{j=1}^{k_i}}^{P_0} l_{i,j}$$

where n is an even integer and every $l_{i,j}$ is a propositional variable $x_{r(i,j)}$ or the negation $\neg x_{r(i,j)}$ of a propositional variable from $X = \{x_1, \ldots, x_n\}$. We note $\epsilon(i,j)$ the sign ($+$ or $-$) of $l_{i,j}$.

To such an instance P, we associate the Kripke structure S_P described in Fig. 3 where we use propositional variables from $\{x_1^+, x_1^-, \dots, A_0, \dots, L_1^1, \dots\}$.

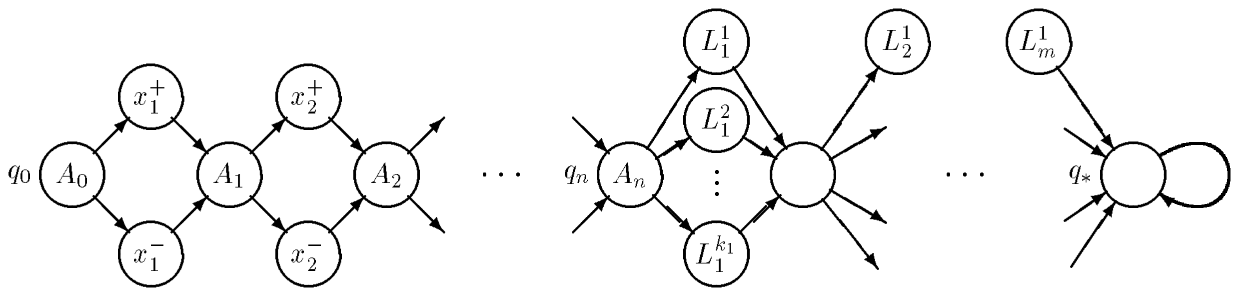

Figure 3: Structure S_P for an instance P of QBF

A computation π in S_P starts from q_0, visits q_n and ends up in q_*. To this computation, we associate v_π, a boolean valuation of the propositional variables $x_1, \dots, x_n$, in a canonical way: $v(x_i) \stackrel{\text{def}}{=} \top$ (resp. $\stackrel{\text{def}}{=} \bot$) iff $x_i^+ \in l_{S_P}(\pi(2i-1))$ (resp. $x_i^- \in l_{S_P}(\pi(2i-1))$). We say that v_π validates P_0 when, for every clause $\bigvee_j l_{i,j}$ in P_0, v_π yields true for at least one literal $l_{i,j}$.

This can be expressed in $PCTL$. Define

$$\varphi_{ok} \stackrel{\text{def}}{=} \bigwedge_{i=1}^{m} \bigwedge_{j=1}^{k_i} \Big(L_i^j \;\Rightarrow\; \mathsf{F}^{-1} x_{r(i,j)}^{\epsilon(i,j)} \Big)$$

Then v_π validates P_0 iff $\pi(0), \dots, \pi(2n) \models \mathsf{EG}\,\varphi_{ok}$, so that P is valid iff $S_P, q_0 \models \varphi_P$ with:

$$\varphi_P \stackrel{\text{def}}{=} \overbrace{\mathsf{AF}\,(A_1}^{\forall x_1} \wedge \overbrace{\mathsf{EF}\,(A_2}^{\exists x_2} \wedge \overbrace{\mathsf{AF}\,(A_3}^{\forall x_3} \wedge \cdots \overbrace{\mathsf{EF}\,(A_n}^{\exists x_n} \wedge \mathsf{EG}\,\varphi_{ok}) \cdots)))$$

This gives us a polynomial-time transformation of QBF into model-checking for $B_r(\mathsf{F}^{-1}, \mathsf{F})$, establishing our first PSPACE-hardness result.

The same idea can be used with X instead of F. Define:

$$\varphi'_P \stackrel{\text{def}}{=} \overbrace{\mathsf{AX}\,\mathsf{AX}}^{\forall x_1}\overbrace{\mathsf{EX}\,\mathsf{EX}}^{\exists x_2}\overbrace{\mathsf{AX}\,\mathsf{AX}}^{\forall x_3} \cdots \overbrace{\mathsf{EX}\,\mathsf{EX}}^{\exists x_n}\mathsf{EX}(\overbrace{\varphi_{ok} \wedge \mathsf{EXEX}(\varphi_{ok} \wedge \mathsf{EXEX}(\cdots \varphi_{ok})}^{m \text{ times } \varphi_{ok}} \cdots))$$

and then P is valid iff $S_P, q_0 \models \varphi'_P$. This gives PSPACE-hardness for $B_r(\mathsf{F}^{-1}, \mathsf{X})$. Moreover given the structure of S_P, we can replace F^{-1} by nested X^{-1} in φ_{ok}. Let φ'_{ok} be the following formula :

$$\varphi'_{ok} \stackrel{\text{def}}{=} \bigwedge_{i=1}^{m} \bigwedge_{j=1}^{k_i} \Big(L_i^j \;\Rightarrow\; \mathsf{X}^{-2i-2n+2r(i,j)} x_{r(i,j)}^{\epsilon(i,j)} \Big)$$

Using φ'_{ok} in φ_P and φ'_P directly gives a proof of PSPACE-hardness for $B_r(\mathsf{F}, \mathsf{X}^{-1})$ and $B_r(\mathsf{X}, \mathsf{X}^{-1})$. □

As direct consequence of lemma 4.6 and theorem 4.7, we have :

Corollary 4.8. *Model-checking for PCTL is PSPACE-complete.*

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

In view of Theorem 4.7, we advocate a *translation-based approach* for extensions of CTL [Laroussinie and Schnoebelen, 1995]. That is, we argue that, when possible, a convenient way to handle extensions of CTL in verification tools is to translate these extensions back into equivalent CTL formulae, so that the finely-tuned technology of CTL model-checkers can be reused without modification. A second 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 [Laroussinie and Schnoebelen, 1995] we showed how $CTL + \mathsf{F}^{-1} + \mathsf{N}$ can be translated into CTL. Other extensions of CTL for enhanced practical expressivity have been proposed (e.g. CTL^{+} in [Emerson and Halpern, 1985] or CTL^2 in [Kupferman and Grumberg, 1996]) but these works did not argue for a translation-based approach to model-checking.

In this and 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*. 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:

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

Clearly, in view of Proposition 2.4, we are looking for translations modulo initial equivalence.

Definition 5.1. 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 and reasonably efficient method for the translation.)

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

Theorem 5.2. [Laroussinie and Schnoebelen, 1995]
$B(\mathsf{X},\mathsf{U},\mathsf{S})$*, i.e.* $CTL + \mathsf{S}$*, cannot be translated into* CTL*.*
$B(\mathsf{X},\mathsf{U},\mathsf{X}^{-1})$*, i.e.* $CTL + \mathsf{X}^{-1}$*, 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. We already know that

Theorem 5.3. [Laroussinie and Schnoebelen, 1995]
$B(\mathsf{X},\mathsf{U},\mathsf{F}^{-1},\mathsf{N})$ *(i.e.* $CTL + \mathsf{F}^{-1} + \mathsf{N}$*) can be effectively translated into* CTL*.*

It is not clear what is the full practical range of Theorem 5.3. Indeed, we started developing our Lift specification (see appendix) precisely with the idea of testing whether writing specifications is much simplified by the use of $CTL + \mathsf{F}^{-1} + \mathsf{N}$, or if F^{-1} is still far from $\mathsf{S} + \mathsf{X}^{-1}$.

The main theoretical result of this paper is a non-trivial extension of Theorem 5.3 to a larger fragment of $PCTL$. This extension has to respect the limits already given by Theorem 5.2. The inspiration for this extension was given by the Lift specification which helped us notice a precisely delineated fragment of $PCTL$ that (1) supports the Lift specification, and (2) can be translated into CTL.

We now define $NCTL$, the aforementioned fragment of $PCTL$:

Definition 5.4. The logic $NCTL$

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

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

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

Remark 5.5. The Lift specification in the appendix only needs $NCTL$ formulae.

Now we have the following result:

Theorem 5.6. *$NCTL$ can be (effectively) translated into CTL*

This is the main theorem. The next section is devoted to its proof.

6 A separation theorem for $NCTL$

Definition 6.1. A PTL formula φ is *separated* if no past-time combinator occurs in the scope of a future-time combinator.

Example 6.2. A pure-future formula is separated, a pure-past formula is separated, $\mathsf{F}^{-1}(\mathsf{AX}a)$ is separated, $\mathsf{EF}(\mathsf{X}^{-1}a)$ is not.

Our definition is different from the simpler classical notion used in e.g. [Gabbay, 1989] where a formula is separated iff it is a boolean combination of pure-future and pure-past formula (with this definition, $\mathsf{F}^{-1}(\mathsf{X}a)$ is not "separated"). While the classical notion does not allow the separation for the branching-time logics [Laroussinie and Schnoebelen, 1995], the following lemma shows that our more general notion is sufficient for our translation purpose :

Lemma 6.3. *Any separated $PCTL$ formula is initially equivalent to a CTL formula.*

Proof. The proof is by structural induction on the formula φ to be separated.

1. $\varphi = \mathsf{EX}\psi$, $\mathsf{E}\psi\mathsf{U}\psi'$, or $\mathsf{A}\psi\mathsf{U}\psi'$: The formula ψ and ψ' are pure-future (no past-time operator can occur inside because φ is separated), then φ is in CTL.

2. $\varphi = \mathsf{X}^{-1}\psi$: then $\varphi \equiv_i \bot$.

3. $\varphi = \psi\mathsf{S}\psi'$: we have $\varphi \equiv_i \psi'$ and, by ind. hyp. there is $\bar{\psi}'$ in CTL s.t. $\psi' \equiv_i \bar{\psi}'$, then we have $\varphi \equiv_i \bar{\psi}'$.

4. $\varphi = \mathsf{N}\psi$: By ind. hyp. there exists $\bar{\psi} \in CTL$ s.t. $\psi \equiv_i \bar{\psi}$ and then $\mathsf{N}\psi \equiv \bar{\psi}$ (because $\bar{\psi}$ is pure-future). Finally we deduce $\varphi \equiv_i \bar{\psi}$.

5. Remaining cases: trivial with the induction hypothesis.

□

Theorem 6.4. (Separation theorem) *Any NCTL formula is equivalent to a separated NCTL formula.*

The rest of this section is devoted to a step-by-step proof. In fact we display an effective method computing a separated equivalent of any *NCTL* formula.

We use a set of rewrite rules for this. Read from left to right, rules R1 to R8 (Fig. 4) handle a $\mathsf{X}^{-1}x$ or a $x\mathsf{S}y$ subformula under the immediate scope of a future combinator.

$$
\begin{array}{ll}
\text{(R1)} & \mathsf{E}\varphi\mathsf{U}(\alpha \wedge \mathsf{X}^{-1}x) \equiv \Big(\alpha \wedge \mathsf{X}^{-1}x\Big) \vee \Big(\mathsf{E}\varphi\mathsf{U}(\varphi \wedge x \wedge \mathsf{EX}\alpha)\Big) \\
\text{(R2)} & \mathsf{E}\varphi\mathsf{U}(\alpha \wedge \neg\mathsf{X}^{-1}x) \equiv \Big(\alpha \wedge \neg\mathsf{X}^{-1}x\Big) \vee \Big(\mathsf{E}\varphi\mathsf{U}(\varphi \wedge \neg x \wedge \mathsf{EX}\alpha)\Big) \\
\text{(R3)} & \mathsf{EX}(\alpha \wedge \mathsf{X}^{-1}x) \equiv x \wedge \mathsf{EX}\alpha \\
\text{(R4)} & \mathsf{EX}(\alpha \wedge \neg\mathsf{X}^{-1}x) \equiv \neg x \wedge \mathsf{EX}\alpha \\
\text{(R5)} & \mathsf{E}\varphi\mathsf{U}(\alpha \wedge x\mathsf{S}y) \\
 & \equiv \mathsf{E}\varphi\mathsf{U}(\alpha \wedge y) \vee \mathsf{E}\varphi\mathsf{U}\Big(\varphi \wedge y \wedge \mathsf{EXE}(\varphi \wedge x)\mathsf{U}(\alpha \wedge x)\Big) \\
 & \quad \vee\Big(x\mathsf{S}y \wedge \Big(\alpha \vee \mathsf{E}(\varphi \wedge x)\mathsf{U}(\alpha \wedge x)\Big)\Big) \\
\text{(R6)} & \mathsf{E}\varphi\mathsf{U}(\alpha \wedge \neg(x\mathsf{S}y)) \\
 & \equiv \mathsf{E}\varphi\mathsf{U}(\alpha \wedge \neg x \wedge \neg y) \vee \mathsf{E}\varphi\mathsf{U}\Big(\varphi \wedge \neg x \wedge \neg y \wedge \mathsf{E}(\varphi \wedge \neg y)\mathsf{U}(\alpha \wedge \neg y)\Big) \\
 & \quad \vee\Big(\neg(x\mathsf{S}y) \wedge \Big(\alpha \vee \mathsf{E}(\varphi \wedge \neg y)\mathsf{U}(\alpha \wedge \neg y)\Big)\Big) \\
\text{(R7)} & \mathsf{EX}(\alpha \wedge x\mathsf{S}y) \equiv \mathsf{EX}(\alpha \wedge y) \vee \Big(x\mathsf{S}y \wedge \mathsf{EX}(\alpha \wedge x)\Big) \\
\text{(R8)} & \mathsf{EX}(\alpha \wedge \neg(x\mathsf{S}y)) \equiv \mathsf{EX}(\alpha \wedge \neg x \wedge \neg y) \vee \Big(\neg(x\mathsf{S}y) \wedge \mathsf{EX}(\alpha \wedge \neg y)\Big)
\end{array}
$$

Figure 4: Rules to extract S and X^{-1} from the scope of future combinators.

Rules R9 to R12 (Fig. 5) handle $\mathsf{F}^{-1}x$ subformulae. In essence, they already appeared in [Laroussinie and Schnoebelen, 1995]. The $\mathsf{F}^{-1}x$ subformulae are simpler to deal with, but, in the *NCTL* framework, they have to be dealt with in more contexts.

6.1 Soundness of the rules

Lemma 6.5. (Soundness) *Rules R1 to R12 are correct for FKS's, i.e. the equivalences hold for any PCTL formulae φ, x, y, α, β, γ, α', β' and γ'.*

Proof. As an example, we prove the soundness of R5 and let the reader verify the soundness of the other rules.
($\Rightarrow$): Assume $\sigma \models \mathsf{E}(\varphi\mathsf{U}(\alpha \wedge x\mathsf{S}y))$ and let n denote $|\sigma|$. There exists a $\pi \in \Pi(\sigma)$ and an $n' \geq n$ s.t. $\pi_{|n'} \models \alpha \wedge x\mathsf{S}y$ and $\pi_{|n+i} \models \varphi$ for any $n \leq n+i < n'$. Then $\pi_{|n'} \models x\mathsf{S}y$ and there is a m with $0 \leq m \leq n'$ s.t. $\pi_{|m} \models y$ and $\pi_{|m'} \models x$ for all $m < m' \leq n'$.

$$
\begin{aligned}
&\text{(R9)} && \mathsf{E}\Big((\mathsf{F}^{-1}x \wedge \alpha) \vee (\neg\mathsf{F}^{-1}x \wedge \beta) \vee \gamma\Big)\mathsf{U}\Big((\mathsf{F}^{-1}x \wedge \alpha') \vee (\neg\mathsf{F}^{-1}x \wedge \beta') \vee \gamma'\Big)\\
& && \equiv\ \mathsf{F}^{-1}x\ \wedge\ \mathsf{E}(\alpha \vee \gamma)\mathsf{U}(\alpha' \vee \gamma')\\
& && \quad \vee\ \ \neg\mathsf{F}^{-1}x\ \wedge\ \mathsf{E}\Big(\neg x \wedge (\beta \vee \gamma)\Big)\mathsf{U}\Big(x \wedge \mathsf{E}(\alpha \vee \gamma)\mathsf{U}(\alpha' \vee \gamma')\Big)\\
& && \quad \vee\ \ \neg\mathsf{F}^{-1}x\ \wedge\ \mathsf{E}\Big(\neg x \wedge (\beta \vee \gamma)\Big)\mathsf{U}\Big(\neg x \wedge (\beta' \vee \gamma')\Big)\\
&\text{(R10)} && \mathsf{EG}\Big((\mathsf{F}^{-1}x \wedge \alpha) \vee (\neg\mathsf{F}^{-1}x \wedge \beta) \vee \gamma\Big)\\
& && \equiv\ \mathsf{F}^{-1}x\ \wedge\ \mathsf{EG}(\alpha \vee \gamma)\\
& && \quad \vee\ \ \neg\mathsf{F}^{-1}x\ \wedge\ \mathsf{E}\Big(\neg x \wedge (\beta \vee \gamma)\Big)\mathsf{U}\Big(x \wedge \mathsf{EG}(\alpha \vee \gamma)\Big)\\
& && \quad \vee\ \ \neg\mathsf{F}^{-1}x\ \wedge\ \mathsf{EG}\Big(\neg x \wedge (\beta \vee \gamma)\Big)\\
&\text{(R11)} && \mathsf{EX}(\alpha \wedge \mathsf{F}^{-1}x)\ \equiv\ \mathsf{EX}(\alpha \wedge x) \vee \Big(\mathsf{F}^{-1}x \wedge \mathsf{EX}\alpha\Big)\\
&\text{(R12)} && \mathsf{EX}(\alpha \wedge \neg\mathsf{F}^{-1}x)\ \equiv\ \neg\mathsf{F}^{-1}x \wedge \mathsf{EX}(\alpha \wedge \neg x)
\end{aligned}
$$

Figure 5: Rules to extract F^{-1} from the scope of future combinators.

We distinguish three cases depending on the position of m w.r.t n and n': **case 1** is when $m = n'$, **case 2:** is when $n < m < n'$, and **case 3:** is when $m \leq n$.

Figure 6 displays the three corresponding situations and shows how each case account for one subformula in the disjunction that form the right-hand side of R5.

($\Leftarrow$): Symmetrically, the three different situations depicted in Figure 6 exactly correspond to the temporal formulae that form the right-hand side disjunction. In all three cases we have $\sigma \models \mathsf{E}(\varphi\mathsf{U}(\alpha \wedge x\mathsf{S}y))$. □

6.2 Stability for *NCTL*

Soundness holds for the full of *PCTL* (and in fact for more general logics) but we only apply the rules to *NCTL* formulae. We need to show *stability*, which means that rules R1 to R12 only yield *NCTL* formulae if given *NCTL* formulae.

This is already true for all the usual purely boolean manipulation rules one uses (distributivity, disjunctive normalization, ...): they are stable for *NCTL*. Additionally we have the following

Lemma 6.6. *For all $\varphi, x, y, \alpha, \beta, \alpha', \beta'$ and γ, for all rule R1 to R12, the left hand-side of the rule is a NCTL formula (resp. limited formula) iff the right-hand side of the rule is a NCTL formula (resp. limited formula).*

Proof. By checking all 12 rules. As an example, let us consider rule R10 and stability for *NCTL*. (The remaining cases are left to the reader.)

Consider R10 and assume the left-hand side $\mathsf{EG}\Big((\mathsf{F}^{-1}x \wedge \alpha) \vee (\neg\mathsf{F}^{-1}x \wedge \beta) \vee \gamma\Big)$ is in *NCTL*. Then, because EG is in fact a short-hand for $\neg\mathsf{AF}\neg$, the argument $(\mathsf{F}^{-1}x \wedge \alpha) \vee (\neg\mathsf{F}^{-1}x \wedge \beta) \vee \gamma$ must be a limited *NCTL* formula, so that $x, \alpha, \beta, \gamma \in NCTL_{\text{lim}}$. Then it is easy to see that both sides of R10 are in $NCTL_{\text{lim}}$. Exactly the same reasoning applies if we start with the assumption that the right-hand side of R10 is in *NCTL*. □

In the remaining of this section, the explanations of how the transformations we apply to *NCTL* formulae (resp. $NCTL_{\text{lim}}$ formulae) indeed return *NCTL* formulae (resp. $NCTL_{\text{lim}}$ formulae) will

case 1: $m = n'$

$$
\begin{array}{cccccccc}
 & & & & & & y & \\
 & & & \varphi & & \varphi & \alpha & \\
\pi: & q_0 & \cdots & q_n & \cdots & q_{n'-1} & q_{n'} & \cdots
\end{array}
$$

$$\hookrightarrow \ \sigma \models \mathsf{E}\varphi\mathsf{U}(\alpha \wedge y)$$

case 2: $n < m < n'$

$$
\begin{array}{ccccccccccc}
 & & & & & y & x & & x & x & \\
 & & & \varphi & & \varphi & \varphi & & \varphi & \alpha & \\
\pi: & q_0 & \cdots & q_n & \cdots & q_m & q_{m+1} & \cdots & q_{n'-1} & q_{n'} & \cdots
\end{array}
$$

$$\hookrightarrow \ \sigma \models \mathsf{E}\varphi\mathsf{U}(\varphi \wedge y \wedge \mathsf{EXE}(\varphi \wedge x)\mathsf{U}(\alpha \wedge x))$$

case 3: $0 \le m \le n$

$$
\begin{array}{ccccccccccc}
 & & & & & \varphi & & \varphi & \alpha & \\
 & & & y & x & & x & & x & x & \\
\pi: & q_0 & \cdots & q_m & q_{m+1} & \cdots & q_n & \cdots & q_{n'-1} & q_{n'} & \cdots
\end{array}
$$

$$\hookrightarrow \ \sigma \models x\mathsf{S}y \wedge (\alpha \vee \mathsf{E}(\varphi \wedge x)\mathsf{U}(\alpha \wedge x))$$

Figure 6: Proving soundness of R5: three cases

be left implicit. The underlying reason will always be that we used rules from the R1-R12 set, simple boolean manipulations, stable replacements of formulae by equivalent formulae, induction hypothesis involving stable equivalence.

6.3 Separation strategy for *NCTL*

Because they cover all cases, we have, with R1 to R12, rules allowing the extraction of 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 the rules will always eventually separate past from future. E.g. consider R10: it extracts $\mathsf{F}^{-1}x$ from the scope of EG, but at the same time it (1) duplicates γ, and (2) 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 describe how a precise strategy ensures eventual separation. This requires some notations. Square brackets with variables will be used to denote contexts. E.g. $\varphi[x]$ is a context, yielding a *PCTL* formula when x is replaced by some *PCTL* formula. A key point is that we allow multiple appearances of x inside φ. We use this to collect multiple copies of duplicated subformulae. E.g. $\varphi[x]$ may be $\mathsf{E}x \wedge \psi_1\mathsf{U}x \wedge \psi_2$. Then $\varphi[\mathsf{EX}a]$ is $\mathsf{E}(\mathsf{EX}a) \wedge \psi_1\mathsf{U}(\mathsf{EX}a) \wedge \psi_2$. Our equivalences are extended to contexts: $\varphi_1[x, y] \equiv \varphi_2[x, y]$ iff $\varphi_1[\psi, \psi'] \equiv \varphi_2[\psi, \psi']$ for all *PCTL* formulae ψ, ψ'.

Lemma 6.7. *Let $f[x]$ be a CTL context s.t. $f[x_1\mathsf{S}x_2]$ is a NCTL context. Then there exists a CTL context $f'[x_1, x_2, x_3]$ s.t. $f[x_1\mathsf{S}x_2] \equiv f'[x_1, x_2, x_1\mathsf{S}x_2]$ and $f'[x_1, x_2, x_1\mathsf{S}x_2]$ is a separated context.*

Proof. The proof is by structural induction on $f[x]$. By assumption, there is no past-time construct in $f[x]$. We have four basic situations:

1. **$f[x]$ is some $\mathsf{A}\varphi[x]\mathsf{U}\psi[x]$:** then the assumption assumed that $f[x_1\mathsf{S}x_2]$ is a *NCTL* context implies that, in fact, x cannot possibly occur in f so that $f' \stackrel{\text{def}}{=} f$ is enough.

2. **$f[x]$ is some $\mathsf{E}\varphi[x]\mathsf{U}\psi[x]$:** we assumed $f[x_1\mathsf{S}x_2]$ is a *NCTL* context, thus x does not occur in φ, which is then a *CTL* formula.

   By induction hypothesis, $\psi[x_1\mathsf{S}x_2]$ 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. Using boolean manipulations, we can group all the occurrences of $x_1\mathsf{S}x_2$ and obtain

   $$\psi'[x_1\mathsf{S}x_2] \equiv (\alpha \wedge x_1\mathsf{S}x_2) \vee (\beta \wedge \neg(x_1\mathsf{S}x_2)) \vee \gamma$$

   where α, β and γ are pure-future. Then, using $\mathsf{E\_U}$ distributivity

   $$\mathsf{E}g\mathsf{U}(h \vee h') \equiv (\mathsf{E}g\mathsf{U}h) \vee (\mathsf{E}g\mathsf{U}h') \tag{D1}$$

   we get

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

   Here rules R5 and R6 are enough to extract $x_1\mathsf{S}x_2$ from the scope of the U's and we get the required $f'[x_1, x_2, x_3]$.

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

   $$\mathsf{EX}(h \vee h') \equiv \mathsf{EX}h \;\vee\; \mathsf{EX}h' \tag{D2}$$

   we end up with

   $$f[x_1\mathsf{S}x_2] \equiv \mathsf{EX}(\alpha \wedge x_1\mathsf{S}x_2) \vee \mathsf{EX}(\beta \wedge \neg(x_1\mathsf{S}x_2)) \vee \mathsf{EX}\gamma \tag{2}$$

   where we use rules R7 and R8 and get the required $f'[x_1, x_2, x_3]$.

4. **Remaining cases:** f may be some $\neg\varphi$ or some $\varphi \wedge \psi$ or some a. These are easy with the induction hypothesis.

□

Lemma 6.8. *Let $f[x]$ be a CTL context s.t. $f[\mathsf{F}^{-1}x_1]$ is a NCTL context. Then there exists a CTL context $f'[x_1, x_2]$ s.t. $f[\mathsf{F}^{-1}x_1] \equiv f'[x_1, \mathsf{F}^{-1}x_1]$ and $f'[x_1, \mathsf{F}^{-1}x_1]$ is a separated context.*

Proof. The proof is by structural induction on $f[x]$. By assumption, there is no past-time construct in $f[x]$. We have four basic situations:

1. **$f[x]$ is some $\mathsf{E}\varphi[x]\mathsf{U}\psi[x]$:** $f[\mathsf{F}^{-1}x_1]$ is always a $NCTL$ context and then x may occur in both φ and ψ. By induction hypothesis, $\varphi[\mathsf{F}^{-1}x_1]$ 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 group the occurrences of $\mathsf{F}^{-1}x_1$ and obtain:
$$f[\mathsf{F}^{-1}x_1] \equiv \mathsf{E}\Big((\mathsf{F}^{-1}x_1 \wedge \alpha) \vee (\neg\mathsf{F}^{-1}x_1 \wedge \beta) \vee \gamma\Big)\mathsf{U}\Big((\mathsf{F}^{-1}x_1 \wedge \alpha') \vee (\neg\mathsf{F}^{-1}x_1 \wedge \beta') \vee \gamma'\Big) \quad (3)$$
where $\alpha, \beta, \gamma, \alpha', \beta'$ and γ' are pure-future. Then we use rule R9 and extract all occurrences of $\mathsf{F}^{-1}x_1$ from the scope of $\mathsf{E\_U}$, yielding the required $f'[x_1, \mathsf{F}^{-1}x_1]$.
2. **$f[x]$ is some $\mathsf{EX}\varphi[x]$:** We proceed similarly. Using the induction hypothesis (and some boolean manipulations) $\varphi[\mathsf{F}^{-1}x_1]$ is equivalent to some $(\mathsf{F}^{-1}x_1 \wedge \alpha') \vee (\neg\mathsf{F}^{-1}x_1 \wedge \beta') \vee \gamma'$. so that we can use D2, R11 and R12 and get the required f'.
3. **$f[x]$ is some $\mathsf{EG}\varphi[x]$:** We proceed similarly and get
$$f[\mathsf{F}^{-1}x_1] \equiv \mathsf{EG}\Big((\alpha \wedge \mathsf{F}^{-1}x_1) \vee (\beta \wedge \neg\mathsf{F}^{-1}x_1) \vee \gamma\Big) \quad (4)$$
Then we only need rule R10.
4. **Remaining cases:** Finally, the other cases are obvious, or can be reduced to earlier cases thanks to $\mathsf{A}g\mathsf{U}h \equiv \neg\mathsf{EG}\neg h \ \wedge \ \neg\Big(\mathsf{E}\neg h\mathsf{U}\neg g \wedge \neg h\Big)$.

□

Lemma 6.9. *Let $f[x]$ be a CTL context s.t. $f[\mathsf{X}^{-1}x_1]$ is a $NCTL$ context. Then there exists a CTL context $f'[x_1, x_2]$ s.t. $f[\mathsf{X}^{-1}x_1] \equiv f'[x_1, \mathsf{X}^{-1}x_1]$ and $f'[x_1, \mathsf{X}^{-1}x_1]$ is a separated context.*

Proof. We follow exactly the plan of the proof for Lemma 6.8. The assumption that $f[\mathsf{X}^{-1}x_1]$ is a $NCTL$ context leave us with even less cases to consider. The $\mathsf{E}\varphi[x]\mathsf{U}\psi[x]$ is simpler and only needs R1, R2 and D1. The $\mathsf{EX}\varphi[x]$ case only needs R3, R4 and D2. There is no EG or $\mathsf{A\_U}$ case. □

A *simple past context* contains just one past combinator and the required variables. Hence such a context has the form $y\mathsf{S}z$, $\mathsf{F}^{-1}y$ or $\mathsf{X}^{-1}y$.

Lemma 6.10. *Assume $f[x_1, \ldots, x_n]$ is a CTL context and $g_1, \ldots, g_n$ are simple past contexts, of the form $y_i\mathsf{S}z_i$, $\mathsf{F}^{-1}y_i$ or $\mathsf{X}^{-1}y_i$ $(i = 1, \ldots, n)$. If $f[g_1, \ldots, g_n]$ is a NCTL context, then there exists a CTL context $f'[y_1, z_1, u_1, \ldots, y_n, z_n, u_n]$ s.t. $f[g_1, \ldots, g_n] \equiv f'[y_1, z_1, g_1, \ldots, y_n, z_n, g_n]$ and $f'[y_1, z_1, g_1, \ldots, y_n, z_n, g_n]$ is a separated context.*

Proof. By induction on n. There is nothing to prove for the base case $n = 0$. Consider then $f[x_1, \ldots, x_{n+1}]$ and the simple past context g_{n+1}. $f[x_1, \ldots, x_n, g_{n+1}]$ is a $NCTL$ context. Depending on the form of g_{n+1}, we use Lemma 6.7 or 6.8 or 6.9 to get a CTL context $f''[x_1, \ldots, x_n, y_{n+1}, z_{n+1}, u_{n+1}]$ s.t. $f[x_1, \ldots, x_n, g_{n+1}] \equiv f''[x_1, \ldots, x_n, y_{n+1}, z_{n+1}, g_{n+1}]$ and $f''[x_1, \ldots, x_n, y_{n+1}, z_{n+1}, g_{n+1}]$ is separated.

Because g_{n+1} has a past-combinator at the root, all its occurences in the separated $f''[x_1, \ldots, x_n, y_{n+1}, z_{n+1}, g_{n+1}]$ are under boolean combinators only. Then $f''[x_1, \ldots, x_n, y_{n+1}, z_{n+1}, g_{n+1}]$ is a boolean combination of g_{n+1}'s and CTL contexts of the form $h_k[x_1, \ldots, x_n, y_{n+1}, z_{n+1}]$. The contexts $h_k[g_1, \ldots, g_n, y_{n+1}, z_{n+1}]$ can be separated by induction hypothesis and replacing the corresponding separated contexts into the boolean combination concludes the proof. □

Lemma 6.11. *Let $f[x_1, \dots, x_n]$ be a CTL context and $\psi_1^-, \dots, \psi_n^-$ be pure-past NCTL formulae without* N. *If $f[\psi_1^-, \dots, \psi_n^-]$ is a NCTL formula, then it is equivalent to a separated NCTL formula.*

Proof. The proof is by induction on the maximum temporal depth (maximum number of nested temporal combinators) of the ψ_i^-'s. W.l.o.g. we can suppose that the ψ_i^-'s have a past combinator at the root, i.e. $\psi_i^- = \mathsf{X}^{-1}\varphi_i$ or $\mathsf{F}^{-1}\varphi_i$ or $\varphi_i \mathsf{S} \varphi_i'$ (this may require changing f and n). Then every ψ_i^- is of the form $g_i[\varphi_i, \varphi_i']$ with g_i a simple past context (as in Lemma 6.10). Now $f[\psi_1^-, \dots, \psi_n^-]$ is $f[g_1[\varphi_1, \varphi_1'], \dots, g_n[\varphi_n, \varphi_n']]$. By Lemma 6.10, there exists a CTL context $f''[y_1, z_1, u_1, \dots, y_n, z_n, u_n]$ s.t. $f[g_1, \dots, g_n]$ is equivalent to the separated context $f''[y_1, z_1, g_1, \dots, y_n, z_n, g_n]$. This entails that $f[\psi_1^-, \dots, \psi_n^-]$ is equivalent to $f''[\varphi_1, \varphi_1', \psi_1^-, \dots, \varphi_n, \varphi_n', \psi_n^-]$, a formula in which the ψ_i^-'s only occur in the scope of boolean combinators and where there only remains to separate pure-past subformulae of lower temporal depth. We do this with the induction hypothesis. □

Lemma 6.12. *Let $f[x_1, \dots, x_n]$ be a CTL context and $\psi_1, \dots, \psi_n$ be separated NCTL formulae without* N. *If $f[\psi_1, \dots, \psi_n]$ is a NCTL formula, then it is equivalent to a separated NCTL 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 $g_i^-[x_1, \dots, x_{m_i}]$. Lemma 6.11 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 6.13. *Any NCTL formula is equivalent to a separated NCTL formula without* N.

Proof. The proof is by structural induction on the formula φ to be separated.
1 - φ has the form $\psi \wedge \psi'$, or $\neg\psi$, or $\psi \mathsf{S} \psi'$, or $\mathsf{X}^{-1}\psi$: In φ we replace ψ and ψ' by the equivalent separated formulae without N that must exist by induction hypothesis. The result is an equivalent formula which is separated because φ does not have a future-time construct at its root.
2 - φ has the forme $\mathsf{EX}\psi$, or $\mathsf{E}\psi\mathsf{U}\psi'$, or $\mathsf{A}\psi\mathsf{U}\psi'$: The induction hypothesis and Lemma 6.12 allow to conclude.
3 - φ has the form $\mathsf{N}\psi$: By induction hypothesis ψ is equivalent to a separated ψ' without N. Lemma 6.3 says that there exists a CTL formula $\bar{\psi}'$ initially equivalent to ψ'. Now $\varphi = \mathsf{N}\psi \equiv \bar{\psi}'$. □

Now we can prove Theorem 5.6. Given some $NCTL$ formula φ, we use Lemma 6.13 and translate φ into a separated formula. Then we use Lemma 6.3 to get an initally equivalent CTL formula. For example, we have:

$$
\begin{array}{lll}
 & \mathsf{E}\, a\, \mathsf{U}\, (b \wedge c \mathsf{S} d) & \text{a } NCTL \text{ formula} \\[1em]
\equiv & \left.\begin{array}{l} c\mathsf{S}d \ \wedge\ \mathsf{E}\,(a \wedge c)\,\mathsf{U}\,(b \wedge c) \\ \vee\ \ \mathsf{E}\, a\, \mathsf{U} \Big(a \wedge d \wedge \mathsf{EX}\, \mathsf{E}(a \wedge c)\mathsf{U}(b \wedge c) \Big) \\ \vee\ \ \mathsf{E}\, a\, \mathsf{U}\, (b \wedge d) \end{array}\right\} & \text{a separated } NCTL \text{ formula} \\[2em]
\equiv_i & \left.\begin{array}{l} \mathsf{E}\, a\, \mathsf{U} \Big(a \wedge d \wedge \mathsf{EX}\, \mathsf{E}(a \wedge c)\mathsf{U}(b \wedge c) \Big) \\ \vee\ \ \mathsf{E}\, a\, \mathsf{U}\, (b \wedge d) \end{array}\right\} & \text{a } CTL \text{ formula}
\end{array}
$$

A consequence of Theorem 5.6 is that all formulae used in the Lift specification can be automatically translated into (initially) equivalent CTL formulae 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 6.14. Theorem 5.6 can be extended to $NCTL^+$, an extension of $NCTL$ where boolean combinations of path-formulae are allowed under a path quantifier, exactly as CTL^+ is obtained by extending CTL [Emerson, 1990].

It can further be extended to $NECTL^+$, an extension of $NCTL^+$ where the $\mathsf{E}\overset{\infty}{\mathsf{F}}$ combinator is allowed. This time, the translation is 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 considering $PCTL$, an Ockhamist extension of CTL. We investigated the feasibility of verification with $PCTL$ and showed that model-checking is PSPACE-complete.

We showed that $PCTL$ is well-suited for practical specification through an extensive example: the specification of a lift system. 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 $NCTL$ our earlier work on $CTL + \mathsf{F}^{-1}$.

An important question is the complexity of the translation : In the worst cases, our translation algorithm may induce combinatorial explosions, even with limited temporal height [Laroussinie and Schnoebelen, 1995]. As far as we know, informative lower bounds on the size of the resulting formulae (rather than on the time-complexity of a given translation algorithm) are not known, even in the linear-time framework of [Gabbay, 1989], and the same problem is still open with the much simpler translation from CTL^+ to CTL. Clearly, in view of Theorem 4.7, effective translations must be PSPACE-hard even for very small fragments. This does not mean that the translation approach is not feasible: a CTL translation of the Lift example does not even require doubling the size of the formulae.

However, only an actual implementation of Theorem 5.6 will demonstrate feasibility. 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

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

[Bouajjani et al., 1996] Bouajjani, A., Lakhnech, Y., and Yovine, S. (1996). 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.

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

[Clarke et al., 1994] Clarke, E., Grumberg, O., and Long, D. (1994). 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.

[Clarke et al., 1997] Clarke, E. M., Grumberg, O., and Hamaguchi, K. (1997). Another look at LTL model checking. *Formal Methods in System Design*, 10(1):47–71.

[Demri and Schnoebelen, 1998] Demri, S. and Schnoebelen, P. (1998). The complexity of propositional linear temporal logics in simple cases (extended abstract). In *Proc. 15th Ann. Symp. Theoretical Aspects of Computer Science (STACS'98), Paris, France, Feb. 1998*, volume 1373 of *Lecture Notes in Computer Science*, pages 61–72. Springer.

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

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

[Gabbay, 1989] Gabbay, D. (1989). 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.

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

[Hafer and Thomas, 1987] Hafer, T. and Thomas, W. (1987). 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.

[Hale, 1989] Hale, R. (1989). 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.

[Hennessy and Stirling, 1985] Hennessy, M. and Stirling, C. (1985). The power of the future perfect in program logics. *Information and Control*, 67(1–3):23–52.

[Kesten et al., 1993] Kesten, Y., Manna, Z., McGuire, H., and Pnueli, A. (1993). 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.

[Koymans et al., 1983] Koymans, R., Vytopil, J., and de Roever, W. P. (1983). 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.

[Kupferman, 1998] Kupferman, O. (1998). Personal communication.

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

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

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

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

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

[Manna and Pnueli, 1989] Manna, Z. and Pnueli, A. (1989). 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.

[Manna and Pnueli, 1990] Manna, Z. and Pnueli, A. (1990). A hierarchy of temporal properties. In *Proc. 9th ACM Symp. Principles of Distributed Computing (PODC'90), Quebec City, Canada, Aug. 1990*, pages 377–408.

[Manna and Pnueli, 1992] Manna, Z. and Pnueli, A. (1992). *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer.

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

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

[Pinter and Wolper, 1984] Pinter, S. S. and Wolper, P. (1984). 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.

[Pnueli, 1977] Pnueli, A. (1977). The temporal logic of programs. In *Proc. 18th IEEE Symp. Foundations of Computer Science (FOCS'77), Providence, RI, USA, Oct.-Nov. 1977*, pages 46–57.

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

[Raviv, 1997] Raviv, L. (1997). *LTL Symbolic Model Checking*. M.Sc. thesis, Weizmann Institute of Science, Rehovot, Israel.

[Stirling, 1989] Stirling, C. (1989). 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.

[Stirling, 1992] Stirling, C. (1992). Modal and temporal logics. In Abramsky, S., Gabbay, D. M., and Maibaum, T. S. E., editors, *Handbook of Logic in Computer Science, vol.2. Background : Computational Structures*, pages 477–563. Oxford Univ. Press.

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

[Wolper, 1989] Wolper, P. (1989). 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.

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

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

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

A Appendix : Specification of a lift system

We use the classical example of a lift system (see [Barringer, 1987] for a specification with LTL+Past and [Hale, 1989] for a specification with the Interval Temporal Logic) to experiment with the $PCTL$ and $NCTL$ logic. This example has been chosen because it is rich and realistic but still easy to understand.

Historically, we started this experiment to see whether temporal specifications are clearer and closer to our intuitions when written in $CTL + Past$. We ended up with formulae that could have been expressed in CTL. This prompted us to look whether we could draw a better line between Theorem 5.3 "$CTL + \mathsf{F}^{-1}$ is OK" and Theorem 5.2 "$PCTL$ is not". Inspecting the lift specification led us to the definition of $NCTL$: the whole specification could be naturally written in the $NCTL$ fragment.

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.

A.1 Informal specification

The informal specification we have in mind gathers several properties we list (by order of importance) in Figure 7. This is a variant of the description proposed in [Barringer, 1987]. P1-3 are

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).

Figure 7: An informal lift specification

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. Note that we do not consider physical properties of the lift like "the cabin is never visiting several floors at the same moment".

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*.

A.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:

- $\mathit{Open\_Door}_i$ $(i = 1, \ldots, n)$, 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,
- $\mathit{Call\_Light}_i$ (resp. $\mathit{Send\_Light}_i$) $(i = 1, \ldots, n)$, 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.

A.3 The formal specification

A.3.1 P1. Safe doors

This leaves no room for interpretation : $\bigwedge_{i=1}^{n} \mathsf{AG}(\mathit{Open\_Door}_i \;\Rightarrow\; \mathit{At}_i)$ (S1)

A.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:

$$\mathit{Servicing}_i \stackrel{\text{def}}{=} \mathit{At}_i \wedge \mathit{Open\_Door}_i \quad (i = 1, \ldots, n) \tag{D1}$$

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

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

$$\bigwedge_{i=1}^{n} \mathsf{AG}[\mathit{Send}_i \;\Rightarrow\; (\mathit{Servicing}_i \vee \mathit{Send\_Light}_i)] \tag{S2.2}$$

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

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

and write

$$\bigwedge_{i=1}^{n} \mathsf{AG}[\mathit{Call\_Light}_i \;\Rightarrow\; \mathsf{A}\,\mathit{Call\_Light}_i \mathsf{W} \mathit{Servicing}_i] \tag{S2.3}$$

$$\bigwedge_{i=1}^{n} \mathsf{AG}[\mathit{Send\_Light}_i \;\Rightarrow\; \mathsf{A}\,\mathit{Send\_Light}_i \mathsf{W} \mathit{Servicing}_i] \tag{S2.4}$$

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 the 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*".

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

A.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

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

A.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

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

and that the cabin is motionless unless there is some request

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

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

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

where

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

A.3.5 P5. Diligent service

We formalize "diligent service" as forbidding situations where

1. the cabin was servicing some floor i,
2. then it moved and went to service some other floor j,
3. therefore passing by some intermediary floor k,
4. 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

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

$$\bigwedge_{k=1}^{n} \mathsf{AG}\Big[\big(\mathit{At}_k \wedge \mathit{Pending\_Request}_k\big) \;\Rightarrow\; \mathsf{A}\;\mathit{Not\_Servicing}\,\mathsf{W}\,\mathit{Servicing}_k\Big] \tag{S5'}$$

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:

$$\begin{aligned}\mathit{At}_j\mathit{From}_i \stackrel{\text{def}}{=} & \mathit{Servicing}_j \;\wedge\; (\mathit{Servicing}_j \vee \mathit{Not\_Servicing})\;\mathsf{S}\;\mathit{Servicing}_i \\ & (i, j = 1, \ldots, n)\end{aligned} \tag{D7}$$

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

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

A.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.

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

A.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.

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

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

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.

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

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

A.4 Some lessons from the Lift experiment

We do not claim our informal specification from Fig. 7 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 behavioral properties in a propositional temporal logic like $CTL + Past$,
- The resulting formulae are hard to read and probably they require accompanying explanations or comments before they can 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. These sentences contain past-time constructs so that allowing past-time constructs in the logic was very convenient.