Timed Concurrent Constraint Programming: Decidability Results and their Application to LTL

Frank D. Valencia ⋆

Dept. of Information Technology, Uppsala University
Box 337 SE-751 05 Uppsala, Sweden
Email: frankv@it.uu.se   Fax: +46 18 511 925

Abstract The ntcc process calculus is a timed *concurrent constraint programming* (ccp) model equipped with a first-order *linear-temporal logic* (LTL) for expressing process specifications. A typical behavioral observation in ccp is the *strongest postcondition (sp)*. The ntcc sp denotes the set of all infinite output sequences that a given process can exhibit. The *verification problem* is then whether the sequences in the sp of a given process satisfy a given ntcc LTL formula.
This paper presents new positive decidability results for timed ccp as well as for LTL. In particular, we shall prove that the following problems are decidable: (1) The *sp equivalence* for the so-called *locally-independent* ntcc fragment; unlike other fragments for which similar results have been published, this fragment can specify *infinite-state systems*. (2) *Verification* for locally-independent processes and negation-free *first-order* formulae of the ntcc LTL. (3) *Implication* for such formulae. (4) *Satisfiability* for a first-order fragment of Manna and Pnueli's LTL. The purpose of the last result is to illustrate the applicability of ccp to well-established formalisms for concurrency.

1 Introduction

Concurrent constraint programming (ccp) [21] is a model of concurrency for systems in which agents interact with one another by telling and asking information in a shared medium, a so-called *store*. Timed ccp [18] extends ccp by allowing agents to be constrained by time requirements. Its distinctive feature is that it combines in one framework the operational and algebraic view based upon process calculi with a declarative view based upon *linear-temporal logic* (LTL). So, processes can be treated as computing agents, algebraic terms and LTL formulae. This allows timed ccp to benefit from the large body of techniques of well established theories used in the study of concurrency and, in particular, reactive computations. In fact, timed ccp has been studied extensively as a model for reactive systems [4, 8, 14–20, 23].

The ntcc process calculus [15] is a generalization of thetimed ccp model tcc [18]. The calculus can represent timed concepts such as unit delays, unbounded finite delays, time-outs, pre-emption, synchrony and asynchrony. Furthermore, ntcc is equipped with an LTL to specify timed properties and with an *inference system* for the verification problem (i.e., for proving whether a given process fulfills a given LTL specification).

⋆ This work was supported by the **PROFUNDIS** Project.

This paper presents new decidability results for infinite-state `ntcc` processes, the `ntcc` LTL (here called *constraint* LTL or **CLTL** for short), and for the standard first-order LTL (here called **LTL** for short) described by Manna and Pnueli [10]. The description and relevance of these results are outlined next:

- *On the sp equivalence and verification problem.* The strongest-postcondition (sp) behavior of a given `ntcc` process P denotes the set of all infinite sequences of outputs that P can exhibit. Thus, P fulfills a given specification (i.e., a **CLTL** formula) F iff each sequence in its sp satisfies F. In Section 3, we show that for a substantial fragment of `ntcc` and the negation-free first-order fragment of **CLTL**: (1) the sp equivalence is decidable and (2) the verification problem is decidable.
  - A noteworthy aspect of these two results is that the `ntcc` fragment above admits *infinite-state processes*. All other `ntcc` fragments for which similar results has been published [14, 16] are restricted to finite-state processes.
  - Another noteworthy aspect is that **CLTL** is first-order. Most first-order LTLs in computer science are not recursively axiomatizable let alone decidable [1].
- *On the* `ntcc` *LTL.* In Section 3 we prove that: (3) the validity of implication is decidable for the negation-free first-order fragment of **CLTL**.
  - As for Hoare logic, the `ntcc` inference system [15] mentioned above has the so-called consequence rule which queries an oracle about (**CLTL**) implication. This causes the completeness of the system to be relative to the capability of determining the validity of implication, thus making our third result of relevance to `ntcc`.
  - As a corollary of this result, we obtain the decidability of *satisfiability* for the negation-free first-order fragment of **CLTL**. This is relevant for specification purposes since, as remarked in [25], a specification is "interesting" only if it is satisfiable.
- *On the standard first-order LTL.* In Section 4 we prove that: (4) the satisfiability problem in **LTL** is decidable for all negation-free first-order formulae without rigid variables. This result is obtained from a reduction to **CLTL** satisfiability.
  - Since first-order **LTL** is not recursively axiomatizable [1], satisfiability is undecidable for the full language of **LTL**. Recent work [9] and also [11], however, have taken up the task of identifying first-order decidable fragments of **LTL**. Our fourth result contributes to this task.
  - The reduction from the standard **LTL** satisfiability to **CLTL** satisfiability also contributes to the understanding of the relationship between (timed) ccp and (temporal) classic logic.

In brief, this paper argues for timed ccp as a convenient framework for reactive systems by providing positive decidability results for behavior, specification and verification (1–3), and by illustrating its applicability to the well-established theory of LTL (4).

2 An Overview of `ntcc`

In `ntcc`, time is conceptually divided into *discrete intervals (or time units)*. In a particular timed interval, a process P gets an input (an item of information represented as a *constraint*) c from the environment, it executes with this input as the initial *store*, and when it reaches its resting point, it *outputs* the resulting store d to the environment. The resting point determines a residual process Q, which is then executed in the next time interval. In the rest of this section we shall briefly recall ntcc concepts given in [15].

Definition 1 (Constraint System). *A* constraint system (cs) *is a pair* (Σ, Δ) *where* Σ *is a signature of function and predicate symbols, and* Δ *is a decidable theory over* Σ *(i.e., a decidable set of sentences over* Σ *with a least one model).*

Given a cs (Σ, Δ), let $(\Sigma, \mathcal{V}, \mathcal{S})$ be its underlying first-order language, where $\mathcal{V}$ is a set of variables $x, y, \ldots$, and $\mathcal{S}$ is the set of logic symbols $\neg, \wedge, \vee, \Rightarrow, \exists, \forall, \mathtt{true}$ and $\mathtt{false}$. *Constraints* $c, d, \ldots$ are formulae over this first-order language. We say that c *entails* d in Δ, written $c \models d$, iff $c \Rightarrow d$ is true in all models of Δ. The relation $\models$, which is decidable by the definition of Δ, induces an equivalence $\approx$ given by $c \approx d$ iff $c \models d$ and $d \models c$. Henceforth, $\mathcal{C}$ denotes *the set of constraints under consideration* modulo $\approx$ in the underlying cs. Thus, we simply write $c = d$ iff $c \approx d$.

Definition (Processes, ***Proc*****).** *Processes* $P, Q, \ldots \in$ *Proc are built from constraints* $c \in \mathcal{C}$ *and variables* $x \in \mathcal{V}$ *in the underlying cs by:*

$$\begin{array}{rcl} P, Q, \ldots & ::= & \mathbf{tell}(c) \;\mid \sum_{i \in I} \mathbf{when}\; c_i\; \mathbf{do}\; P_i \mid P \parallel Q \;\mid (\mathbf{local}\, x)\, P \\ & \mid & \mathbf{next}\, P \mid \mathbf{unless}\; c\; \mathbf{next}\, P \;\mid \star\, P \;\mid !\, P \;\mid\; \mathbf{abort} \end{array}$$

Intuitively, $\mathbf{tell}(c)$ adds an item of information c to the store in the current time interval. The *guarded-choice summation* $\sum_{i \in I} \mathbf{when}\; c_i\; \mathbf{do}\; P_i$, where I is a finite set of indexes, chooses in the current time interval one of the P_i's whose c_i is entailed by the store. If no choice is possible, the summation is precluded from execution. We write $\mathbf{when}\; c_{i_1}\; \mathbf{do}\; P_{i_1} + \ldots + \mathbf{when}\; c_{i_n}\; \mathbf{do}\; P_{i_n}$ if $I = \{i_1, \ldots, i_n\}$ and, if no ambiguity arises, omit the "$\mathbf{when}\; c\; \mathbf{do}$" when $c = \mathtt{true}$. So, $\sum_{i \in I} P_i$ denotes the *blind-choice* $\sum_{i \in I} \mathbf{when}\; \mathtt{true}\; \mathbf{do}\; P_i$. We omit the "$\sum_{i \in I}$" if $|I| = 1$ and use $\mathbf{skip}$ for $\sum_{i \in \emptyset} P_i$.

The process $P \parallel Q$ represents the *parallel execution* of P and Q. $\prod_{i \in I} P_i$, where $I = \{i_1, \ldots, i_n\}$, denotes $((P_{i_1} \parallel P_{i_2}) \parallel \ldots P_{i_{n-1}}) \parallel P_{i_n}$. The process $(\mathbf{local}\, x)\, P$ declares an x *local* to P, and thus we say that it *binds* x in P. The *bound variables* $bv(Q)$ (*free variables* $fv(Q)$) are those with a bound (a not bound) occurrence in Q.

The *unit-delay* process $\mathbf{next}\, P$ executes P in the next time interval. The *time-out* $\mathbf{unless}\; c\; \mathbf{next}\, P$ is also a unit-delay, but P will be executed only if c cannot eventually be entailed by the store during the current time interval. Note that $\mathbf{next}\, P$ is not the same as $\mathbf{unless}\; \mathtt{false}\; \mathbf{next}\, P$ since an inconsistent store entails $\mathtt{false}$. We use $\mathbf{next}^n(P)$ for $\mathbf{next}(\mathbf{next}(\ldots(\mathbf{next}\, P)\ldots))$, where $\mathbf{next}$ is repeated n times.

The operator "$\star$" represents an *arbitrary (or unknown) but finite delay* (as "ϵ" in SCCS [12]) and allows asynchronous behavior across the time intervals. Intuitively, $\star\, P$ means $P + \mathbf{next}\, P + \mathbf{next}^2 P + \ldots$, i.e., an unbounded finite delay of P. The *replication* operator "!" is a delayed version of that of the π-calculus [13]: $!\, P$ means $P \parallel \mathbf{next}\, P \parallel \mathbf{next}^2 P \parallel \ldots$, i.e., unboundedly many copies of P but one at a time.

For technical and unification purposes we add to the syntax of ntcc in [15] the tcc process $\mathbf{abort}$ [18] which causes all interactions with the environment to cease.

SOS Semantics. The structural operational semantics (SOS) of `ntcc` considers *transitions* between process-store *configurations* of the form $\langle P, c\rangle$ with stores represented as constraints and processes quotiented by $\equiv$ below.

Definition 2 (Structural Congruence). *Let $\equiv$ be the smallest congruence satisfying: (1) $P \parallel \mathbf{skip} \equiv P$, (2) $P \parallel Q \equiv Q \parallel P$, and (3) $P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$. Extend $\equiv$ to configurations by decreeing that $\langle P, c\rangle \equiv \langle Q, c\rangle$ iff $P \equiv Q$.*

Following standard lines, we extend the syntax with a construct $\mathbf{local}\,(x, d)\,\mathbf{in}\,P$, to represent the evolution of a process of the form $\mathbf{local}\,x\,\mathbf{in}\,Q$, where d is the local information (or store) produced during this evolution. Initially d is "empty", so we regard $\mathbf{local}\,x\,\mathbf{in}\,P$ as $\mathbf{local}\,(x, \mathtt{true})\,\mathbf{in}\,P$.

The transitions of the SOS are given by the relations $\longrightarrow$ and $\Longrightarrow$ defined in Table 1. The *internal* transition $\langle P, d\rangle \longrightarrow \langle P', d'\rangle$ should be read as "P with store d reduces, in one internal step, to P' with store d' ". The *observable transition* $P \xRightarrow{(c,d)} R$ should be read as "P on input c, reduces in one *time unit* to R and outputs d". The observable transitions are obtained from terminating sequences of internal transitions.

$$\text{TELL}\ \frac{}{\langle \mathbf{tell}(c), d\rangle \longrightarrow \langle \mathbf{skip}, d \wedge c\rangle} \qquad \text{SUM}\ \frac{d \models c_j \ j \in I}{\langle \sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i, d\rangle \longrightarrow \langle P_j, d\rangle}$$

$$\text{PAR}\ \frac{\langle P, c\rangle \longrightarrow \langle P', d\rangle}{\langle P \parallel Q, c\rangle \longrightarrow \langle P' \parallel Q, d\rangle} \qquad \text{LOC}\ \frac{\langle P, c \wedge \exists_x d\rangle \longrightarrow \langle P', c'\rangle}{\langle (\mathbf{local}\, x, c)\, P, d\rangle \longrightarrow \langle (\mathbf{local}\, x, c')\, P', d \wedge \exists_x c'\rangle}$$

$$\text{UNL}\ \frac{}{\langle \mathbf{unless}\ c\ \mathbf{next}\ P, d\rangle \longrightarrow \langle \mathbf{skip}, d\rangle}\ \text{if } d \models c$$

$$\text{REP}\ \frac{}{\langle !\, P, d\rangle \longrightarrow \langle P \parallel \mathbf{next}\, !\, P, d\rangle} \qquad \text{STAR}\ \frac{}{\langle \star\, P, d\rangle \longrightarrow \langle \mathbf{next}^{\,n}\, P, d\rangle}\ \text{if } n \geq 0$$

$$\text{STR}\ \frac{\gamma_1 \longrightarrow \gamma_2}{\gamma_1' \longrightarrow \gamma_2'}\ \text{if } \gamma_1 \equiv \gamma_1' \text{ and } \gamma_2 \equiv \gamma_2' \qquad \text{ABORT}\ \frac{}{\langle \mathbf{abort}, d\rangle \longrightarrow \langle \mathbf{abort}, d\rangle}$$

$$\text{OBS}\ \frac{\langle P, c\rangle \longrightarrow^* \langle Q, d\rangle \not\longrightarrow}{P \xRightarrow{(c,d)} R}\ \text{if } R \equiv F(Q)$$

Table1. Rules for internal reduction $\longrightarrow$ (upper part) and observable reduction $\Longrightarrow$ (lower part). $\gamma \not\longrightarrow$ in OBS holds iff for no γ', $\gamma \longrightarrow \gamma'$. $\equiv$ and F are given in Definitions 2 and 3.

We shall only describe some of the rules of in Table 1 due to space restrictions (see [15] for further details). As clarified below, the seemingly missing cases for "next" and "unless" processes are given by OBS. The rule STAR specifies an arbitrary delay of P. REP says that $!P$ creates a copy of P and then persists in the next time unit. ABORT realizes the intuition of $\mathbf{abort}$ causing the interactions with the environment to cease by generating infinite sequences of internal transitions. We shall dwell a little upon the description of Rule LOC as it may seem somewhat complex. Let us consider the process

$$Q = (\mathbf{local}\, x, c)\, P$$

in Rule LOC. The global store is d and the local store is c. We distinguish between the *external* (corresponding to Q) and the *internal* point of view (corresponding to P). From the internal point of view, the information about x, possibly appearing in the "global" store d, cannot be observed. Thus, before reducing P we should first hide the information about x that Q may have in d. We can do this by existentially quantifying x in d. Similarly, from the external point of view, the observable information about x that the reduction of internal agent P may produce (i.e., c') cannot be observed. Thus we hide it by existentially quantifying x in c' before adding it to the global store corresponding to the evolution of Q. Additionally, we should make c' the new private store of the evolution of the internal process for its future reductions.

Rule OBS says that an observable transition from P labeled with (c, d) is obtained from a terminating sequence of internal transitions from $\langle P,\, c\rangle$ to a $\langle Q, d\rangle$. The process R to be executed in the next time interval is equivalent to $F(Q)$ (the "future" of Q). $F(Q)$ is obtained by removing from Q summations that did not trigger activity and any local information which has been stored in Q, and by "unfolding" the sub-terms within "next" and "unless" expressions.

Definition 3 (Future Function). *Let $F : Proc \rightharpoonup Proc$ be defined by*

$$F(Q) = \begin{cases} \mathbf{skip} & \text{if } Q = \sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ Q_i \\ F(Q_1) \parallel F(Q_2) & \text{if } Q = Q_1 \parallel Q_2 \\ (\mathbf{local}\, x)\, F(R) & \text{if } Q = (\mathbf{local}\, x, c)\, R \\ R & \text{if } Q = \mathbf{next}\, R \text{ or } Q = \mathbf{unless}\ c\ \mathbf{next}\, R \end{cases}$$

Remark 1. *F need no to be total since whenever we need to apply F to a Q (*OBS *in Table 1), every $\mathbf{tell}(c)$, $\mathbf{abort}$, $\star R$ and $!\, R$ in Q will occur within a "next" or "unless" expression.*

2.1 Observable Behavior: The Strongest Postcondition.

We now recall the notions of observable behavior for `ntcc` introduced in [16], in particular that of the *strongest postcondition* (sp), central to this paper.

Notation 1 *Throughout this paper $\mathcal{C}^\omega$ denotes the set of infinite (or ω) sequences of constraints in the underlying set of constraints $\mathcal{C}$. We use $\alpha, \alpha', \ldots$ to range over $\mathcal{C}^\omega$.*

Let $\alpha = c_1.c_2.\ldots$ and $\alpha' = c'_1.c'_2.\ldots$. Suppose that P exhibits the following infinite sequence of observable transitions (or *run*): $P = P_1 \xRightarrow{(c_1, c'_1)} P_2 \xRightarrow{(c_2, c'_2)} \ldots$. Given this run of P, we shall use the notation $P \xRightarrow{(\alpha, \alpha')}{}^{\omega}$.

IO and Output Behavior. Observe the above run of P. At the time unit i, the environment *inputs* c_i to P_i which then responds with an output c'_i. As observers, we can see that on α, P responds with α'. We refer to the set of all (α, α') such that $P \xRightarrow{(\alpha, \alpha')}{}^{\omega}$ as the *input-output (io) behavior* of P. Alternatively, if $\alpha = \mathtt{true}^\omega$, we interpret the run as an interaction among the parallel components in P *without the influence of any (external) environment*; as observers what we see is that P produces α on its own. We refer to the set of all α' such that $P \xRightarrow{(\mathtt{true}^\omega, \alpha')}{}^{\omega}$ as the *output* behavior of P.

Quiescent Sequences and SP. Another observation we can make of a process is its quiescent input sequences. These are sequences on input of which P can run without adding any information; we observe whether $\alpha = \alpha'$ whenever $P \xRightarrow{(\alpha,\alpha')}{}^{\omega}$.

In [15] it is shown that the set of quiescent sequences of a given P can be alternatively characterized as *the set of infinite sequences that P can possibly output under arbitrary environments*; the strongest postcondition (sp) of P.

Definition (SP and SP-Equivalence). *The* strongest postcondition *of P, $sp(P)$, is given by $sp(P)$ $=\{\alpha \mid P \xRightarrow{(\alpha',\alpha)}{}^{\omega}$ for some $\alpha'\}$ and its induced observational equivalence $\sim_{sp}$ is given by $P \sim_{sp} Q$ iff $sp(P) = sp(Q)$.*

2.2 LTL Specification and Verification

We now look at the `ntcc` LTL [15]. This particular LTL expresses properties over sequences of constraints and we shall refer to it as **CLTL**. We begin by giving the syntax of LTL formulae and then interpret them with the **CLTL** semantics.

Definition 4 (LTL Syntax). *The formulae $F, G, ... \in \mathcal{F}$ are built from constraints $c \in \mathcal{C}$ and variables $x \in \mathcal{V}$ in the underlying cs by:*

$$F, G, \ldots := c \mid \dot{\mathtt{true}} \mid \dot{\mathtt{false}} \mid F \dot{\wedge} G \mid F \dot{\vee} G \mid \dot{\neg} F \mid \dot{\exists}_x F \mid \circ F \mid \Box F \mid \Diamond F$$

The constraint c (i.e., a first-order formula in the cs) represents a *state formula*. The dotted symbols represent the usual (temporal) boolean and existential operators. As clarified later, the dotted notation is needed as in **CLTL** these operators do not always coincide with those in the cs. The symbols $\circ$, $\Box$, and $\Diamond$ denote the LTL modalities *next*, *always* and *eventually*. We use $F \dot{\Rightarrow} G$ for $\dot{\neg} F \dot{\vee} G$. Below we give the formulae a **CLTL** semantics. We first introduce some notation and the notion of *x-variant*. Intuitively, d is an x-variant of c iff they are the same except for the information about x.

Notation 2 *Given a sequence $\alpha = c_1.c_2 \ldots$, we use $\exists_x \alpha$ to denote the sequence $\exists_x c_1 \exists_x c_2 \ldots$. We shall use $\alpha(i)$ to denote the $i-th$ element of α.*

Definition 5 (x-variant). *A constraint d is an x-*variant *of c iff $\exists_x c = \exists_x d$. Similarly α' is an x-*variant *of α iff $\exists_x \alpha = \exists_x \alpha'$.*

Definition 6 (CLTL Semantics). *We say that α satisfies (or that it is a model of) F in* **CLTL**, *written $\alpha \models_{\mathrm{CLTL}} F$, iff $\langle \alpha, 1\rangle \models_{\mathrm{CLTL}} F$, where:*

$$\begin{array}{lll}
\langle \alpha, i\rangle \models_{\mathrm{CLTL}} \dot{\mathtt{true}} & & \langle \alpha, i\rangle \not\models_{\mathrm{CLTL}} \dot{\mathtt{false}} \\
\langle \alpha, i\rangle \models_{\mathrm{CLTL}} c & \textit{iff} & \alpha(i) \models c \\
\langle \alpha, i\rangle \models_{\mathrm{CLTL}} \dot{\neg} F & \textit{iff} & \langle \alpha, i\rangle \not\models_{\mathrm{CLTL}} F \\
\langle \alpha, i\rangle \models_{\mathrm{CLTL}} F \dot{\wedge} G & \textit{iff} & \langle \alpha, i\rangle \models_{\mathrm{CLTL}} F \textit{ and } \langle \alpha, i\rangle \models_{\mathrm{CLTL}} G \\
\langle \alpha, i\rangle \models_{\mathrm{CLTL}} F \dot{\vee} G & \textit{iff} & \langle \alpha, i\rangle \models_{\mathrm{CLTL}} F \textit{ or } \langle \alpha, i\rangle \models_{\mathrm{CLTL}} G \\
\langle \alpha, i\rangle \models_{\mathrm{CLTL}} \circ F & \textit{iff} & \langle \alpha, i+1\rangle \models_{\mathrm{CLTL}} F \\
\langle \alpha, i\rangle \models_{\mathrm{CLTL}} \Box F & \textit{iff} & \textit{for all } j \geq i \ \ \langle \alpha, j\rangle \models_{\mathrm{CLTL}} F \\
\langle \alpha, i\rangle \models_{\mathrm{CLTL}} \Diamond F & \textit{iff} & \textit{there is a } j \geq i \textit{ such that } \langle \alpha, j\rangle \models_{\mathrm{CLTL}} F \\
\langle \alpha, i\rangle \models_{\mathrm{CLTL}} \dot{\exists}_x F & \textit{iff} & \textit{there is an } x\textit{-variant } \alpha' \textit{ of } \alpha \textit{ such that } \langle \alpha', i\rangle \models_{\mathrm{CLTL}} F.
\end{array}$$

Define $[\![F]\!]=\{\alpha \,|\, \alpha \models_{\mathrm{CLTL}} F\}$. *$F$ is* **CLTL** valid *iff* $[\![F]\!] = \mathcal{C}^\omega$, *and* **CLTL** satisfiable *iff* $[\![F]\!] \neq \emptyset$.

Let us discuss a little about the difference between the boolean operators in the cs and the temporal ones to justify our dotted notation. A state formula c is satisfied only by those $e.\alpha'$ such that $e \models c$. So, the state formula $\mathtt{false}$ has at least one sequence that satisfies it; e.g. $\mathtt{false}^\omega$. On the contrary the temporal formula $\dot{\mathtt{false}}$ has no models whatsoever. Similarly, $c \,\dot\vee\, d$ is satisfied by those $e.\alpha'$ such that either $e \models c$ or $e \models d$ holds. Thus, in general $[\![c \,\dot\vee\, d]\!] \neq [\![c \vee d]\!]$. The same holds true for $\neg c$ and $\dot\neg\, c$.

Example 1. Let $e = c \vee d$ with $c = (x = 42)$ and $d = (x \neq 42)$. One can verify that $\mathcal{C}^\omega = [\![c \vee d]\!] \ni e^\omega \notin [\![c \,\dot\vee\, d]\!]$ and also that $[\![\neg c]\!] \ni \mathtt{false}^\omega \notin [\![\dot\neg\, c]\!]$. □

From the above example, one may be tempted to think of **CLTL** as being intuitionistic. Notice, however, that statements like $\dot\neg\, F \,\dot\vee\, F$ and $\dot\neg\,\dot\neg\, F \dot\Rightarrow F$ are **CLTL** valid.

Process Verification. Intuitively, $P \models_{\mathrm{CLTL}} F$ iff every sequence that P can possibly output, on inputs from arbitrary environments, satisfies F.

Definition 7 (Verification). *P satisfies F, written $P \models_{\mathrm{CLTL}} F$, iff $sp(P) \subseteq [\![F]\!]$.*

For instance, $\star\,\mathbf{tell}(c) \models_{\mathrm{CLTL}} \Diamond c$ as in every sequence output by $\star\,\mathbf{tell}(c)$ there must be an e entailing c. Also $P = \mathbf{tell}(c) + \mathbf{tell}(d) \models_{\mathrm{CLTL}} c \vee d$ and $P \models_{\mathrm{CLTL}} c \,\dot\vee\, d$ as every e output by P entails either c or d. Notice, however, that $Q = \mathbf{tell}(c \vee d) \models_{\mathrm{CLTL}} c \vee d$ but $Q \not\models_{\mathrm{CLTL}} (c \,\dot\vee\, d)$ in general, since Q can output an e which certainly entails $c \vee d$ and still entails neither c nor d - take e, c and d as in Example 1. Therefore, $c \,\dot\vee\, d$ distinguishes P from Q. The reader may now see why we wish to distinguish $c \,\dot\vee\, d$ from $c \vee d$.

3 Decidability Results for `ntcc`

We first present our decidability result for sp equivalence. We then show, with the help of this first result, our decidability result for the `ntcc` verification problem. Finally, we present the decidability results for validity and satisfiability in $\mathbf{CLTL}$.

The theory of Büchi FSA [2] is central to our results. These FSA are ordinary automata with an acceptance condition for infinite (or ω) sequences: an ω sequence is accepted iff the automaton can read it from left to right while visiting a final state infinitely often. The language recognized by a Büchi automaton A is denoted by $L(A)$. Regular ω-languages are those recognized by Büchi FSA.

For a better exposition of our results, it is convenient to look first into previous approaches to the decidability of the `ntcc` observational equivalences. First, we need the following definition.

Definition (Derivatives). *Define $P \Longrightarrow Q$ iff $P \xrightarrow{(c,d)} Q$ for some c, d. A process Q is a* derivative *of P iff $P = P_1 \Longrightarrow \ldots \Longrightarrow P_n = Q$ for some $P_1, \ldots, P_n$.*

Restricted Nondeterminism: Finitely Many States. Nielsen et al [16] show the decidability of output equivalence for the *restricted-nondeterministic* fragment of `ntcc` which only allows *$\star$-free processes* whose *summations are not in the scope of local operators*. First, the authors show that each process in this fragment has a *finite number of derivatives* (up-to output equivalence). Then, they show how to construct for any given restricted-nondeterministic P, a Büchi automaton that recognizes P's output behavior as an $\omega-$ language. Since language equivalence for Büchi FSA is decidable [22] the decidability of output equivalence follows. In his PhD dissertation [24] the author proved the decidability of the sp (and input-output) equivalence for restricted-nondeterministic processes by a reduction to that of output equivalence.

More Liberal Nondeterminism: Infinitely Many States. The above-mentioned FSA have states representing processes and transitions representing observable reductions. The automata are generated by an *algorithm* that uses the `ntcc` *operational semantics* to generate all possible observable reductions. Hence, the finiteness of the set of derivatives is crucial to guarantee termination. In fact, the algorithm may diverge if we allow arbitrary $\star$ processes, or summations within local operators because, as illustrated below, they can have *infinitely many derivatives* each with different observable behavior.

Example 2. (1) Notice that $\star P$ has *infinitely many derivatives* of the form $\mathbf{next}^{\,n} P$ and each of them may exhibit different observable behavior. (2) Also $R = !!P$ generates a sequence $R \Longrightarrow P \parallel R \Longrightarrow (P \parallel P) \parallel R \Longrightarrow \ldots$. If P is not *restricted-nondeterministic*, this sequence can give rise to infinitely many derivatives of R each of them giving different output sequences: E.g., take $P = \star\,\mathbf{tell}(c)$ and note that $\prod_{k+1} P$ can tell c at $k+1$ different time units but $\prod_k P$ can only do it at k different units. The same occurs if $P = \delta\mathbf{tell}(c)$ where δQ denotes an arbitrary *possibly infinite* delay of Q. This delay operator can be recursively defined as $\delta Q \stackrel{\text{def}}{=} Q + \mathbf{next}\,\delta Q$ and such a kind of definitions can be derived in `ntcc` using replication together with *blind choice summations within local processes* [15]. □

3.1 Decidability of SP Equivalence

We shall show a Büchi FSA characterization of the sp for processes that can exhibit infinitely many, observationally different, derivatives—of the kind illustrated in Example 2. One particular difference with the work previously mentioned is that, to get around the algorithmic problems illustrated above, the characterizations will be guided by the *sp denotational semantics* of `ntcc` [14] rather than its operational semantics.

The `ntcc` denotational semantics $[\![\cdot]\!] : \mathit{Proc} \to \mathcal{P}(\mathcal{C}^\omega)$, given in Table 2 by taking $[\![\cdot]\!] = [\![\cdot]\!]^{\mathcal{C}}$, is meant to capture the sp. From [5], however, we know that there cannot be a $f : \mathit{Proc} \to \mathcal{P}(\mathcal{C}^\omega)$, compositionally defined, such that $f(P) = \mathit{sp}(P)$ for all P. Nevertheless, Palamidessi et al [17] showed that $[\![P]\!] = \mathit{sp}(P)$ for all P in the so-called *locally-independent* fragment. This fragment forbids non-unary summations (and "unless" processes) whose guards depend on local variables.

Definition 8 (Locally-Independent Processes). *P is* locally-independent *iff for every* $\mathbf{unless}\ c\ \mathbf{next}\, Q$ *and* $\sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ Q_i$ *($|I| \geq 2$) in P, neither c nor the c_i's contain variables in $bv(P)$ (i.e., the bound variables of P).*

Theorem 1 (Palamidessi et al [17]). *If P is locally-independent then $[\![P]\!] = sp(P)$.*

The locally-independent fragment allows $\star$ processes and also blind-choice within local operators which may exhibit infinitely many observationally different derivatives, as illustrated in Example 2. Furthermore, every summation whose guards are either all equivalent or mutually exclusive can be encoded in this fragment [24]. The applicability of this fragment is witnessed by the fact all the `ntcc` application examples in [15, 16, 24] can be model as local-independent processes.

SP Büchi FSA. We shall give a Büchi characterization of $sp(P)$ from its denotation $[\![P]\!]$. We then benefit from the simple set-theoretical and compositional nature of $[\![\cdot]\!]$ as well as the closure properties of regular $\omega-$languages. A technical problem arises: There can be infinitely many c's such that $c.\alpha \in [\![P]\!]$ as $\mathcal{C}$ may be infinite, but the alphabets of Büchi FSA are finite. It is therefore convenient to confine $[\![P]\!]$ to a suitable finite set $S \subseteq_{fin} \mathcal{C}$ including its so-called *relevant constraints*.

Definition (Relevant Constraints). *Given $\mathcal{S} \subseteq \mathcal{C}$, let $\overline{\mathcal{S}}$ be the closure under conjunction and implication of $\mathcal{S}$. Let $C : Proc \to \mathcal{P}(\mathcal{C})$ be defined as:*

$$
\begin{array}{ll}
C(\mathbf{skip}) = \{\mathtt{true}\} & C(\mathbf{abort}) = \{\mathtt{true}\} \\
C(\sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i) = \bigcup_{i \in I} \{c_i\} \cup C(P_i) & C(\mathbf{tell}(c)) = \{c\} \\
C(P \parallel Q) = C(P) \cup C(Q) & C(\mathbf{next}\, P) = C(P) \\
C(!\, P) = C(P) & C(\star\, P) = C(P) \\
C((\mathbf{local}\, x)\, P) = \{c, \exists_x c, \forall_x c \mid c \in \overline{C(P)}\} &
\end{array}
$$

Define the relevant constraints *for $P_1, ..., P_n$, written $\mathcal{C}(P_1, \ldots, P_n)$, as the closure under conjunction of $\overline{C(P_1)} \cup \ldots \cup \overline{C(P_n)}$.*

The interested reader is referred to the author's PhD dissertation [24] for the intuition behind the notion of relevant constraints. Now, consider two locally-independent processes P and Q. Clearly, $\mathcal{C}(P, Q)$ is finite. Moreover, it contains the constraints that are indeed relevant for the sp equivalence of P and Q.

Theorem 2 (Relevant SP Denotational Characterization). *Let P and Q be locally-independent. Then $[\![P]\!]^{\mathcal{C}(P,Q)} = [\![Q]\!]^{\mathcal{C}(P,Q)}$ iff $P \sim_{sp} Q$.*

Proof. (Outline) Let $S = \mathcal{C}(P, Q)$. Verify by induction on P that $c_1.c_2.\ldots \in [\![P]\!]$ iff $c_1(S).c_2(S).\ldots \in [\![P]\!]^S$, where $c_i(S)$ is the strongest $e \in S$ wrt $\models$ such that $c_i \models e$ ($c_i(S)$ is well-defined as S is closed under $\wedge$). By symmetry the same holds for Q. Then conclude that $[\![P]\!]^S = [\![Q]\!]^S$ iff $[\![P]\!] = [\![Q]\!]$. The result follows from Theorem 1 . $\square$

Büchi Constructions. Having identified our finite set of relevant constraints for the sp equivalence characterization, we now proceed to construct for each P and $S \subseteq_{fin} \mathcal{C}$, a Büchi automaton A_P^S recognizing $[\![P]\!]^S$. Each A_P^S will be *compositionally* constructed in the sense that it will be built solely from information about FSA of the form A_Q^S where Q is a subprocess of P. The most interesting case of this construction is replication, as described in the proof of the lemma below.

DABORT:	$[\![\mathbf{abort}]\!]^S \;=\; \emptyset$
DTELL:	$[\![\mathbf{tell}(c)]\!]^S \;=\; \{d.\alpha \in S^\omega \mid d \models c\}$
DSUM:	$[\![\, \sum_{i\in I} \mathbf{when}\; c_i\; \mathbf{do}\; P_i \,]\!]^S \;=\; \bigcup_{i\in I} \{d.\alpha \in S^\omega \mid d \models c_i \text{ and } d.\alpha \in [\![P_i]\!]^S\} \cup \bigcap_{i\in I}\{d.\alpha \in S^\omega \mid d \not\models c_i\}$
DPAR:	$[\![P \parallel Q]\!]^S \;=\; [\![P]\!]^S \cap [\![Q]\!]^S$
DLOC:	$[\![(\mathbf{local}\, x)\; P]\!]^S \;=\; \{\alpha \in S^\omega \mid \text{there exists } \alpha' \in [\![P]\!]^S \text{ such that } \exists_x \alpha' = \exists_x \alpha\}$
DNEXT:	$[\![\mathbf{next}\, P]\!]^S \;=\; \{d.\alpha \in S^\omega \mid \alpha \in [\![P]\!]^S\}$
DUNL:	$[\![\mathbf{unless}\; c\; \mathbf{next}\; P]\!]^S \;=\; \{d.\alpha \in S^\omega \mid d \models c\} \cup \{d.\alpha \mid d \not\models c \text{ and } \alpha \in [\![P]\!]^S\}$
DREP:	$[\![!\, P]\!]^S \;=\; \{\alpha \in S^\omega \mid \text{ for all } \beta, \alpha' \text{ such that } \alpha = \beta.\alpha', \text{ we have } \alpha' \in [\![P]\!]^S\}$
DSTAR:	$[\![\star\, P]\!]^S \;=\; \{\beta.\alpha \in S^\omega \mid \alpha \in [\![P]\!]^S\}$

Table2. SP Denotation. Above $S \subseteq \mathcal{C}$ and $\beta.\alpha'$ is the concatenation of the finite sequence β followed by α'. The sequence $\exists_x \alpha$ results from applying $\exists_x$ to each constraint in α. In DSUM if $I = \emptyset$, the indexed union and intersection are taken to be $\emptyset$ and S^ω, respectively.

Lemma 1. *Given P and $S \subseteq_{fin} \mathcal{C}$ one can effectively construct a Büchi automaton A_P^S over the alphabet S such that $L(A_P^S) = [\![P]\!]^S$.*

Proof. We shall give the construction of each A_P^S by case analysis on the structure of P. We only describe some of the cases. The others are similar or simpler.

Replication Automaton. $P = !Q$: We have $\alpha \in [\![!Q]\!]^S$ iff every suffix of α is in $[\![Q]\!]^S$. We then need to construct a $A_{!P}^S$ that accepts an infinite sequence α iff *every suffix* of it is accepted by A_Q^S. At first, such a construction may seem somewhat complex to realize. Nevertheless, notice that the process $!Q$ is *dual* to $\star Q$ in the sense expressed in [17]:

$$[\![!Q]\!] = \nu_X([\![Q]\!] \cap \{d.\alpha \mid \alpha \in X\}) \quad \text{while} \quad [\![\star Q]\!] = \mu_X([\![Q]\!] \cup \{d.\alpha \mid \alpha \in X\})$$

where ν, μ are the greatest and least fixpoint (resp.) in the complete lattice $(\mathcal{P}(\mathcal{C}^\omega), \subseteq)$. In fact, $\alpha \in [\![\star Q]\!]^S$ iff *there is a suffix* of α in $[\![Q]\!]^S$. So, given an automaton B define $\star B$ as the automaton that accepts α iff *there is a suffix* of α accepted by B. The construction of $\star B$ is simple and similar to that of $A_{\star Q}^S$ below. Now, given A, one can construct the automaton $\overline{A}$ for the complement of $L(A)$ [22]. Hence, keeping duality in mind, we can take $A_{!Q}^S$ to be $\overline{\star\overline{A_Q^S}}$.

Unbounded but Finite Delay Automaton. $P = \star Q$: The states of $A_{\star Q}^S$ are those of A_Q^S plus an additional state s_0. Its final states are those of A_Q^S. Its initial states are those of A_Q^S plus s_0. The transitions are those of A_Q^S plus transitions labelled with c, for each $c \in S$, from s_0 to itself and from s_0 to each initial state of A_Q^S. The transitions "looping" into the initial state s_0 model the "arbitrary but finite delay" of $\star Q$.

Local Automaton. $P = (\mathbf{local}\, x)\, Q$: The states of $A^S_{(\mathbf{local}\, x)\, Q}$ are those of A^S_Q. Its initial and final states are those of A_Q. For each $c \in S$, $A^S_{(\mathbf{local}\, x)\, Q}$ has a transition from p to q labeled with c iff A^S_Q has a transition from p to q labelled with d for some $d \in S$ such that $\exists_x d = \exists_x e$.

Parallel Automaton. $P = Q \parallel R$: The theory of Büchi FSA gives us a construction for the intersection of ω languages: Given A^S_Q and A^S_R one can construct an automaton that recognizes $L(A^S_Q) \cap L(A^S_R)$ [3]. Take $A^S_{Q\parallel R}$ to be such an automaton.

Skip and Tell Automata. $P = \mathbf{skip}$: $A^S_{\mathbf{skip}}$ has a single (initial, accepting) state, and for each $c \in S$, there is a transition labeled with c from this single state to itself. $P = \mathbf{tell}(c)$: $A^S_{\mathbf{tell}(c)}$ has exactly two states: one is its initial state and the other is its accepting one: the unique state of $A^S_{\mathbf{skip}}$. The transitions of $A^S_{\mathbf{tell}(c)}$ are those of $A^S_{\mathbf{skip}}$ plus a transition from the initial state to the state of $A^S_{\mathbf{skip}}$ labeled with d for each $d \in S$ such that $d \models c$.

One can verify the correctness of A^S_P using induction on the structure of P. It is easy to see that A^S_P can be effectively constructed, thus concluding the proof. □

We now state our first decidability result: The decidability of the sp equivalence.

Theorem 3 (Decidability of $\sim_{sp}$). *Given two locally-independent process P and Q, the question of whether $P \sim_{sp} Q$ is decidable.*

Proof. From Theorem 2, Lemma 1 and the decidability of language equivalence for Büchi FSA [22].

3.2 Decidability Results for Verification and CLTL

Here we show the decidability results for the verification problem (i.e., given P and F whether $P \models_{\mathrm{CLTL}} F$, see Definition 7) as well as for **CLTL** (Definition 6).

Recall the **CLTL** models of a given formula F are in $\mathcal{C}^\omega$. Thus, for our decidability results we may attempt to give a Büchi characterization of **CLTL** formulae facing therefore the problem pointed out in the previous section: $\mathcal{C}$ may be infinite but Büchi FSA only have finite alphabets. One could try to restrict $[\![F]\!]$ to its "relevant" constraints as we did for $[\![P]\!]$. This appears to be possible for negation-free formulae but we do not know yet how to achieve this for the full language.

Nevertheless, for negation-free formulae there is an alternative to obtain the results by appealing to Theorem 3 and the correspondence between processes and LTL formulae. More precisely, we show that one can construct a locally-independent R_F whose sp corresponds to $[\![F]\!]$ if F is a restricted-negation formula in the following sense:

Definition 9 (Restricted-Negation LTL). *F is a* restricted-negation formula *iff whenever $\dot\neg\, G$ appears in F then G is a state formula (i.e., $G = c$ for some c).*

Recall that in **CLTL** $\dot\neg\, c$ and the state formula $\neg c$ do not match (Example 1). In fact, we need $\dot\neg\, c$ should we want to express the minimal implication form $c \mathrel{\dot\Rightarrow} D = \dot\neg\, c \mathrel{\dot\vee} D$.

Lemma 2. *Let F be a restricted-negation formula. One can effectively construct a locally-independent R_F such that $[\![F]\!] = sp(R_F)$.*

Proof. Take $R_F = h(F)$ where h is the map from restricted-negation formulae to locally-independent processes given by

$$
\begin{array}{ll}
h(\dot{\texttt{true}}) = \mathbf{skip} & h(\dot{\texttt{false}}) = \mathbf{abort} \\
h(c) = \mathbf{tell}(c) & h(\dot{\neg}\, c) = \mathbf{when}\ c\ \mathbf{do\ abort} \\
h(F \dot{\wedge} G) = h(F) \parallel h(G) & h(F \dot{\vee} G) = h(F) + h(G) \\
h(\dot{\exists}\, xF) = (\mathbf{local}\, x)\, h(F) & h(\circ F) = \mathbf{next}\, h(G) \\
h(\Box F) = !\, h(F) & h(\Diamond F) = \star h(F)
\end{array}
$$

Obviously, $h(F)$ can be effectively constructed. One can verify that $[\![F]\!] = [\![h(F)]\!]$ by induction on the structure of F. From Theorem 1 we obtain $[\![F]\!] = sp(h(F))$. □

Notice that the map h above reveals the close connection between `ntcc` and LTL. We can now state the decidability of the verification problem for `ntcc`.

Theorem 4 (Decidability of Verification). *Let F be a restricted-negation formula. Let P be locally-independent. The question of whether $P \models_{\text{CLTL}} F$ is decidable.*

Proof. From Theorem 3 by using the following reduction to sp equivalence:

$$
\begin{array}{rllll}
P \models_{\text{CLTL}} F & \text{iff } sp(P) & \subseteq & [\![F]\!] & \text{(Definition 7)} \\
& \text{iff } sp(P) & \subseteq & [\![R_F]\!] & \text{(Lemma 2)} \\
& \text{iff } \quad [\![P]\!] & \subseteq & [\![R_F]\!] & \text{(Theorem 1)} \\
& \text{iff } \quad [\![P]\!] & = & [\![R_F]\!] \cap [\![P]\!] & \\
& \text{iff } \quad [\![P]\!] & = & [\![R_F \parallel P]\!] & \text{(Definition of } [\![\cdot]\!]) \\
& \text{iff } & P \sim_{sp} & R_F \parallel P & \text{(Theorem 2)}
\end{array}
$$
□

We can reduce the validity of implication to the verification problem. Therefore,

Theorem 5 (Decidability for Validity of Implication). *Let F and G be restricted-negation formulae. The question of whether $F \dot{\Rightarrow} G$ is* **CLTL** *valid is decidable.*

Proof. $F \dot{\Rightarrow} G$ iff $[\![F]\!] = sp(R_F) \subseteq [\![G]\!]$ by Definition 6 and Lemma 2. Then $F \dot{\Rightarrow} G$ iff $R_F \models_{\text{CLTL}} G$ by Definition 7. The result follows from Theorem 4. □

As an immediate consequence of the above theorem we obtain the following:

Corollary 1 (Decidability of CLTL Satisfiability and Validity). *Let F be an arbitrary restricted-negation formula. The questions of whether F is* **CLTL** *valid and whether F is* **CLTL** *satisfiable are both decidable.*

Proof. F is **CLTL** valid iff $\dot{\texttt{true}} \dot{\Rightarrow} F$ is **CLTL** valid, and F is **CLTL** satisfiable iff $F \dot{\Rightarrow} \dot{\texttt{false}}$ is not **CLTL** valid. The result follows from Theorem 5. □

4 An Application to Manna and Pnueli's LTL

We now apply the previous results on our `ntcc` LTL (**CLTL**) to standard first-order LTL, henceforth called **LTL**, as presented by Manna and Pnueli in [10]. Namely, we obtain a new positive decidability result on the satisfiability of a first-order fragment of **LTL** by a reduction to that of **CLTL**. The relevance of our result is that **LTL** is not recursively axiomatizable [1] and, therefore, the satisfiability problem is undecidable for the full language of **LTL**. We confine ourselves to having $\circ$, $\Box$, and $\Diamond$ as modalities. This is sufficient for making a recursive axiomatization impossible [11].

We shall recall briefly some LTL notions given in [10]. We presuppose an underlying first-order language $\mathcal{L}$ (including equality) with its (nonlogical) symbols interpreted over some concrete domains such as the natural numbers.

A *state* s is an interpretation that assigns to each variable x in $\mathcal{L}$ a value $s[x]$ over the appropriate domain. The interpretation is extended to $\mathcal{L}$ expressions in the usual way. For example, if f is a function symbol of arity 1, $s[f(x)] = f(s[x])$. We write $s \models c$ iff c is true wrt s in the given interpretation of the $\mathcal{L}$ symbols. For instance, if $+$ is interpreted as addition over the natural numbers and $s[x] = 42$ then $s \models \exists_y(x = y + y)$. We say that c is *state valid* iff $s \models c$ for every state s.

A *model* is an infinite sequence of states. We shall use σ to range over models. The variables of $\mathcal{L}$ are partitioned into *rigid* and *flexible* variables. Each model σ must satisfy the *rigidity* condition: If x is rigid and s, s' are two states in σ then $s[x] = s'[x]$.

The syntax of **LTL** is to that of **CLTL** given in Definition 4. In this case, however, x is a variable in $\mathcal{L}$ and c represents a first-order formula over $\mathcal{L}$.

The semantics of **LTL** is similar that of **CLTL** (Definition 6) except that now the formulae are satisfied by sequences of states. We then need to extend the notion of x-variant (Definition 5) to states: s is *x-variant* of s' iff $s[y] = s'[y]$ for every variable y in $\mathcal{L}$ different from x.

Definition (LTL Semantics). *A model σ* satisfies *F in* **LTL***, notation $\sigma \models_{\mathrm{LTL}} F$, iff $\langle \sigma, 1\rangle \models_{\mathrm{LTL}} F$ where $\langle \sigma, i\rangle \models_{\mathrm{LTL}} F$ is obtained from Definition 6 by replacing α and $\models_{\mathrm{CLTL}}$ with σ and $\models_{\mathrm{LTL}}$, respectively. We say that F is* **LTL** satisfiable *iff $\sigma \models_{\mathrm{LTL}} F$ for some σ, and that F is* **LTL** valid *iff $\sigma \models_{\mathrm{LTL}} F$ for all σ.*

In order to prove our decidability result, *we assume that state validity (the set of valid state formulae) is decidable*. From [9] we know that even under this assumption the **LTL** defined above is undecidable. In contrast, under the assumption, **LTL** satisfiability is decidable for the fragment in which temporal operators are not allowed within the scope of quantifiers as it can be reduced to that of propositional LTL [1].

Example 3. Let us now illustrate the interaction between quantifiers, modalities, flexible and rigid variables in **LTL**. The formula $\Box \dot{\exists}_u(x = u \mathbin{\dot\wedge} \circ x = u + 1)$, where x is flexible and u is rigid, specifies sequences in which x increases by 1 from each state to the next. This example also illustrates that existential quantification over rigid variables provides for the specification of counter computations, so we may expect their absence to be important in our decidability result. In fact, we shall state the **LTL** decidability for the *restricted-negation* fragment (Definition 9) with flexible variables only. $\square$

Removing Existential Quantifiers. One might be tempted to think that, without universal quantification and without rigid variables, one could remove the existential quantifiers rather easily: Pull them into outermost position with the appropriate α-conversions to get a prenex form, then remove them since $\dot{\exists}_x F$ is **LTL** satisfiable iff F is **LTL** satisfiable. But this procedure does not quite work; it does not preserve satisfiability:

Example 4. Let $F = (x = 42 \mathbin{\dot\wedge} \circ x \neq 42)$, $G = \dot{\exists}_x \Box F$ and $H = \Box \dot{\exists}_x F$ where x is flexible. One can verify that unlike H, $\Box F$ and thus G are not **LTL** satisfiable. Getting rid of existential quantifiers is not as obvious as it may seem. $\Box$

Relating* CLTL *and* LTL *Satisfiability. Let us give some intuition on how to obtain a *reduction* from **LTL** satisfiability to **CLTL** satisfiability. In what follows we confine ourselves to restricted-negation formulae without rigid variables. One can verify that $\neg c$ and $\dot\neg\, c$ have the same **LTL** models. So, in the reduction we can assume wlg that F has no $\dot\neg$ symbols. Notice that $F = (x = 42 \mathbin{\dot\vee} x \neq 42)$ is **LTL** valid but not **CLTL** valid (Example 1). However, F is satisfiable in both logics. In general, if F is **LTL** satisfiable then F is **CLTL** satisfiable. The other direction does not necessarily hold. $\Diamond\, \texttt{false}$ is not **LTL** satisfiable but it is **CLTL** satisfiable—recall from Section 2.2 that in **CLTL** $\texttt{false}$ is not the same as $\dot{\texttt{false}}$. For example, $\texttt{false}^\omega \models_{\text{CLTL}} \Diamond\, \texttt{false}$. However, we can get around this mismatch by using $\Box \dot\neg\, \texttt{false}$ to exclude **CLTL** models containing $\texttt{false}$ as shown in the lemma below.

Recall that (Σ, Δ) (Definition 1) denotes the underlying cs and $\mathcal{L}$ denotes the underlying first-order language of state formulae. Also recall that both Δ and the set of valid state formulae are required to be decidable.

Lemma 3. *Assume that the cs (Σ, Δ) has $\mathcal{L}$ as first-order language and Δ is the set of valid state formulae. Then*

$$F \textit{ is } \textbf{LTL} \textit{ satisfiable} \quad \textit{iff} \quad F \mathbin{\dot\wedge} \Box \dot\neg\, \texttt{false} \textit{ is } \textbf{CLTL} \textit{ satisfiable,}$$

if F is a restricted-negation formula with no occurrences of $\dot\neg$ and with no rigid variables.

Proof. (Outline) "If" direction: Verify that if $\alpha = c_1.c_2.\ldots \models_{\text{CLTL}} F \mathbin{\dot\wedge} \Box \dot\neg\, \texttt{false}$ then for any $\sigma = s_1.s_2.\ldots$ where $s_i \models c_i$ $(i \geq 1)$, we have $\sigma \models_{\text{LTL}} F$. Conclude the result by using the observation that if $\alpha \models_{\text{CLTL}} F \mathbin{\dot\wedge} \Box \dot\neg\, \texttt{false}$ then α contains no $\texttt{false}$. "Only if" direction: Verify that if $\sigma = s_1.s_2 \ldots \models_{\text{LTL}} F$ then there is an $\alpha = c_1.c_2 \ldots$ with $s_i \models c_i$ $(i \geq 1)$ such that $\alpha \models_{\text{CLTL}} F \mathbin{\dot\wedge} \Box \dot\neg\, \texttt{false}$. Each c_i can be taken to be the strongest constraint (under $\models$) satisfied by s_i in the closure under conjunction of the constraints in F. $\Box$

We can now state the decidability result we claimed for first-order LTL.

Theorem 6 (Decidability of LTL Satisfaction). *Let F be a restricted-negation formula without rigid variables. The question of whether F is* **LTL** *satisfiable is decidable.*

Proof. From Lemma 3 and Corollary 1, and the fact that one can freely replace $\dot\neg$ by $\neg$ in F and the resulting formula will have the same **LTL** models than the original F. $\Box$

5 Concluding Remarks

We presented positive decidability results for the sp behavioral equivalence, the `ntcc` verification problem, and the `ntcc` specification first-order temporal logic **CLTL** . These results apply to *infinite-state* processes. A somewhat interesting aspect is that for proving the results it turned out to be convenient to work with the `ntcc` *denotational semantics* rather than with its operational counterpart. Also the use of Büchi automata-theoretic techniques in these results highlights the automata-like flavor of `ntcc`.

Furthermore, by using a reduction to **CLTL** satisfiability, we identified a first-order fragment of the standard LTL [10] for which satisfiability is decidable. The result contributes to the understanding of the relation between (timed) ccp and (temporal) classic logic and also illustrates the applicability of timed ccp to other theories of concurrency.

Related Work. Nielsen et al [14] proved the decidability of the sp equivalence and other behavioral equivalences for several deterministic timed ccp languages. In another work Nielsen et al [16] showed that output equivalence is decidable for a restricted nondeterministic `ntcc` fragment. The results in [14, 16] are obtained by showing that the processes in these languages are indeed *finite-state*. Nielsen et al [14] also show that the sp equivalence is *undecidable* if recursion is present in the language.

Saraswat et al [21] showed how to compile parameterless recursion tcc processes (basically finite-state deterministic `ntcc` processes) into FSA in a compositional way. Such FSA provide a simple and useful execution model for tcc but not a direct way of verifying sp (or input-output) equivalence. In fact, unlike our FSA constructions, the standard language equivalence between these FSA does not necessarily imply sp equivalence (or input-output) of the processes they represent.

Another interesting approach to timed ccp verification is that by Falaschi et al [8]. The authors show how to construct structures (models) of tcc processes which then, by restricting the domains of variables to be finite, can be used for model-checking. Notice that in our results we make no assumptions about the domains of variables being finite.

The notion of constraint in other declarative formalisms such as Constraint Logic Programming (CLP) and Constraint Programming (CP) has also been used for the verification of infinite-state systems. Delzanno and Podelski [6] showed how to translate infinite-state systems into CLP programs to verify safety and liveness properties. Esparza and Melzer [7] used CP in a semi-decision algorithm to verify 1-safe Petri Nets.

Merz [11] and Hodkinson et al [9] identified interesting decidable first-order fragments of LTL. These fragments are all monadic and without equality. A difference with our work is that these fragments do not restrict the use of negation or rigid variables, and our fragment is not restricted to be monadic or equality-free.

Future Work. The author believes that we can dispense with the restriction on the occurrences of negation in our results. If the claim is true, then the inference system for `ntcc` [17] would be complete for locally-independent processes (and not just relative complete). This is because we would be able to determine the validity of arbitrary `ntcc` LTL implication as required by the consequence rule. It will be, therefore, interesting to be able to prove this claim.

In our Büchi FSA constructions we were not concerned with state space issues (e.g., see the "double complementation" construction for the replication automaton in Section 3.1). For verification purposes, it is important to look into these issues.

Acknowledgments. Many thanks to Catuscia Palamidessi, Mogens Nielsen, Joachim Parrow, Jiri Srba, Igor Walukiewicz, Martin Abadi, Ian Hodkinson, Stephan Merz, Richard Mayr, Gerardo Schneider and Martin Leucker for helpful comments.

References

1. M. Abadi. The power of temporal proofs. *Theoretical Computer Science*, 65:35–84, 1989.
2. J. R. Buchi. On a decision method in restricted second order arithmetic. In *Proc. Int. Conf. on Logic, Methodology, and Philosophy of Science*, pages 1–11, 1962.
3. Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *Computer and System Sciences*, 10:19–35, 1974.
4. F. de Boer, M. Gabbrielli, and M. C. Meo. A timed concurrent constraint language. *Information and Computation*, 161:45–83, 2000.
5. F. de Boer, M. Gabbrielli, E. Marchiori, and C. Palamidessi. Proving concurrent constraint programs correct. *ACM Transactions on Programming Languages and Systems*, 19(5), 1997.
6. G. Delzanno and A. Podelski. Model checking in CLP. *TACAS'99.* LNCS 1579, 1999.
7. J. Esparza and S. Melzer. Model checking LTL using constraint programming. In *Proc. of ICATPN'97.* LNCS 1248, 1997.
8. M. Falaschi, A. Policriti, and A. Villanueva. Modelling timed concurrent systems in a temporal concurrent constraint language - I. ENTCS 48, 2001.
9. I. Hodkinson, F. Wolter, and M. Zakharyasche. Decidable fragments of first-order temporal logic. *Ann. Pure. Appl. Logic*, 106:85–134, 2000.
10. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems, Specification*. Springer, 1991.
11. S. Merz. Decidability and incompleteness results for first-order temporal logics of linear time. *Journal of Applied Non-Classical Logic*, 2(2), 1992.
12. R. Milner. A finite delay operator in synchronous ccs. TR CSR-116-82, Univ. of Edinburgh.
13. R. Milner. *Communicating and Mobile Systems: the π-calculus*. 1999.
14. M. Nielsen, C. Palamidessi, and F. Valencia. On the expressive power of concurrent constraint programming languages. In *PPDP 2002*, pages 156–167. ACM Press, October 2002.
15. M. Nielsen, C. Palamidessi, and F. Valencia. Temporal concurrent constraint programming: Denotation, logic and applications. *Nordic Journal of Computing*, 9(2):145–188, 2002.
16. M. Nielsen and F. Valencia. *Temporal Concurrent Constraint Programming: Applications and Behavior.* LNCS 2300:298–324, 2002.
17. C. Palamidessi and F. Valencia. A temporal concurrent constraint programming calculus. In *Proc. of CP'01*. LNCS 2239, 2001.
18. V. Saraswat, R. Jagadeesan, and V. Gupta. Foundations of timed concurrent constraint programming. In *Proc. of LICS'94*, pages 71–80, 1994.
19. V. Saraswat, R. Jagadeesan, and V. Gupta. Programming in timed concurrent constraint languages. In *Constraint Programming: Proc. 1993*, pages 361–410. Springer-Verlag, 1994.
20. V. Saraswat, R. Jagadeesan, and V. Gupta. Timed default concurrent constraint programming. *Journal of Symbolic Computation*, 22:475–520, 1996.
21. V. Saraswat, M. Rinard, and P. Panangaden. The semantic foundations of concurrent constraint programming. In *POPL '91*, pages 333–352, 1991.
22. A. Sistla, M. Vardi, and P. Wolper. The complementation problem for buchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.
23. S. Tini. On the expressiveness of timed concurrent constraint programming. ENTCS 27, 1999.
24. F. Valencia. *Temporal Concurrent Constraint Programming*. PhD thesis, BRICS Univ. of Aarhus, 2003. Available online via http://www.brics.dk/∼fvalenci/publications.html.
25. M. Vardi. An automata-theoretic approach to linear temporal logic. LNCS 1043, 1996.