Abstract Diagnosis for *tccp* using a Linear Temporal Logic*

Marco Comini[1], Laura Titolo[1], and Alicia Villanueva[2]

[1] DIMI, Università degli Studi di Udine,
{`marco.comini,laura.titolo`}`@uniud.it`
[2] DSIC, Universitat Politècnica de València
`villanue@dsic.upv.es`

Abstract. Automatic techniques for program verification usually suffer the well-known state explosion problem. Most of the classical approaches are based on browsing the structure of some form of model (which represents the behavior of the program) to check if a given specification is valid. This implies that a part of the model has to be built, and sometimes the needed fragment is quite huge.
In this work, we provide an alternative automatic decision method to check whether a given property, specified in a linear temporal logic, is *valid* w.r.t. a *tccp* program. Our proposal (based on abstract interpretation techniques) does not require to build any model at all. Our results guarantee correctness but, as usual when using an abstract semantics, completeness is lost.

Key Words: concurrent constraint paradigm, linear temporal logic, abstract diagnosis, decision procedures, program verification

1 Introduction

The Concurrent Constraint Paradigm (*ccp*) is a simple, logic model which is different from other (concurrent) programming paradigms mainly due to the notion of store-as-constraint that replaces the classical store-as-valuation model. It is based on an underlying constraint system that handles constraints on variables, thus, it deals with partial information. One challenging characteristic of the *ccp* framework is that programs can manifest non-monotonic behaviors, implying that standard approaches cannot be directly adapted. Within this family, [dBGM00] introduced the *Timed Concurrent Constraint Language* (*tccp* in short) by adding to the original *ccp* model the notion of time and the ability to capture the absence of information. With these features, one can specify behaviors typical of reactive systems such as *timeouts* or *preemption* actions.

* This work has been partially supported by the EU (FEDER) and the Spanish MEC/MICINN, ref. TIN 2010-21062-C02-02, and by Generalitat Valenciana, ref. PROMETEO2011/052.

It is well-known that modeling and verifying concurrent systems by hand can be an extremely hard task. Thus, the development of automatic formal methods is essential. One of the most known techniques for formal verification is model checking, that was originally introduced in [CE81,QS82] to automatically check if a finite-state system satisfies a given property. It consisted in an exhaustive analysis of the state-space of the system; thus the state-explosion problem is its main drawback and, for this reason, many proposals in the literature try to mitigate it. Some of the more successful ones are the symbolic approach [BCM+92,HNSY94,BCC+03], on-the-fly model checking [Hol96] and the abstract interpretation based techniques [CGL92,Dam96]. The model-checking technique for *tccp* was first defined in [FV06], and also in this setting (optimized) symbolic and abstract versions were later defined [AFV05,AGPV05].

All the proposals of model checking have in common that a *part* of the model of the (target) program has to be built, and sometimes the needed fragment is quite huge. In this work, we propose a completely different approach to the formal verification of temporal (LTL) properties of concurrent (reactive) systems specified in *tccp*. We formalize a method to validate a specification of the expected behavior of a *tccp* program P, expressed by a linear temporal formula ϕ, which does not require to build any model at all.

The linear temporal logic we use to express specifications, csLTL, is an adaptation of the propositional LTL logic to the concurrent constraint framework, following the ideas of [PV01,dBGM01,dBGM02,Val05]. It is expressive enough to represent the abstract semantics of *tccp* with much precision. This logic is also used as the basis of the abstract domain for a new (abstract) semantics for the language.

In brief, our method is an *extension* of abstract diagnosis for *tccp* [CTV11] where the abstract domain $\mathbb{F}$ is formed by csLTL formulas. We cannot use the original abstract diagnosis framework of [CTV11] since $\mathbb{F}$ is not a complete lattice.

The contributions of this work are the following:

– A new abstract semantics for *tccp* programs based on csLTL formulas;
– A novel and effective method to validate csLTL properties based on the ideas of abstract diagnosis. This proposal intuitively consists in viewing P as a formula transformer by means of an (abstract) immediate consequence operator $\hat{\mathcal{D}}[\![P]\!]$ which works on csLTL formulas. Then, to decide the validity of ϕ, we just have to check if $\hat{\mathcal{D}}[\![P]\!]_\phi$ (i.e., the P-transformation of ϕ) implies ϕ;
– An automatic decision procedure for csLTL properties that makes our method effective.

With our technique we can check, for instance, that each time a train is approaching, the gate is down, or that whenever a train has crossed, the gate is up. When a property results non valid, the method identifies the buggy process declaration.

A proof of concept prototype of our technique is available online at URL `http://safe-tools.dsic.upv.es/tadi/`.

2 The small-step operational behavior of the *tccp* language

The *tccp* language [dBGM00] is particularly suitable to specify both reactive and time critical systems. As the other languages of the *ccp* paradigm [Sar93], it is parametric w.r.t. a *cylindric constraint system* which handles the data information of the program in terms of constraints. The computation progresses as the concurrent and asynchronous activity of several agents that can (monotonically) accumulate information in a *store*, or query some information from it. Briefly, a cylindric constraint system[3] is an algebraic structure $\mathbf{C} = \langle \mathcal{C}, \leq, \otimes, \oplus, \mathit{false}, \mathit{true}, \mathit{Var}, \exists \rangle$ composed of a set of constraints $\mathcal{C}$ such that $(\mathcal{C}, \leq)$ is a complete algebraic lattice where $\otimes$ is the *lub* operator and *false* and *true* are respectively the greatest and the least element of $\mathcal{C}$; *Var* is a denumerable set of variables and $\exists$ existentially quantifies variables over constraints. The *entailment* $\vdash$ is the inverse of order $\leq$.

Given a cylindric constraint system $\mathbf{C}$ and a set of process symbols Π, the syntax of agents is given by the grammar:

$$A ::= \mathsf{skip} \mid \mathsf{tell}(c) \mid A \parallel A \mid \exists x\, A \mid \textstyle\sum_{i=1}^{n} \mathsf{ask}(c_i) \to A \mid \mathsf{now}\ c\ \mathsf{then}\ A\ \mathsf{else}\ A \mid p(\vec{x})$$

where c, $c_1, \ldots, c_n$ are finite constraints in $\mathbf{C}$; $p_{/m} \in \Pi$ and $\vec{x}$ denotes a generic tuple of variables. A *tccp* program is an object of the form $D\,.\,A$, where A is an agent, called *initial agent*, and D is a set of *process declarations* of the form $p(\vec{x}) :- A$ (for some agent A). The notion of time is introduced by defining a discrete and global clock.

Let us now define the operational semantics of the language with a slight difference from the original one in [dBGM00]. Essentially, to follow the philosophy of computations defined in [SRP91], we stop computations that reach an inconsistent store instead of continuing the computation. The *operational semantics* of *tccp* is formally described by a transition system $T = (\mathit{Conf}, \to)$.[4] Configurations in *Conf* are pairs $\langle A, c\rangle$ representing the agent A to be executed in the current global store c. The transition relation $\to \subseteq \mathit{Conf} \times \mathit{Conf}$ is the least relation satisfying the rules of Fig. 1. Each transition step takes exactly one time-unit.

Informally, the $\mathsf{tell}(c)$ agent adds the constraint c to the store in the next time instant and then stops. The choice agent $\sum_{i=1}^{n} \mathsf{ask}(c_i) \to A_i$ consults the store and non-deterministically executes (at the following time instant) one of the agents A_i whose corresponding guard c_i is entailed by the current store; otherwise, if no guard is entailed by the store, the agent suspends. The conditional agent $\mathsf{now}\ c\ \mathsf{then}\ A\ \mathsf{else}\ B$ behaves in the current time instant like A (respectively B) if c is (respectively is not) entailed by the store. $A \parallel B$ models the parallel composition of A and B in terms of maximal parallelism. The agent $\exists x\, A$ makes variable x local to A. To this end, it uses the $\exists$ operator of the constraint system. Finally, the agent $p(\vec{x})$ takes from D a declaration of the form $p(\vec{x}) :- A$ and then executes A at the following time instant.

[3] See [dBGM00,Sar93] for more details on cylindric constraint systems.
[4] Reviewers can find details in [CTV13a].

$$\frac{}{\langle \mathsf{tell}(c),\, d\rangle \to \langle \mathsf{skip},\, c \otimes d\rangle}\; d \neq \mathit{false} \qquad \textbf{(R1)}$$

$$\frac{}{\langle \sum_{i=1}^{n} \mathsf{ask}(c_i) \to A_i,\, d\rangle \to \langle A_j,\, d\rangle}\; j \in [1,n],\, d \vdash c_j,\, d \neq \mathit{false} \qquad \textbf{(R2)}$$

$$\frac{\langle A,\, d\rangle \to \langle A',\, d'\rangle}{\langle \mathsf{now}\ c\ \mathsf{then}\ A\ \mathsf{else}\ B,\, d\rangle \to \langle A',\, d'\rangle}\; d \vdash c \qquad \textbf{(R3)}$$

$$\frac{\langle A,\, d\rangle \not\to}{\langle \mathsf{now}\ c\ \mathsf{then}\ A\ \mathsf{else}\ B,\, d\rangle \to \langle A,\, d\rangle}\; d \vdash c,\, d \neq \mathit{false} \qquad \textbf{(R4)}$$

$$\frac{\langle B,\, d\rangle \to \langle B',\, d'\rangle}{\langle \mathsf{now}\ c\ \mathsf{then}\ A\ \mathsf{else}\ B,\, d\rangle \to \langle B',\, d'\rangle}\; d \nvdash c \qquad \textbf{(R5)}$$

$$\frac{\langle B,\, d\rangle \not\to}{\langle \mathsf{now}\ c\ \mathsf{then}\ A\ \mathsf{else}\ B,\, d\rangle \to \langle B,\, d\rangle}\; d \nvdash c \qquad \textbf{(R6)}$$

$$\frac{\langle A,\, d\rangle \to \langle A',\, d'\rangle \quad \langle B,\, d\rangle \to \langle B',\, c'\rangle}{\langle A \parallel B,\, d\rangle \to \langle A' \parallel B',\, d' \otimes c'\rangle} \qquad \textbf{(R7)}$$

$$\frac{\langle A,\, d\rangle \to \langle A',\, d'\rangle \quad \langle B,\, d\rangle \not\to}{\begin{array}{c}\langle A \parallel B,\, d\rangle \to \langle A' \parallel B,\, d'\rangle \\ \langle B \parallel A,\, d\rangle \to \langle B \parallel A',\, d'\rangle\end{array}} \qquad \textbf{(R8)}$$

$$\frac{\langle A,\, l \otimes \exists_x d\rangle \to \langle B,\, l'\rangle}{\langle \exists^{l} x\, A,\, d\rangle \to \langle \exists^{l'} x\, B,\, d \otimes \exists_x l'\rangle} \qquad \textbf{(R9)}$$

$$\frac{}{\langle p(\vec{x}),\, d\rangle \to \langle A,\, d\rangle}\; p(\vec{x}) :- A \in D,\, d \neq \mathit{false} \qquad \textbf{(R10)}$$

Fig. 1. The transition system for *tccp*.

Let us introduce a (non trivial) example of *tccp* program that we use through the paper to illustrate the achievements of our proposal.

Example 1 (Guiding example).
Through the paper, we use as guiding example a part of the full specification of a railway crossing system introduced in [AGPV06]. Due to the monotonicity of the store, streams (written in a list-fashion way) are used to model *imperative-style* variables [dBGM00].

```
master(C, G) :- ∃C′, G′ (
    now (C = [near | _]) then
      tell(C = [near | C′]) ∥ tell(G = [down | G′]) ∥ master(C′, G′)
    else  now (C = [out | _]) then
            tell(C = [out | C′]) ∥ tell(G = [up | G′]) ∥ master(C′, G′)
          else master(C, G))
```

The master process uses an *input channel* C (implemented as a stream) through which it receives signals from the environment (trains), and an *output channel* G through which it sends orders to the gate process. It checks the input channel for a *near* signal (the guard in the first now agent), in which case it sends (tells) the order *down* through G, links the future values (C') of the stream C and restarts the check at the following time instant (recursive call $master(C', G')$). If the *near* signal is not detected, then, the else branch looks for the *out* signal and (if present) behaves dually to the first branch. Finally, if no signal is detected at the current time instant (last else branch), then the process keeps checking from the following time instant(the process call takes one time instant). The gate process reacts to the signals from the master:

$$
\begin{aligned}
&gate(G,S) \coloneq \exists G', S' \Big(\\
&\qquad \mathsf{ask}(G = [down \mid \_]) \to \\
&\qquad\quad \big(\mathsf{tell}(G = [down \mid G']) \parallel \mathsf{ask}(true)^{100} \to (\mathsf{tell}(S = [down \mid S']) \parallel gate(G', S'))\big) \\
&\qquad + \mathsf{ask}(G = [up \mid \_]) \to \\
&\qquad\quad \big(\mathsf{tell}(G = [up \mid G']) \parallel \mathsf{ask}(true)^{100} \to (\mathsf{tell}(S = [up \mid S']) \parallel gate(G', S'))\big)\Big)
\end{aligned}
$$

where $\mathsf{ask}(true)^n$ denotes the n-times repetition of the agent $\mathsf{ask}(true)$, and it corresponds to a delay of n time units. Through the input channel G, orders are received, and the state of the gate (represented by the stream S) is consequently updated. The ask agent (with two branches) makes the gate wait (suspend) until one of the guards is entailed, i.e., until one of the two orders is received. Once a signal is detected, after 100 time instants, the state of the gate is appropriately updated. and a recursive call is done in order to keep the gate active (i.e., waiting for the successive order).

In this work, we prove the correctness of our technique w.r.t. the denotational concrete semantics of [CTV13a], which is fully-abstract w.r.t. the small-step behavior of *tccp*. Also csLTL is interpreted over this denotational model. We thus need to introduce (at least) intuitivelythe most relevant aspects of such semantics. The missing definitions, as well as the proofs of all the results, can be found in [CTV13a].

The denotational semantics of a *tccp* program consists of a set of *conditional (timed) traces* that represent, in a compact way, all the possible behaviors that the program can manifest when fed with an *input* (initial store). Intuitively, conditional traces can be seen as hypothetical computations in which, for each time instant, we have a conditional state where each condition represents the information that the global store has to satisfy in order to proceed to the next time instant. Briefly, a conditional trace is a (possibly infinite) sequence $t_1 \cdots t_n \cdots$ of *conditional states*, which can be of three forms:

conditional store: a pair $\eta \rightarrowtail c$, where η is a *condition* and $c \in \mathbf{C}$ a store;
stuttering: the construct $stutt(C)$, with $C \subseteq \mathbf{C} \smallsetminus \{true\}$;
end of a process: the construct $\boxtimes$.

The conditional store $\eta \rightarrowtail c$ means that η must be satisfied by the current store in order to make the computation proceed. c is the information computed up to the current time instant. The conditional store $\eta \rightarrowtail c$ is used to represent a hypothetical computation step, where η is the condition that the current store must satisfy in order to make the computation proceed. c represents the information that is provided by the program up to the current time instant. A *condition* η is a pair $\eta = (\eta^+, \eta^-)$ where $\eta^+ \in \mathbf{C}$ and $\eta^- \in \wp(\mathbf{C})$ are called positive and negative condition, respectively. The positive/negative condition represents information that a given store must/must not entail, thus they have to be consistent in the sense that $\forall c^- \in \eta^- \ \eta^+ \nvdash c^-$.

Due to the partial nature of the constraint system, we cannot use disjunction to represent that *some* given constraints cannot be entailed, thus we have to use a set of constraints for the negative condition. The conditional states of conditional traces are monotone (i.e., for each $t_i = \eta_i \rightarrowtail c_i$ and $t_j = \eta_j \rightarrowtail c_j$ such that $j \geq i$, $c_j \vdash c_i$) and consistent (i.e., each store does not entail the negative conditions of the corresponding conditional state).

We abuse in notation and define as $\bar{\exists}_x r$ the sequence resulting by removing from the conditional trace r all the information about the variable x.

A set of conditional traces C is called *maximal* if no conditional trace is the prefix of another. We denote the domain of *maximal conditional trace sets* as $\mathbb{M}$.$(\mathbb{M}, \sqsubseteq, \bigsqcup, \bigsqcap, \mathbf{M}, \{\epsilon\})$ is a complete lattice, where $M_1 \sqsubseteq M_2 \Leftrightarrow \forall r_1 \in M_1 \, \exists r_2 \in M_2 . \, r_1$ is a prefix of r_2. We denote by $\mathbf{M}$ the top element of $\mathbb{M}$.

We distinguish two special classes of conditional traces. $r \in \mathbf{M}$ is said to be *self-sufficient* if the first condition is $(\mathit{true}, \varnothing)$ and, for each $t_i = (\eta_i^+, \eta_i^-) \rightarrowtail c_i$ and $t_{i+1} = (\eta_{i+1}^+, \eta_{i+1}^-) \rightarrowtail c_{i+1}$, $c_i \vdash \eta_{i+1}^+$ (each store satisfies the successive condition). Moreover, r is *self-sufficient w.r.t.* $x \in \mathit{Var}$ (*x-self-sufficient*) if $\bar{\exists}_{\mathit{Var} \smallsetminus \{x\}} r$ is self-sufficient. Thus, this definition demands that for self-sufficient conditional traces, no additional information (from other agents) is needed in order to *complete* the computation. In an x-self-sufficient conditional trace the same happens but only considering information about variable x.

The semantics definition is based on a semantics evaluation function $\mathcal{A}[\![A]\!]_{\mathcal{I}}$ which, given an agent A and an interpretation $\mathcal{I}$, builds the conditional traces associated to A. The interpretation $\mathcal{I}$ is a function which associates to each process symbol a set of (maximal)conditional traces "modulo variance". The semantics for a set of process declarations D is the fixpoint $\mathcal{F}[\![D]\!] \coloneqq \mathit{lfp}(\mathcal{D}[\![D]\!])$ of the continuous immediate consequences operator $\mathcal{D}[\![D]\!]_{\mathcal{I}}(p(\vec{x})) \coloneqq \bigsqcup_{p(\vec{x}) :- A \in D} \mathcal{A}[\![A]\!]_{\mathcal{I}}$. Proof of full abstraction w.r.t. the operational behavior of *tccp* is given in [CTV13a].

Example 2. Let D_{rc} be the set of declarations of Example 1. Given an interpretation $\mathcal{I}$, the semantics of $\mathit{master}(C, G)$ is graphically represented in Fig. 2, where we have used some shortcuts for characteristic constraints. Namely, $c_{near} \coloneqq (C = [\mathit{near} \mid \_])$ (*near* signal has arrived), $c'_{near} \coloneqq \exists_{C'}(C = [\mathit{near} \mid C'])$ (link of channel C to the *future* channel C'), $c_{down} \coloneqq \exists_{G'}(G = [\mathit{down} \mid G'])$ (delivery of order *down*), $c_{out} \coloneqq (C = [\mathit{out} \mid \_])$, $c'_{out} \coloneqq \exists_{C'}(C = [\mathit{out} \mid C'])$, $c_{up} \coloneqq \exists_{G'}(G = [\mathit{up} \mid G'])$.

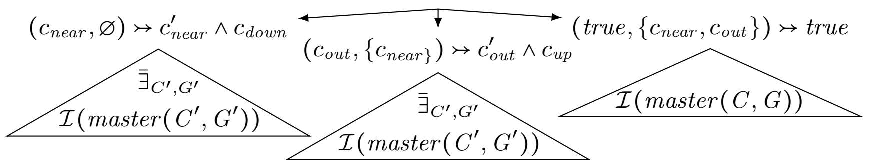
$(c_{near}, \varnothing) \rightarrowtail c'_{near} \wedge c_{down}$
$(c_{out}, \{c_{near}\}) \rightarrowtail c'_{out} \wedge c_{up}$
$(true, \{c_{near}, c_{out}\}) \rightarrowtail true$
$\bar{\exists}_{C',G'}$
$\mathcal{I}(master(C', G'))$
$\bar{\exists}_{C',G'}$
$\mathcal{I}(master(C', G'))$
$\mathcal{I}(master(C, G))$

Fig. 2. Tree representation of $\mathcal{D}[\![D_{rc}]\!]_{\mathcal{I}}(master(C, G))$ of Example 2.

The branch on the left represents the computation when a *near* signal arrives. The first conditional state requires that c_{near} holds, thus the constraints c'_{near} and c_{down} are *concurrently* added to the store during that computational step. A recursive call is also concurrently invoked. Process calls do not modify the store when invoked, but they affect the store from the following time instant, which is graphically represented by the triangle labeled with the interpretation of the process. The branch in the middle is taken only if c_{out} is entailed and c_{near} is not entailed by the initial store (it occurs in the negative condition of the first conditional state in that branch). Finally, the branch on the right represents the recursive call invokedwhen both c_{near} and c_{out} are not entailed by the initial store.

3 Abstract semantics for *tccp* over csLTL formulas

In this section, we present a novel abstract semantics over formulas that approximates the small-step semantics described in Section 2 and, therefore, the observable behavior of the program. To this end, we first define an abstract domain of logic formulas which is a variation of the classical Linear Temporal Logic [MP92]. Following [PV01,dBGM01,dBGM02,Val05], the idea is to replace atomic propositions by constraints of the underlying constraint system.

Definition 1 (csLTL formulas). *Given a cylindric constraint system* $\mathbf{C}$, $c \in \mathbf{C}$ *and* $x \in Var$, *formulas of the* Constraint System Linear Temporal Logic *over* $\mathbf{C}$ *are defined by using the grammar:*

$$\phi ::= \dot{true} \mid \dot{false} \mid c \mid \dot{\neg}\, \phi \mid \phi \mathbin{\dot{\wedge}} \phi \mid \dot{\exists}_x\, \phi \mid \bigcirc \phi \mid \phi\, \mathcal{U}\, \phi.$$

We denote with $\mathsf{csLTL}_{\mathbf{C}}$*the set of all temporal formulas over* $\mathbf{C}$ *(we omit* $\mathbf{C}$ *if clear from the context).*

The formulas $\dot{true}$, $\dot{false}$ and connectives $\dot{\neg}$, $\dot{\wedge}$ have the classical logical meaning. The atomic formula $c \in \mathbf{C}$ states that c has to be entailed by the current store. $\dot{\exists}_x\, \phi$ is the existential quantification over the set of variables Var. $\bigcirc \phi$ states that ϕ holds at the next time instant, while $\phi_1\, \mathcal{U}\, \phi_2$ states that ϕ_2 eventually holds and in all previous instants ϕ_1 holds.In the sequel (as usual), we use $\phi_1 \mathbin{\dot{\vee}} \phi_2$ as a shorthand for $\dot{\neg}\, \phi_1 \mathbin{\dot{\wedge}} \dot{\neg}\, \phi_2$; $\phi_1 \mathbin{\dot{\rightarrow}} \phi_2$ for $\dot{\neg}\, \phi_1 \mathbin{\dot{\vee}} \phi_2$; $\phi_1 \mathbin{\dot{\leftrightarrow}} \phi_2$ for $\phi_1 \mathbin{\dot{\rightarrow}} \phi_2 \mathbin{\dot{\wedge}} \phi_2 \mathbin{\dot{\rightarrow}} \phi_1$; $\Diamond \phi$ for $\dot{true}\, \mathcal{U}\, \phi$ and $\Box \phi$ for $\dot{\neg} \Diamond \dot{\neg} \phi$. $\phi_1\, \mathcal{W}\, \phi_2$ for $(\phi_1\, \mathcal{U}\, \phi_2) \mathbin{\dot{\vee}} \Box \phi_1$. $\Diamond \phi$ holds if at some point in the future ϕ is true, and $\Box\,\phi$ holds if ϕ holds in the current instant and always in the future.

A *constraint formula* is an atomic formula c or its negation $\dot{\neg}\,c$. Formulas of the form $\bigcirc\,\phi$ and $\dot{\neg}\bigcirc\,\phi$ are called *next* formulas. Constraint and next formulas are said to be *elementary* formulas. Finally, formulas of the form $\phi_1\;\mathcal{U}\;\phi_2$ (or $\Diamond\,\phi$ or $\dot{\neg}(\Box\,\phi)$) are called *eventualities.*

We define the abstract domain $\mathbb{F} := \mathsf{csLTL}/_{\dot{\leftrightarrow}}$ (i.e., the domain formed by csLTL formulas modulo logical equivalence) ordered by $\dot{\rightarrow}$. The algebraic structure $(\mathbb{F}, \dot{\rightarrow}, \dot{\bigvee}, \dot{\bigwedge}, \dot{true}, \dot{false})$ is a (bounded) lattice but not a complete lattice, since both $\dot{\bigwedge}$ and $\dot{\bigvee}$ always exist just for finite sets of formulas.

The semantics of a temporal formula is typically defined in terms of an infinite sequence of states which validates it. Here we use conditional traces instead. As usually done in the context of temporal logics, we define the satisfaction relation $\vDash$ only for infinite conditional traces. We implicitly transform finite traces (which end in $\boxtimes$) by replicating the last store infinite times. Namely, the trace $(\eta_1^+,\eta_1^-) \rightarrowtail c_1 \dots (\eta_n^+,\eta_n^-) \rightarrowtail c_n \cdot \boxtimes$ becomes $(\eta_1^+,\eta_1^-) \rightarrowtail c_1 \dots (\eta_n^+,\eta_n^-) \rightarrowtail c_n \cdot (c_n,\varnothing) \rightarrowtail c_n \cdots (c_n,\varnothing) \rightarrowtail c_n \cdots$, while $\boxtimes$ becomes $(true,\varnothing) \rightarrowtail true \cdots (true,\varnothing) \rightarrowtail true \cdots$.

Definition 2. *The* semantics *of $\phi \in \mathbb{F}$ is given by function $\gamma^{\mathbb{F}}\colon \mathbb{F} \to \mathbb{M}$ defined as*

$$\gamma^{\mathbb{F}}(\phi) := \bigsqcup\{r \in \mathbf{M} \mid r \vDash \phi\}, \tag{3.1}$$

where, for each $\phi,\phi_1,\phi_2 \in$ csLTL, $c \in \mathbf{C}$ and $r \in \mathbf{M}$, satisfaction relation $\vDash$ is defined as:

$$r \vDash \dot{true} \quad and \quad r \nvDash \dot{false} \tag{3.2a}$$
$$(\eta^+,\eta^-) \rightarrowtail d \cdot r' \vDash c \qquad iff\ \eta^+ \vdash c \tag{3.2b}$$
$$stutt(\eta^-) \cdot r' \vDash c \qquad iff\ \forall d^- \in \eta^-.\, c \nvdash d^-\ and\ r' \vDash c \tag{3.2c}$$
$$r \vDash \dot{\neg}\,\phi \qquad iff\ r \nvDash \phi \tag{3.2d}$$
$$r \vDash \phi_1 \mathbin{\dot{\wedge}} \phi_2 \qquad iff\ r \vDash \phi_1\ and\ r \vDash \phi_2 \tag{3.2e}$$
$$r \vDash \bigcirc\,\phi \qquad iff\ r^1 \vDash \phi \tag{3.2f}$$
$$r \vDash \phi_1\,\mathcal{U}\,\phi_2 \qquad iff\ \exists i \geq 1.\, \forall j < i.\ r^i \vDash \phi_2\ and\ r^j \vDash \phi_1 \tag{3.2g}$$
$$r \vDash \dot{\exists}_x\,\phi \qquad iff\ exists\ r'\ s.t.\ \bar{\exists}_x\, r' = \bar{\exists}_x\, r,\ r'\ x\text{-}self\text{-}sufficient\ and\ r' \vDash \phi \tag{3.2h}$$

We say that $\phi \in \mathbb{F}$ is a sound approximation *of $R \in \mathbb{M}$ if $R \sqsubseteq \gamma^{\mathbb{F}}(\phi)$.*
By abusing of notation, we extend the notion of $\vDash$ to sets of formulas in the following way

$$r \vDash \Phi \iff \forall \phi \in \Phi.\ r \vDash \phi \tag{3.3}$$

A formula ϕ is said to be satisfiable *if there exists $r \in \mathbf{M}$ such that $r \vDash \phi$, while it is said to be* valid *if, for all $r \in \mathbf{M}$, $r \vDash \phi$.*

[4] r^k denotes the sub-sequence of r starting from state k.

All the cases are fairly standard except (3.2b) and (3.2c). The conditional trace $r = (\eta^+, \eta^-) \rightarrowtail d \cdot r'$ prescribes that η^+ is entailed by the current store, thus r models all the constraint formulas c such that $\eta^+ \vdash c$. We have to note that, by the monotonicity of the store of *tccp* computations, the positive conditions in conditional traces contains all the information previously added in the constraint store. Furthermore, by the definition of condition, since η^+ cannot be in contradiction with η^-, it holds that neither c is in contradiction with η^-. Thus, the conditional trace $\mathit{stutt}(\eta^-) \cdot r'$ models all the constraint formulas c that are not in contradiction with the set η^- and such that c holds in the continuation r' by monotonicity.

Lemma 1. *The function $\gamma^{\mathbb{F}}$ is monotonic, injective and $\bigsqcap$-distributive.*

3.1 csLTL Abstract Semantics

The technical core of our semantics definition is the csLTL agent semantics evaluation function $\hat{\mathcal{A}}[\![A]\!]$ which, given an agent A and an interpretation $\hat{\mathcal{I}}$ (for the process symbols of A), builds a csLTL formula which is a sound approximation of the (concrete) behavior of A. In the sequel, we denote by $\mathbb{A}^{\Pi}_{\mathbf{C}}$ the set of agents and $\mathbb{D}^{\Pi}_{\mathbf{C}}$ the set of sets of process declarations built on signature Π and constraint system $\mathbf{C}$.

Definition 3. *Let $\mathbb{PC} \coloneqq \{p(\vec{x}) \mid p \in \Pi, \vec{x}$ are distinct variables$\}$. An* $\mathbb{F}$-interpretation *is a function $\mathbb{PC} \to \mathbb{F}$ modulo variance*[5]*. Two functions $I, J : \mathbb{PC} \to \mathbb{F}$ are* variants *if for each $\pi \in \mathbb{PC}$ there exists a renaming ρ such that $(I\pi)\rho = J(\pi\rho)$. The semantic domain $\mathbb{I}_{\mathbb{F}}$ is the set of all $\mathbb{F}$-interpretations ordered by the point-wise extension of $\dot{\to}$.*

Definition 4 (csLTL Semantics). *Given $A \in \mathbb{A}^{\Pi}_{\mathbf{C}}$ and $\hat{\mathcal{I}} \in \mathbb{I}_{\mathbb{F}}$, we define the* csLTL semantics evaluation *$\hat{\mathcal{A}}[\![A]\!]_{\hat{\mathcal{I}}}$ by structural induction as follows.*

$$\hat{\mathcal{A}}[\![\mathsf{skip}]\!]_{\hat{\mathcal{I}}} \coloneqq \mathit{true} \tag{3.4a}$$

$$\hat{\mathcal{A}}[\![\mathsf{tell}(c)]\!]_{\hat{\mathcal{I}}} \coloneqq \bigcirc c \tag{3.4b}$$

$$\hat{\mathcal{A}}[\![\textstyle\sum_{i=1}^{n} \mathsf{ask}(c_i) \to A_i]\!]_{\hat{\mathcal{I}}} \coloneqq \square(\dot{\bigwedge}_{i=1}^{n} \dot{\neg}\, c_i) \mathbin{\dot{\vee}} \Big((\dot{\bigwedge}_{i=1}^{n} \dot{\neg}\, c_i) \,\mathcal{U}\, \dot{\bigvee}_{i=1}^{n} \big(c_i \mathbin{\dot{\wedge}} \bigcirc \hat{\mathcal{A}}[\![A_i]\!]_{\hat{\mathcal{I}}}\big)\Big) \tag{3.4c}$$

$$\hat{\mathcal{A}}[\![\mathsf{now}\ c\ \mathsf{then}\ A_1\ \mathsf{else}\ A_2]\!]_{\hat{\mathcal{I}}} \coloneqq (c \mathbin{\dot{\wedge}} \hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}}) \mathbin{\dot{\vee}} (\dot{\neg}\, c \mathbin{\dot{\wedge}} \hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}}) \tag{3.4d}$$

$$\hat{\mathcal{A}}[\![A_1 \parallel A_2]\!]_{\hat{\mathcal{I}}} \coloneqq \hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}} \mathbin{\dot{\wedge}} \hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}} \tag{3.4e}$$

$$\hat{\mathcal{A}}[\![\exists x\, A]\!]_{\hat{\mathcal{I}}} \coloneqq \dot{\exists}_x\, \hat{\mathcal{A}}[\![A]\!]_{\hat{\mathcal{I}}} \tag{3.4f}$$

$$\hat{\mathcal{A}}[\![p(\vec{x})]\!]_{\hat{\mathcal{I}}} \coloneqq \bigcirc \hat{\mathcal{I}}(p(\vec{x})) \tag{3.4g}$$

Given $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$ we define the immediate consequence operator $\hat{\mathcal{D}}[\![D]\!] : \mathbb{I}_{\mathbb{F}} \to \mathbb{I}_{\mathbb{F}}$ as

$$\hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{I}}}(p(\vec{x})) \coloneqq \dot{\bigvee} \left\{ \hat{\mathcal{A}}[\![A]\!]_{\hat{\mathcal{I}}} \,\middle|\, p(\vec{x}) \coloneq A \in D \right\}$$

[5] i.e., a family of elements of $\mathbb{F}$, indexed by $\mathbb{PC}$, modulo variance.

It is straightforward to notice that $\hat{\mathcal{A}}[\![A]\!]$ and $\hat{\mathcal{D}}[\![D]\!]$ are monotonic.

Example 3. Consider the set of declarations D_{rc} of Example 1 and let us use $\bigcirc^n$ to abbreviate the repetition of $\bigcirc$ n-times. Given $\hat{\mathcal{I}} \in \mathbb{I}_{\mathbb{F}}$, with Definition 4 we compute

$$\begin{aligned}
&\phi_M(\hat{\mathcal{I}}) := \hat{\mathcal{D}}[\![D_{rc}]\!]_{\hat{\mathcal{I}}}(master(C,G))) = \phi_{near}(\hat{\mathcal{I}}) \dot{\vee} \phi_{out}(\hat{\mathcal{I}}) \dot{\vee} \phi_{cwait}(\hat{\mathcal{I}}) \\
&\phi_g(\hat{\mathcal{I}}) := \hat{\mathcal{D}}[\![D_{rc}]\!]_{\hat{\mathcal{I}}}(gate(G,S))\big(\phi_{gwait}\,\mathcal{U}\,(\phi_{down}(\hat{\mathcal{I}}) \dot{\vee} \phi_{up}(\hat{\mathcal{I}}))\big) \dot{\vee} \Box \phi_{gwait}
\end{aligned}$$

where

$$\begin{aligned}
\phi_{near}(\hat{\mathcal{I}}) = \dot{\exists}_{C',G'} \big(&C = [near \mid \_] \dot{\wedge} \bigcirc C = [near \mid C'] \dot{\wedge} \\
&\bigcirc G = [down \mid G'] \dot{\wedge} \bigcirc \hat{\mathcal{I}}(master(C',G'))\big) \\
\phi_{out}(\hat{\mathcal{I}}) = \dot{\exists}_{C',G'} \big(&\dot{\neg}(C = [near \mid \_]) \dot{\wedge} \bigcirc C = [out \mid C'] \dot{\wedge} \\
&C = [out \mid \_] \dot{\wedge} \bigcirc G = [up \mid G'] \dot{\wedge} \bigcirc \hat{\mathcal{I}}(master(C',G'))\big) \\
\phi_{cwait}(\hat{\mathcal{I}}) = \;&\dot{\neg}(C = [near \mid \_]) \dot{\wedge} \dot{\neg}(C = [out \mid \_]) \dot{\wedge} \bigcirc \hat{\mathcal{I}}(master(C,G)) \\
\phi_{gwait} = \;&\dot{\neg}(G = [down \mid \_]) \dot{\wedge} \dot{\neg}(G = [up \mid \_]) \\
\phi_{down}(\hat{\mathcal{I}}) = \dot{\exists}_{G',S'} \big(&G = [down \mid \_] \dot{\wedge} \bigcirc \big(\bigcirc G = [down \mid S'] \dot{\wedge} \\
&\bigcirc^{100}(\bigcirc S = [down \mid S'] \dot{\wedge} \bigcirc \hat{\mathcal{I}}(gate(G',S'))))\big) \\
\phi_{up}(\hat{\mathcal{I}}) = \dot{\exists}_{G',S'} \big(&G = [up \mid \_] \dot{\wedge} \bigcirc \big(\bigcirc G = [up \mid G'] \dot{\wedge} \\
&\bigcirc^{100}(\bigcirc S = [up \mid S'] \dot{\wedge} \bigcirc \hat{\mathcal{I}}(gate(G',S'))))\big)
\end{aligned}$$

The three disjoints of $\phi_M(\hat{\mathcal{I}})$ match the three possible behaviors of $master(C,G)$: when signal *near* is emitted by the train ($\phi_{near}(\hat{\mathcal{I}})$), when *out* is emitted ($\phi_{out}(\hat{\mathcal{I}})$), and when no signal arrives ($\phi_{cwait}(\hat{\mathcal{I}})$).

Similarly, the formula $\phi_g(\hat{\mathcal{I}})$ states that, either the process waits forever, or when a signal is received, then it changes the state of the gate ($\phi_{down}(\hat{\mathcal{I}})$ and $\phi_{up}(\hat{\mathcal{I}})$).

We have that $\hat{\mathcal{A}}$ is a sound approximation of $\mathcal{A}$ and $\hat{\mathcal{D}}$ is a sound approximation of $\mathcal{D}$.

Theorem 1 (Correctness of $\hat{\mathcal{A}}$ and $\hat{\mathcal{D}}$). *Let $A \in \mathbb{A}^{\Pi}_{\mathbf{C}}$, $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$ and $\hat{\mathcal{I}} \in \mathbb{I}_{\mathbb{F}}$. Then, $\mathcal{A}[\![A]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})} \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![A]\!]_{\hat{\mathcal{I}}})$ and $\mathcal{D}[\![D]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})} \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{I}}})$.*

4 Abstract diagnosis of *tccp* with csLTL formulas

Since $\mathbb{F}$ is just a bounded lattice, it is impossible to find for the function $\gamma^{\mathbb{F}}$ an adjoint function α which forms a Galois Connection $\langle\alpha, \gamma\rangle$, and therefore we cannot use the abstract diagnosis framework for *tccp* defined in [CTV11] (which is actuallyparametric w.r.t. a Galois Insertion $\langle\alpha, \gamma\rangle$). Thus, we propose in this section a new weaker version of abstract diagnosis that works on $\mathbb{F}$ and which is defined using just $\gamma^{\mathbb{F}}$ [6].

Given a set of declarations D and $\hat{\mathcal{S}} \in \mathbb{I}_{\mathbb{F}}$, which is the specification of the abstract intended behavior of D over $\mathbb{F}$, we say that

1. D is (abstractly) *partially correct* w.r.t. $\hat{\mathcal{S}}$ if $\mathcal{F}[\![D]\!] \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{S}})$.
2. D is (abstractly) *complete* w.r.t. $\hat{\mathcal{S}}$ if $\gamma^{\mathbb{F}}(\hat{\mathcal{S}}) \sqsubseteq \mathcal{F}[\![D]\!]$.

The differences between $\mathcal{F}[\![D]\!]$ and $\gamma^{\mathbb{F}}(\hat{\mathcal{S}})$ are usually called *symptoms*. Many of the symptoms are just a consequence of some "originating" ones, those which are the direct consequence of errors. The *abstract diagnosis* determines exactly the "originating" symptoms and, in the case of incorrectness, the faulty process declarations in D. This is captured by the definitions of *abstractly incorrect process declaration* and *abstract uncovered element*:[7]

Definition 5. *Let $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$, R a process declaration for process p, $\phi_t \in \mathbb{F}$ and $\hat{\mathcal{S}} \in \mathbb{I}_{\mathbb{F}}$.*

- *R is* abstractly incorrect *w.r.t. $\hat{\mathcal{S}}$ (on testimony ϕ_t) if $\phi_t \dot{\to} \hat{\mathcal{D}}[\![\{R\}]\!]_{\hat{\mathcal{S}}}(p(\vec{x}))$ and $\phi_t \dot{\wedge} \hat{\mathcal{S}}(p(\vec{x})) = \dot{false}$.*
- *ϕ_t is an* uncovered element *for $p(\vec{x})$ w.r.t. $\hat{\mathcal{S}}$ if $\phi_t \dot{\to} \hat{\mathcal{S}}(p(\vec{x}))$ and $\phi_t \dot{\wedge} \hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{S}}}(p(\vec{x})) = \dot{false}$.*

Informally, R is abstractly incorrect if it derives a wrong abstract element ϕ_t from the intended semantics. Dually, ϕ_t is uncovered if the declarations cannot derive it from the intended semantics.

Theorem 2. *Let $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$ and $\hat{\mathcal{S}} \in \mathbb{I}_{\mathbb{F}}$.*

1. *If there are no abstractly incorrect process declarations in D (i.e., $\hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{S}}} \dot{\to} \hat{\mathcal{S}}$), then D is partially correct w.r.t. $\hat{\mathcal{S}}$.*
2. *Let D be partially correct w.r.t. $\hat{\mathcal{S}}$. If D has abstract uncovered elements then D is not complete.*

Absence of abstractly incorrect declarations is a sufficient condition for partial correctness, but it is not necessary. Because of the approximation, it can happen that a (concretely) correct declaration is abstractly incorrect. Hence, abstract incorrect declarations are in general just a warning about a possible source of errors. However, an abstract correct declaration cannot contain an error; thus, no (manual) inspection is needed for declarations which are not abstractly incorrect. Moreover, as shown by the following theorem, all concrete errors—that are "visible"—are indeed detected, as they lead to an abstract incorrectness or

[6] Actually, the proposal could be defined parametrically w.r.t. a suitable family of concretization functions. However, for the sake of simplicity, we formulate just the specific instance for $\gamma^{\mathbb{F}}$.

[7] It is worth noticing that although the notions defined in this section are similar to those defined for the standard approach, the formal definitions and proofs are different due to the weaker framework.

abstract uncovered. Intuitively, in this setting,a concrete error is *visible* if it is possible to express a formula ϕ whose concretization reveals the error (i.e., if the logic is expressive enough).

Theorem 3. *Let R be a process declaration for $p(\vec{x})$, $\mathcal{S}$ a concrete specification and $\hat{\mathcal{S}}$ a sound approximation for $\mathcal{S}$ (i.e., $\mathcal{S} \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{S}})$).*

1. *If $\mathcal{D}[\![\{R\}]\!]_{\mathcal{S}} \not\sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{S}})$ and it exists ϕ_t such that $\gamma^{\mathbb{F}}(\phi_t) \sqsubseteq \mathcal{D}[\![\{R\}]\!]_{\mathcal{S}}(p(\vec{x}))$ and $\phi_t \mathbin{\dot\wedge} \hat{\mathcal{S}}(p(\vec{x})) = false$, then R is abstractly incorrect w.r.t. $\hat{\mathcal{S}}$ (on testimony ϕ_t).*
2. *If there exists an abstract uncovered element ϕ w.r.t. $\hat{\mathcal{S}}$, then there exists $r \in \gamma^{\mathbb{F}}(\phi)$ such that $r \notin \mathcal{D}[\![\{R\}]\!]_{\mathcal{S}}(p(\vec{x}))$.*

Point 1 can be read as:the concrete error has an abstract symptom which is not hidden by the approximation on $\hat{\mathcal{S}}$ and, moreover, there exists a formula ϕ_t which can express it.

In the following examples, we borrow from [AGPV06] the notation for *last entailed value* of a stream: $X \dot{=} c$ holds if the last instantiated value in the stream X is c.

Example 4. We verify (for Example 1) that each time a *near* signal arrives from a train, the gate eventually is down. To model this property, we define the specification (of the property) $\hat{\mathcal{S}}_{down}$ as

$$\begin{aligned}
&\phi_{ordersent} \coloneqq \hat{\mathcal{S}}_{down}(master(C,G)) \coloneqq \Box(C \dot{=} near \mathbin{\dot\rightarrow} \Diamond(G \dot{=} down))\\
&\phi_{gatedown} \coloneqq \hat{\mathcal{S}}_{down}(gate(G,S)) \coloneqq \Box(G \dot{=} down \mathbin{\dot\rightarrow} \Diamond(S \dot{=} down))
\end{aligned}$$

To check whether the program implies the specification ($\hat{\mathcal{D}}[\![D_{rc}]\!]_{\hat{\mathcal{S}}_{down}} \mathbin{\dot\rightarrow} \hat{\mathcal{S}}_{down}$) (see Example 3) we have to check if $\phi_M(\hat{\mathcal{S}}_{down}) \mathbin{\dot\rightarrow} \phi_{ordersent}$ and $\phi_g(\hat{\mathcal{S}}_{down}) \mathbin{\dot\rightarrow} \phi_{gatedown}$. Recall the fixpoint characterization of the temporal operators, i.e., $\Box p = p \mathbin{\dot\wedge} \bigcirc \Box p$ and $\Diamond p = p \mathbin{\dot\vee} \bigcirc \Diamond p$. It can be seen that each of the three disjoints of $\phi_M(\hat{\mathcal{S}}_{down})$, see Example 3, implies $\phi_{ordersent}$. For $\phi_g(\hat{\mathcal{S}}_{down})$, while the process is waiting, the antecedent of both implications cannot be derived, thus the formula is true. Moreover, when the order down arrives (in the second component of the until), it also occurs that the state is updated (see ϕ_{down} in Example 3). Thus, $\phi_g(\hat{\mathcal{S}}_{down}) \mathbin{\dot\rightarrow} \hat{\mathcal{S}}_{down}$ and then $\hat{\mathcal{D}}[\![D_{rc}]\!]_{\hat{\mathcal{S}}_{down}} \mathbin{\dot\rightarrow} \hat{\mathcal{S}}_{down}$. Thus, by Theorem 2, D_{rc} is partially correct w.r.t. $\hat{\mathcal{S}}_{down}$.

Example 5. We can verify that it is not valid that eventually in the future either the order *up* or *down* is sent by the gate. This is a warning about a possible error in the definition of gate process w.r.t. the specification $\hat{\mathcal{S}}_w$. We define the specification:

$$\phi_{updown} \coloneqq \hat{\mathcal{S}}_w(gate(G,S)) \coloneqq \Diamond(G \dot{=} up \mathbin{\dot\vee} G \dot{=} down)$$

Since

$$\phi_g(\hat{\mathcal{S}}_w) \coloneqq \hat{\mathcal{D}}[\![D_{rc}]\!]_{\hat{\mathcal{S}}_w}(gate(G,S)) = (\phi_{gwait}\, \mathcal{U}\, (\phi_{down}(\hat{\mathcal{S}}_w) \mathbin{\dot\vee} \phi_{up}(\hat{\mathcal{S}}_w))) \mathbin{\dot\vee} \Box\, \phi_{gwait}$$

then $\phi_g(\hat{\mathcal{S}}_w) \mathbin{\dot{\not\rightarrow}} \phi_{updown}$ (since $\Box\, \phi_{gwait} = \Box(\dot\neg(G \dot{=} up) \mathbin{\dot\wedge} G \dot{=} down)$ does not imply $\Diamond(G \dot{=} up \mathbin{\dot\vee} G \dot{=} down)$).

When the check of a process declaration R agains a specification S fails, then our method reports that R is not partially correct w.r.t. the specification.

Example 6. Now we show how our technique detects an error in a buggy set of declarations obtained from D_{rc} by removing instruction $\mathsf{tell}(G = [up \mid G'])$ in the definition of process master. To avoid misunderstandings, we call the modified process $master'$ and let R be the new set of declarations.

We aim to verify that the order *up* is sent whenever the signal *out* is received. Thus, we define the (property) specification:

$$\phi := \hat{\mathcal{S}}_{up}(master'(C, G)) := \square((C \dot{=} out) \dot{\rightarrow} \diamond(G \dot{=} up))$$

We need to compute the (one step) semantics for the (buggy version of the) process:

$$\phi' := \hat{\mathcal{D}}[\![\{R\}]\!]_{\hat{\mathcal{S}}_{up}}(master'(C, G)) = \phi'_{near} \dot{\vee} \phi'_{out} \dot{\vee} \phi'_{cwait}$$

where

$$\begin{aligned}
\phi'_{near} &:= \dot{\exists}_{C',G'} \big(C = [near \mid \_] \dot{\wedge} \bigcirc C = [near \mid C'] \dot{\wedge} \\
&\qquad\qquad \bigcirc G = [down \mid G'] \dot{\wedge} \bigcirc \hat{\mathcal{S}}_{up}(master'(C', G'))\big) \\
\phi'_{out} &:= \dot{\exists}_{C',G'} \big(\dot{\neg}(C = [near \mid \_]) \dot{\wedge} C = [out \mid \_] \dot{\wedge} \\
&\qquad\qquad \bigcirc(C = [out \mid C'] \dot{\wedge} \bigcirc \hat{\mathcal{S}}_{up}(master'(C', G')))\big) \\
\phi'_{cwait} &:= \dot{\neg}(C = [near \mid \_]) \dot{\wedge} \dot{\neg}(C = [out \mid \_]) \dot{\wedge} \bigcirc \hat{\mathcal{S}}_{up}(master'(C, G))
\end{aligned}$$

We detect an incorrectness of R (in $master'$ process) w.r.t. $\hat{\mathcal{S}}_{up}$ on testimony ϕ'_{out} since $\phi'_{out} \dot{\rightarrow} \phi'$ and $\phi'_{out} \dot{\wedge} \phi = \dot{false}$.

When we deal with programs with loops that do not produce contributes at all (these are in some sense non meaningful programs), then our technique shows a negative phenomenon, which in general happens for sets of declarations D where $\hat{\mathcal{D}}[\![D]\!]$ has more than one fixpoint. In such situation, we can have that the actual behavior does not model a specification $\hat{\mathcal{S}}$ which is a non-least fixpoint of $\hat{\mathcal{D}}[\![D]\!]$, but, since $\hat{\mathcal{S}}$ is a fixpoint, we do not detect abstractly incorrect declarations.

Example 7 (Pathological cases). Let $D_p := \{q(y) \text{ :- } \mathsf{now}\ y = 1\ \mathsf{then}\ q(y)\ \mathsf{else}\ q(y)\}$. It is worth noticing that this program is a loop that does nothing at all since, independently upon the check if $x = 1$, it calls itself. the property to be checked $\hat{\mathcal{S}}_p(q(y)) := \diamond(y = 1)$. Then,

$$\hat{\mathcal{D}}[\![D_p]\!]_{\hat{\mathcal{S}}_p}(q(y)) = (y = 1 \dot{\wedge} \diamond x = 1) \dot{\vee} (\dot{\neg} y = 1 \dot{\wedge} \diamond x = 1)$$

We can see that $\hat{\mathcal{D}}[\![D_p]\!]_{\hat{\mathcal{S}}_p} \dot{\rightarrow} \diamond(y = 1)$, thus and, by Theorem 2, D_p is partially correct w.r.t. $\hat{\mathcal{S}}_p$. However, it can be noticed that $y = 1$ is not explicitly added by the process $q(y)$.

	α	$A(\alpha)$
R1	$\dot\neg\,\dot\neg\,\phi$	$\{\phi\}$
R2	$\phi_1 \mathbin{\dot\wedge} \phi_2$	$\{\phi_1, \phi_2\}$
R3	$\dot\neg \bigcirc \phi$	$\{\bigcirc \dot\neg\, \phi\}$

	β	$B_1(\beta)$	$B_2(\beta)$
R4	$\dot\neg(\phi_1 \mathbin{\dot\wedge} \phi_2)$	$\{\dot\neg\,\phi_1\}$	$\{\dot\neg\,\phi_2\}$
R5	$\dot\neg(\phi_1 \,\mathcal{U}\, \phi_2)$	$\{\dot\neg\,\phi_1, \dot\neg\,\phi_2\}$	$\{\phi_1, \dot\neg\,\phi_2, \dot\neg \bigcirc(\phi_1 \,\mathcal{U}\, \phi_2)\}$
R6	$\phi_1 \,\mathcal{U}\, \phi_2$	$\{\phi_2\}$	$\{\phi_1, \dot\neg\,\phi_2, \bigcirc(\phi_1 \,\mathcal{U}\, \phi_2)\}$
R7	$\phi_1 \,\mathcal{U}\, \phi_2$	$\{\phi_2\}$	$\{\phi_1, \dot\neg\,\phi_2, \bigcirc((\mathit{\Gamma}^* \mathbin{\dot\wedge} \phi_1) \,\mathcal{U}\, \phi_2)\}$

Fig. 3. α- and β-formulas rules

Note that, if $\hat{\mathcal{S}}(p(\vec{x}))$ is assumed to hold for each process $p(\vec{x})$ defined in D and $\hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{S}}} \mathbin{\dot\rightarrow} \hat{\mathcal{S}}$, then $\mathcal{F}[\![D]\!]$ satisfies $\hat{\mathcal{S}}$.

To conclude this section, we would like to point out that with our method have validated/unvalidated all the properties of systems already present in the *tccp* literature.

4.1 An automatic decision procedure for csLTL

In order to make our abstract diagnosis approach effective, we have defined an automatic decision procedure to check the validity of the csLTL formulas that are involved in Definition 5 (of the form $\psi \mathbin{\dot\rightarrow} \phi$ with $\phi = \hat{\mathcal{S}}(p(\vec{x}))$ and $\psi = \hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{S}}}(p(\vec{x})))$. In this section, we obtain such decision procedure by adaptingto csLTL the tableau construction for Propositional LTL (PLTL) of [GHLN08,GHL+09]. [CTV13b] contains a preliminary version of the method. The presented results work only for specifications without existential quantifications. Actually, this restriction is quite irrelevant in our context since, in general, we are interested in proving properties related to the *visible* behavior of the program, not to the local variables.

Intuitively, a tableau construction builds a tree whose nodes are labeled with sets of formulas. The root is labeled with the set of formulas which has to be checked for satisfiability. Branches are built according to rules defined on the syntax of formulas (see Fig. 3 defining α and β formulas). If all leaves of the tree are *closed*, then the formula has no models. Otherwise, we can obtain a model that satisfies the formula from the *open* leaves.

A tableau rule is applied to a node n labeled with the set of formulas $L(n)$. Each rule application requires the selection of a formula from $L(n)$. We call context the set of formulas $L(n) \setminus \{\phi\}$ and we denote it with $\mathit{\Gamma}$. Conjunctions are α-formulas and disjunctions β-formulas. Fig. 3 presents the rules for $\alpha-$ and $\beta-$formulas. Tables in Fig. 3 are interpreted as follows. Each row in a table represents a rule. Each time that an $\alpha-$rule is applied to a node of the tree, a formula of the node matching the pattern in column α is replaced in a child node by the corresponding $A(\alpha)$. For the β-rules, two children nodes are generated, one for each column $B_1(\beta)$ and $B_2(\beta)$.

Almost all the rules are standard. However, Rule R7 uses the so-called context $\mathit{\Gamma}^*$, which is defined as $\mathit{\Gamma}^* := \dot\bigvee_{\gamma \in \mathit{\Gamma}} \dot\neg\,\gamma$. The use of contexts is the mechanism to detect the loops that allows one to mark branches containing eventuality formulas as *open*. This kind of rules were first introduced in [GHL+07]. The idea is that, by using contexts, loops where no formula changes are *discarded* since they cannot close a branch.

Note that there is no rule defined for the $\bigcirc$ operator. In fact, the $\mathsf{next}(\Phi)$ function transforms a set of elementary formulas Φ into another: $\mathsf{next}(\Phi) := \{\phi \mid \bigcirc\phi \in \Phi\} \cup \{\dot{\neg}\phi \mid \dot{\neg}\bigcirc\phi \in \Phi\} \cup \{c \mid c \in \Phi, c \in \mathbf{C}\}$. This operator is different from the corresponding one of PLTL in that, in addition to keeping the internal formula of the next formulas, it also *passes* the constraints that are entailed at the current time instant to the following one. This makes sense for *tccp* computations since, as already mentioned, the store in a computation is monotonic, thus no information can be removed and it happens that, always, c implies $\bigcirc c$. The next operator is a key notion in the kind of tableaux defined in [GHLN08,GHL+09]. This operator allows one to identify *stages* in a tableaux which represent time instants in the model. Some additional checks (explained in the following sections) allow to avoid the use of auxiliary graph representations for determining satisfiability/unsatisfiability of formulas.

A second main difference w.r.t. the PLTL case regards the existential quantification. The csLTL existential quantification does not correspond to the first-order logic one. It is introduced to model information about local variables, thus, the formula $\dot{\exists}_x\,\phi$ can be seen as the formula ϕ where the information about x is local.

We define a specific rule for the $\dot{\exists}$ case. In particular, when the selected formula of a given node is of the form $\dot{\exists}_x\,\phi$, it is created a node, child of n, whose labeling is that of n except that the formula $\dot{\exists}_x\,\phi$ is replaced by ϕ.

In this section, we present a tableau method for our csLTL formulas following the ideas of [GHLN08,GHL+09].

4.2 Semantic csLTL tableaux

Definition 6 (csLTL tableau). *A csLTL tableau for a finite set of formulas Φ is a tuple $\mathcal{T}_\Phi = (Nodes, n_\Phi, L, B, R\,)$ such that:*

1. *Nodes is a finite non-empty set of nodes;*
2. *$n_\Phi \in Nodes$ is the initial node;*
3. *$L : Nodes \to \wp(\mathsf{csLTL})$ is the labeling function that associates to each node the formulas which are true in that node; the initial node is labeled with Φ;*
4. *B is the set of branches such that exactly one of the following points holds for every branch $b = n_0, \dots, n_i, n_{i+1}, \dots, n_k \in B$ and every $0 \le i < k$:*
   (a) for an α-formula $\alpha \in L(n_i)$, $L(n_{i+1}) = \{A(\alpha)\} \cup L(n_i) \setminus \{\alpha\}$;
   (b) for a β-formula $\beta \in L(n_i)$, $L(n_{i+1}) = \{B_1(\beta)\} \cup L(n_i) \setminus \{\beta\}$ and there exists another branch in B of the form $b' = n_0, \dots, n_i, n'_{i+1}, \dots, n'_k$ such that $L(n'_{i+1}) = \{B_2(\beta)\} \cup L(n_i) \setminus \{\beta\}$;
   (c) for an existential quantified formula $\dot{\exists}_x\,\phi' \in L(n_i)$, $L(n_{i+1}) = \{\phi''\} \cup L(n_i) \setminus \{\dot{\exists}_x\,\phi'\}$ where $\phi'' := \phi'[y/x]$ with y fresh variable;
   (d) in case $L(n_i)$ is a set formed only by elementary formulas, $L(n_{i+1}) = \mathsf{next}(L(n_i))$, where $\mathsf{next}(\Phi) := \{\phi \mid \bigcirc\phi \in \Phi\} \cup \{\dot{\neg}\phi \mid \dot{\neg}\bigcirc\phi \in \Phi\} \cup (\Phi \cap \mathbf{C})$.

A branch $b \in B$ is said to be maximal *if it is not a proper prefix of another branch in B.*

Rules 4a and 4b are standard, replacing α and β-formulas with one or two formulas according to the matching pattern of rules in Fig. 3, except for Rule R7 that uses the so-called context Γ^*, which is defined in the following. The next operator used in Rule 4d is different from the corresponding one of PLTL since it also preserves the constraint formulas. This is needed for guaranteeing correctness since, as already mentioned, in *tccp* computations the store is monotonic, thus ($c \dot{\rightarrow} \bigcirc c$ and) constraint information has to be permanent.Finally, Rule 4c is specific for the $\dot{\exists}$ case: $\dot{\exists}_x$ is removed after renaming x with a fresh variable.[8]

Definition 7. *A node in the tableau is* inconsistent *if it contains*

– *a couple of formulas $\phi, \dot{\neg}\phi$, or*
– *the formula $\dot{false}$, or*
– *a constraint formula $\dot{\neg} c'$ such that the merge c of all the (positive) constraint formulas $c_1, \ldots, c_n$ in the node ($c \coloneqq c_1 \otimes \cdots \otimes c_n$) is such that $c \vdash c'$.*

When a branch contains an inconsistent node, it is said to be closed*, otherwise it is said to be* open*.*

The last condition for inconsistence of a node is particular to the *ccp* context.

Similarly to the PLTL case, there exists only a finite number of different labels in a tableau. Thus, if there exists an infinite branch $b = n_0, n_1, \ldots n_k \ldots$, it necessarily contains a cycle (i.e., contains infinitely many repetition of nodes with the same label). These branches are called *cyclic branches* and can be finitely represented as $\mathsf{path}(b) = n_0, n_1, \ldots, n_j, (n_{j+1}, \ldots, n_k)^\omega$ when $L(n_k) = L(n_j)$ for $0 \leq j < k$. Every branch of a tableau is divided into stages, denoted by $\mathsf{stages}(b)$. A *stage* is a sequence of consecutive nodes between two consecutive applications of the next operator. We abuse of notation and denote by $L(s)$ the labeling of a stage s defined as $\bigcup_{n \in s} L(n)$. It can be noticed that if b contains a cyclic sequence of nodes, then $\mathsf{stages}(b)$ is a cyclic sequence of stages.

Definition 8 ([GHL+09]). *A stage s is* saturated *if no α-, β- or hiding rule can be applied to any of its nodes.*

An eventuality formula $\phi_1 \,\mathcal{U}\, \phi_2$ that belongs to the labeling of a stage s in a branch it is fulfilled *if there exists a subsequent stage s' such that $\phi_2 \in L(n')$.*

A sequence of stages S is fulfilling *if all the eventuality formulas are fulfilled in S and a branch b is* fulfilling *if all $\mathsf{stages}(b)$ are fulfilling.*

An open branch is expanded *if it is fulfilling and all its stages are saturated.*

A tableau is called expanded *if every maximal branch is expanded or closed.*

An expanded tableau is closed *if every branch ends in an inconsistent node, otherwise it is* open*.*

[8] Note that the csLTL existential quantification does not correspond to the one of first-order logic. It is introduced to model information about local variables, and $\dot{\exists}_x \phi$ can be seen just as ϕ where the information about x is local.

These notions are needed to formalize the tableau construction since only branches that are non-expanded and open are selected to be further developed.

Definition 9 (systematic tableau algorithm). *Given a finite set of formulas Φ, the* systematic tableau $\mathcal{T}_\Phi$ *is built by repeatedly selecting an unmarked leaf node l and applying, in order, one of the points shown below.*

1. *Select an eventuality in l (if there is at least one) and distinguish it.*
2. *If l is an inconsistent node, then mark it as closed ($\times$).*
3. *If $L(l)$ is a set of constraint formulas, mark l as open ($\odot$).*
4. *Choose $\phi \in L(l)$ such that ϕ is not a* next *formula and it is not the distinguished eventuality. Then,*
   - *if ϕ is an $\exists$-formula ($\phi = \dot{\exists}_x\, \phi'$), then create a new node l' as a child of l and label it as $L(l') = (L(l) \setminus \{\phi\}) \cup \{\phi'\}$, where $\phi' := \phi[y/x]$ with y fresh variable;*
   - *if ϕ is an α-formula, create a new node l' as a child of l and label it as $L(l') = (L(l) \setminus \{\phi\}) \cup A(\phi)$ by using the corresponding rule in Fig. 3;*
   - *if ϕ is a β-formula, create two new nodes l' and l'' as children of l and label them as $L(l') = (L(l)\setminus\{\phi\})\cup B_1(\phi)$ and $L(l'') = (L(l)\setminus\{\phi\})\cup B_2(\phi)$ by using the corresponding rule in Fig. 3. For Rule R7, when ϕ is an eventuality, we choose $\Gamma^* := true$.*
5. *When all the non distinguish formulas have been selected, apply Rule R7 with $\Gamma^* := \dot{\bigvee}_{\gamma\in\Gamma} \dot{\neg}\gamma$ to the distinguish eventuality ϕ: create two new nodes l' and l'' as children of l and label them as $L(l') = (L(l) \setminus \{\phi\}) \cup B_1(\phi)$ and $L(l'') = (L(l) \setminus \{\phi\}) \cup B_2(\phi)$. Then, distinguish the* next *formula in $B_2(\beta)$;.*
6. *If $L(l)$ is a set of elementary formulas, then*
7. *if $L(l) = L(l')$ for l' ancestor of l (i.e., we detect a cycle), take the oldest ancestor of l that is labeled as $L(l)$ (denote it by l'') and check if all the eventualities in the path between l'' and l have been distinguished in such path. In this case mark l as open ($\odot$). Otherwise, apply the* next *operator: create a new node l' as child of l and label it as $L(l') = \mathsf{next}(L(l))$. Then, distinguish a new eventuality in $L(l')$ following a fair strategy.*
8. *If no cycle has been detected, apply the* next *operator: create a new node l' as child of l and label it as $L(l') = \mathsf{next}(L(l))$. If ϕ is the distinguished formula in l and $\mathsf{next}(\phi) = \phi'$, phi$'$ becomes the distinguished eventuality in l'. Otherwise, distinguish a new eventuality in $L(l')$ following a fair strategy.*

The construction terminates when every branch is marked.

Lemma 2. *Given a finite set $\Phi \subseteq$ csLTL, the algorithm of Definition 9 using a fair strategy terminates and builds an expanded tableau for $\mathcal{T}_\Phi$.*

The proposed algorithm is sound and complete for proving the satisfiability/unsatisfiability of csLTL formulas.

Theorem 4 (soundness and completeness). *$\Phi \subseteq$ csLTL is unsatisfiable if and only if there exists a closed systematic tableau for Φ.*

Proposition 1. *Let $\mathcal{T}_\Phi$ be an open systematic tableau for $\Phi = \{\psi, \dot{\neg}\phi\}$, b be an open branch in $\mathcal{T}_\Phi$, φ_i be the conjunction of the constraint formulas occurring in the i-th stage of b and φ be $\varphi_1 \dot{\wedge} \bigcirc\varphi_2 \ldots \dot{\wedge} \bigcirc^n \varphi_n$. Then $\varphi \dot{\rightarrow} \psi$ and $\varphi \dot{\nrightarrow} \phi$.*

The construction consists in selecting at each step a non-expanded branch that can be extended by using α or β rules or $\dot{\exists}$ elimination. When none of these can be applied, the next operator is used to pass to the next stage. When dealing with eventualities, to determine the context Γ^* in Rule R7, it is necessary to *distinguish* the eventuality that is being unfolded in the path. Given a node n and $\phi \in L(n)$, $\Gamma := L(n) \setminus \{\phi\}$. Then, when Rule R7 is applied to a *distinguished* eventuality, we set $\Gamma^* := \dot{\bigvee}_{\gamma \in \Gamma} \dot{\neg}\gamma$; otherwise $\Gamma^* := \mathit{true}$. If a node does not contain any distinguished eventuality, then the algorithm distinguishes one of them and rule R7 is applied to it. Each node of the tableau has at most one distinguished eventuality.

The algorithm marks nodes when they cannot be further processed. In particular, a node is marked as *closed* when it is inconsistent and is marked as *open* when it contains just constraint formulas or when it is the last node of an expanded branch (all the eventualities in the branch have been distinguished).

By construction, each stage in the systematic tableau $\mathcal{T}_\Phi$ for Φ is saturated. In order to ensure termination of the algorithm it is necessary to use a *fair* strategy to distinguish eventualities, in the sense that every eventuality in an open branch must be distinguished at some point. This assumption and the fact that, given a finite set of initial formulas, there exists only a finite set of possible labels in a systematic tableau, imply termination.

It is worth noticing that, by the application of the rules in Fig. 3, when both ϕ and $\neg\phi$ belong to the labeling of a stage in a branch b, then any branch prefixed by b is closed. Moreover, by construction, non-fulfilled undistinguished eventualities in a branch are maintained until they are fulfilled or they become distinguished. Thanks to Theorem 4, to check the validity of a formula (involved in Definition 5) of the form $\psi \dot{\rightarrow} \phi$, with $\phi = \hat{\mathcal{S}}(p(\vec{x}))$ and $\psi = \hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{S}}}(p(\vec{x}))$, we just have to build the tableau for its negation $\mathcal{T}_{\dot{\neg}(\psi \dot{\rightarrow} \phi)}$ and check if it is closed or not. If it is, we have that D is abstractly correct. Otherwise, by Proposition 1, we have thatfrom $\mathcal{T}_{\dot{\neg}(\psi \dot{\rightarrow} \phi)}$ we can extractan explicit testimony φ of the abstract incorrectness of D. , since $\varphi \dot{\rightarrow} \hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{S}}}(p(\vec{x}))$ and $\varphi \dot{\nrightarrow} \hat{\mathcal{S}}(p(\vec{x}))$.

The construction of $\psi = \hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{S}}}(p(\vec{x}))$ is linear in the size of D. The systematic tableau construction of $\dot{\neg}(\psi \dot{\rightarrow} \phi)$ (from what said in [GHL+09]) has worst case $O(2^{O(2^{|\dot{\neg}(\psi \dot{\rightarrow} \phi)|})})$. However, we believe that such bound for the worst-case asymptotic behavior is quite meaningless in this context, since it is not very realistic to think that the formulas of the specification should grow much (big formulas are difficult to comprehend and in real situations people would hardly try even to imagine them). Consequently, we would not have big implications $\psi \dot{\rightarrow} \phi$, since ψ is bounded by ϕ.Moreover, note that tableau explosion is due to nesting of eventualities and in practice few eventualities are used in specifications. Therefore, in real situations, we do not expect that (extremely) big tableaux will be built.

$$\begin{tikzcd}
\underline{\dot{\exists}_x \phi \mathbin{\dot\wedge} \Box(\dot\neg y = 1)} \arrow[r, "\alpha"] & \underline{\dot{\exists}_x \phi}, \Box(\dot\neg y = 1) \arrow[r, "\exists"] & \underline{\phi}, \Box(\dot\neg y = 1) \arrow[dll] \arrow[dr, "\beta"'] & \\
\underline{y = 1 \mathbin{\dot\wedge} \bigcirc x = 5 \mathbin{\dot\wedge} \bigcirc(\Diamond y = 1)}, \Box(\dot\neg y = 1) \arrow[d, "\alpha"] & & & \underline{\dot\neg y = 1 \mathbin{\dot\wedge} \bigcirc y = 1}, \Box(\dot\neg y = 1) \arrow[d, "\alpha"] \\
y = 1, \bigcirc x = 5, \bigcirc(\Diamond y = 1), \underline{\Box(\dot\neg y = 1)} \arrow[d, "\alpha"] & & & \dot\neg y = 1, \bigcirc y = 1, \underline{\Box(\dot\neg y = 1)} \arrow[d, "\alpha"] \\
y = 1, \bigcirc x = 5, \bigcirc(\Diamond y = 1), \dot\neg y = 1, \bigcirc\Box(\dot\neg y = 1) & & & \dot\neg y = 1, \bigcirc y = 1, \bigcirc(\Box(\dot\neg y = 1)) \arrow[d, "X"] \\
\times & & & y = 1, \underline{\Box(\dot\neg y = 1)} \arrow[d, "\alpha"] \\
& & & y = 1, \dot\neg y = 1, \bigcirc(\Box(\dot\neg y = 1)) \\
& & & \times
\end{tikzcd}$$

Fig. 4. Tableau for $\dot{\exists}_x \phi \mathbin{\dot\rightarrow} \Diamond y = 1$ of Example 9.

Furthermore, since we can work with incomplete programs, we can validate pieces of code in isolation (even before having wrote the whole program), thus the computational cost can be diluted during development.

Example 9. Let us assume that we are trying to check whether process

$$R := p(y) :- \exists x\,(\mathsf{now}\ y = 1\ \mathsf{then}\ \mathsf{tell}(x = 5) \parallel p(y)\ \mathsf{else}\ \mathsf{tell}(y = 1))$$

satisfies $\hat{\mathcal{S}}(p(\vec{x})) := \Diamond(y = 1)$. Since $\hat{\mathcal{D}}[\![\{R\}]\!]_{\hat{\mathcal{S}}} = \dot{\exists}_x \phi$, where

$$\phi = (y = 1 \mathbin{\dot\wedge} \bigcirc x = 5 \mathbin{\dot\wedge} \bigcirc(\Diamond y = 1)) \mathbin{\dot\vee} (\dot\neg y = 1 \mathbin{\dot\wedge} \bigcirc y = 1)$$

Thus, we have to check if $\dot{\exists}_x \phi \mathbin{\dot\rightarrow} \Diamond(y = 1)$. Fig. 4 shows the systematic tableau built for the negation of the formula, i.e., $\dot{\exists}_x \phi \mathbin{\dot\wedge} \Box \dot\neg(y = 1)$.

Arrows labeled with α and β correspond to the application of α and β rules, respectively; arrows labeled with X represent the application of the next operator. Finally, arrows labeled with $\exists$ correspond to the elimination of the existential quantification for the formula $\dot{\exists}_x \phi$.

Since both branches are closed, the tableaux is closed, which means that $\dot{\exists}_x \phi \mathbin{\dot\wedge} \Box \dot\neg(y = 1)$ is not satisfied and its negation $\dot{\exists}_x \phi \mathbin{\dot\rightarrow} \Diamond(y = 1)$ is valid, meaning that R is abstractly correct w.r.t. $\hat{\mathcal{S}}$.

5 Related Work

A Constraint Linear Temporal Logic is defined in [Val05] for the verification of a different timed concurrent language, called *ntcc*, which shares with *tccp* the concurrent constraint nature and the non-monotonic behavior. A fragment of the proposed logic, the restricted negation fragment where negation is only allowed for state formulas, is shown to be decidable. However, no efficient decision procedure is given (apart from the proof itself). Moreover, the verification results are given for the locally-independent fragment of *ntcc*, which avoids the non-monotonicity of the original language. In contrast, in this work, we address the problem of checking temporal properties for the full *tccp* language.

Some model-checking techniques have been defined for *tccp* in the past, seex [FV06,AFV05,AGPV05,FPV01]. It is worth noting that the notions of correctness and completeness in these works are defined in terms of $\mathcal{F}[\![D]\!]$, i.e., in terms of the concrete semantics, and therefore their check requires a (potentially infinite) fixpoint computation. In contrast, the notions of abstractly incorrect declarations and abstract uncovered elements are defined in terms of *just one* application of $\hat{\mathcal{D}}[\![D]\!]$ to $\hat{\mathcal{S}}$. Moreover, since $\hat{\mathcal{D}}[\![D]\!]$ is defined compositionally, all the checks are defined on each process declaration in isolation. Hence, our proposal can be used with partial sets of declarations. It can also be used with partial specifications (an undefined specification of a process is implicitly false). Obviously, one cannot detect errors in declarations involving processes which have not been specified, but for the declarations that involve processes that have a specification, the check can be made, even if the whole set of declarations has not been written yet.

This is particularly useful for applications, since the diagnosis could be used from the beginning of the development phase. Moreover, it could be performed incrementally, thus the overall computational cost can be parceled over time. On the contrary, model checking can be applied only with a fully specified system and its inherent complexity resides in its necessity to compute somehow a semantics fixpoint.

With our proposal, we can easily specify a possible intervention coming from a surrounding environment simply by adding a suitable formula. With model checking, this needs to be done by simulating such environment in software with an additional set of declarations. When a property is falsified, model checking provides a counterexample in terms of an erroneous execution trace, leaving to the user the problem of locating the source of the bug. On the contrary, we identify the faulty process declaration.

In [FOPV07], a first approach to the declarative debugging of a *ccp* language is presented. However, it does not cover the particular extra difficulty of the non-monotonicity behavior, common to all timed concurrent constraint languages. This makes our approach significantly different. Moreover, although they propose the use of LTL for the specification of properties, their formulation, based on the depth-k concretization function, complicates the task of having an efficient implementation.

Finally, this proposal clearly relates to the abstract diagnosis framework for *tccp* defined for Galois Insertions [CTV11]. That work can compete with the precision of model checking, but its main drawback is the fact that the abstract domain used theredid not allow to specify temporal properties in a compact way. In fact, specifications consisted of sets of *abstract conditional traces*. Thus, specifications were big and unnatural to be written. The use of temporal logic in this proposal certainly overcomes this problem.

6 Conclusion and Future Work

In this paper, we have defined an abstract semantics for *tccp* based on the domain of a linear temporal logic with constraints. The semantics is correct w.r.t. the behavior of the language.

By using this abstract semantics, we have defined a method to validate csLTL formulas for *tccp* sets of declarations. Since the abstract semantics cannot be defined by means of a Galois Connection, we cannot use the abstract diagnosis framework for *tccp* defined in [CTV11], thus we devised (from scratch) a weak version of the abstract diagnosis framework based only on a concretization function γ. It works by applying $\hat{\mathcal{D}}[\![D]\!]$ to the abstract specification and then by checking the validity of the resulting implications (whether that computation implies the abstract specification). The computational cost depends essentially on the cost of that check of the implication.

We have also presented an automatic decision procedure for the csLTL logic, thus we can effectively check the validity of that implication. We are currently finishing to implement a proof of concept tool, which is available online at URL `http://safe-tools.dsic.upv.es/tadi/`, that realizes the proposed instance. Then we would be able to compare with other tools and assess the "real life" goodness of our proposal.

In the future, we also plan to explore other instances of the method based on logics for which decision procedures or (semi)automatic tools exists. This proposal can also be immediately adapted to other concurrent (non-monotonic) languages (like *tcc* and *ntcc*) once a suitable fully abstract semantics has been developed.

References

AFV05. M. Alpuente, M. Falaschi, and A. Villanueva. A Symbolic Model Checker for tccp Programs. In *First International Workshop on Rapid Integration of Software Engineering Techniques (RISE 2004), Revised Selected Papers*, volume 3475 of *Lecture Notes in Computer Science*, pages 45–56. Springer-Verlag, 2005.

AGPV05. M. Alpuente, M.M. Gallardo, E. Pimentel, and A. Villanueva. A Semantic Framework for the Abstract Model Checking of tccp Programs. *Theoretical Computer Science*, 346(1):58–95, 2005.

AGPV06. M. Alpuente, M.M. Gallardo, E. Pimentel, and A. Villanueva. Verifying Real-Time Properties of tccp Programs. *Journal of Universal Computer Science*, 12(11):1551–1573, 2006.

BCC$^+$03. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded Model Checking. *Advances in Computers*, 58:117–148, 2003.

BCM$^+$92. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. Comput.*, 98(2):142–170, 1992.

CE81. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

CGL92. E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. In *Conference Record of the 19th ACM Symp. on Principles of Programming Languages (POPL'92)*, pages 343–354, New York, NY, USA, 1992. ACM Press.

CTV11. M. Comini, L. Titolo, and A. Villanueva. Abstract Diagnosis for Timed Concurrent Constraint programs. *Theory and Practice of Logic Programming*, 11(4-5):487–502, 2011.

CTV13a. M. Comini, L. Titolo, and A. Villanueva. A Condensed Goal-Independent Bottom-Up Fixpoint Modeling the Behavior of `tccp`. Technical report, DSIC, Universitat Politècnica de València. Available at http://riunet.upv.es/handle/10251/8351, 2013.

CTV13b. M. Comini, L. Titolo, and A. Villanueva. Towards an Effective Decision Procedure for LTL formulas with Constraints. In *23rd Workshop on Logic-based methods in Programming Environments (WLPE 2013)*, volume abs/1308.2055, 2013.

Dam96. D. R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD thesis, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands, 1996.

dBGM00. F. S. de Boer, M. Gabbrielli, and M. C. Meo. A Timed Concurrent Constraint Language. *Information and Computation*, 161(1):45–83, 2000.

dBGM01. F. S. de Boer, M. Gabbrielli, and M. C. Meo. A Temporal Logic for Reasoning about Timed Concurrent Constraint Programs. In *TIME '01: Proceedings of the Eighth International Symposium on Temporal Representation and Reasoning (TIME'01)*, page 227, Washington, DC, USA, 2001. IEEE Computer Society.

dBGM02. F. S. de Boer, M. Gabbrielli, and M. C. Meo. Proving correctness of Timed Concurrent Constraint Programs. *CoRR*, cs.LO/0208042, 2002.

FOPV07. M. Falaschi, C. Olarte, C. Palamidessi, and F. D. Valencia. Declarative Diagnosis of Temporal Concurrent Constraint Programs. In V. Dahl and I. Niemelä, editors, *Logic Programming, 23rd International Conference, ICLP 2007, Proceedings*, volume 4670 of *Lecture Notes in Computer Science*, pages 271–285. Springer-Verlag, 2007.

FPV01. M. Falaschi, A. Policriti, and A. Villanueva. Modeling concurrent systems specified in a temporal concurrent constraint language-I. *Electronic Notes in Theoretical Computer Science*, 48:197–210, 2001.

FV06. M. Falaschi and A. Villanueva. Automatic verification of timed concurrent constraint programs. *Theory and Practice of Logic Programming*, 6(3):265–300, 2006.

GHL$^+$07. J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro, and F. Orejas. A cut-free and invariant-free sequent calculus for PLTL. In J. Duparc and T. A. Henzinger, editors, *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 481–495. Springer, 2007.

GHL$^+$09. J. Gaintzarain, M. Hermo, P. Lucio, M. Navarro, and F. Orejas. Dual Systems of Tableaux and Sequents for PLTL. *The Journal of Logic and Algebraic Programming*, 78(8):701–722, 2009.

GHLN08. J. Gaintzarain, M. Hermo, P. Lucio, and M. Navarro. Systematic semantic tableaux for PLTL. *Electronic Notes in Theoretical Computer Science*, 206:59–73, 2008.

HNSY94. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111(2):193–244, 1994.

Hol96. G. J. Holzmann. On-The-Fly model checking. *ACM Comput. Surv.*, 28(4es):120, 1996.

MP92. Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems - specification.* Springer, 1992.

PV01. C. Palamidessi and F. D. Valencia. A Temporal Concurrent Constraint Programming Calculus. In *7th International Conference on Principles and Practice of Constraint Programming (CP'01)*, volume 2239 of *Lecture Notes in Computer Science*, pages 302–316. Springer, 2001.

QS82. J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In M. Dezani-Ciancaglini and U. Montanari, editors, *Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

Sar93. V. A. Saraswat. *Concurrent Constraint Programming.* The MIT Press, Cambridge, Mass., 1993.

SRP91. V. A. Saraswat, M. Rinard, and P. Panangaden. The Semantic Foundations of Concurrent Constraint Programming. In *Proceedings of the 18th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 333–352, New York, NY, USA, 1991. ACM.

Val05. F. D. Valencia. Decidability of infinite-state timed CCP processes and first-order LTL. *Theoretical Computer Science*, 330(3):577–607, 2005.

A Proofs and results of Sections 3 and 4

In this section we present the proofs of the results presented in Sections 3 and 4 and some auxiliary results.

Lemma 1. *The function $\gamma^{\mathbb{F}}$ is monotonic, injective and $\bigsqcap$-distributive.*

Proof. **$\gamma^{\mathbb{F}}$ is monotonic.** Let $\phi_1, \phi_2 \in \mathbb{F}$ such that $\phi_1 \dot{\rightarrow} \phi_2$. By Definition 2, for all $r \in \mathbb{M}$, if $r \vDash \phi_1$ then $r \vDash \phi_2$. Thus, $\bigsqcup\{r \mid r \vDash \phi_1\} \sqsubseteq \bigsqcup\{r \mid r \vDash \phi_2\}$ and, by Equation (3.1), $\gamma^{\mathbb{F}}(\phi_1) \sqsubseteq \gamma^{\mathbb{F}}(\phi_2)$.

$\gamma^{\mathbb{F}}$ is injective. Let $\phi_1, \phi_2 \in \mathbb{F}$ such that $\gamma^{\mathbb{F}}(\phi_1) = \gamma^{\mathbb{F}}(\phi_2)$. By Equation (3.1) and Definition 2, this means that ϕ_1 and ϕ_2 have the same models, thus, $\phi_1 \dot{\leftrightarrow} \phi_2$.

$\gamma^{\mathbb{F}}$ is $\bigsqcap$-distributive. Consider $\phi_1, \phi_2 \in \mathbb{F}$, we show that $\gamma^{\mathbb{F}}(\phi_1 \dot{\wedge} \phi_2) = \gamma^{\mathbb{F}}(\phi_1) \sqcap \gamma^{\mathbb{F}}(\phi_1)$.

$$
\begin{aligned}
\gamma^{\mathbb{F}}(\phi_1 \dot{\wedge} \phi_2) =& \bigsqcup\{r \in \mathbb{M} \mid r \vDash \phi_1 \dot{\wedge} \phi_2\}\\
& \quad [\text{by Equation (3.2e)}]\\
=& \bigsqcup\{r \in \mathbb{M} \mid r \vDash \phi_1 \text{ and } r \vDash \phi_2\}\\
& \quad [\text{by definition of } \textstyle\bigsqcap]\\
=& \bigsqcup\{r \in \mathbb{M} \mid r \vDash \phi_1\} \sqcap \bigsqcup\{r \in \mathbb{M} \mid r \vDash \phi_2\}\\
& \quad [\text{by Definition 2}]\\
=& \gamma^{\mathbb{F}}(\phi_1) \sqcap \gamma^{\mathbb{F}}(\phi_1)
\end{aligned}
$$

Lemma 3 (Correctness of $\dot{\bigwedge}$). *Given $\phi_1, \phi_2 \in \mathbb{F}$, $\gamma^{\mathbb{F}}(\phi_1 \dot{\wedge} \phi_2) \sqsupseteq \bigsqcup\{r_1 \mathbin{\|} r_2 \mid r_1 \in \gamma^{\mathbb{F}}(\phi_1),\ r_2 \in \gamma^{\mathbb{F}}(\phi_2),\ r_1 \mathbin{\|} r_2$ is defined$\}$.*

Proof. Consider $r_1 \in \gamma^{\mathbb{F}}(\phi_1)$ and $r_2 \in \gamma^{\mathbb{F}}(\phi_2)$ such that $r_1 \mathbin{\|} r_2$ is defined. We show that $r_1 \mathbin{\|} r_2 \in \gamma^{\mathbb{F}}(\phi_1 \dot{\wedge} \phi_2)$ (i.e., $r_1 \mathbin{\|} r_2 \vDash \phi_1 \dot{\wedge} \phi_2$). Since $r_1 \mathbin{\|} r_2$ is defined, the conditions and stores in r_2 cannot be in contradiction with those in r_1, thus neither with ϕ_1, which means that $(r_1 \mathbin{\|} r_2) \vDash \phi_1$. Following a similar reasoning, we have that $(r_1 \mathbin{\|} r_2) \vDash \phi_2$ and finally, from Equation (3.2e) we can conclude that $(r_1 \mathbin{\|} r_2) \vDash \phi_1 \dot{\wedge} \phi_2$. □

Lemma 4 (Correctness of $\dot{\exists}_x$).
Given $\phi \in \mathbb{F}$, $\gamma^{\mathbb{F}}(\dot{\exists}_x \phi) \sqsupseteq \bigsqcup\{\bar{\exists}_x r \mid r \in \gamma^{\mathbb{F}}(\phi),\ r$ is x-self-sufficient$\}$.

Proof. Let $r \in \gamma^{\mathbb{F}}(\phi)$ be x-self-sufficient. By Equation (3.1) $r \vDash \phi$, and by Equation (3.2h) it follows directly that $\bar{\exists}_x r \vDash \dot{\exists}_x \phi$. By Equation (3.1) we can conclude that $\bar{\exists}_x r \in \gamma^{\mathbb{F}}(\dot{\exists}_x \phi)$. □

Theorem 1. *Let $A \in \mathbb{A}^{\Pi}_{\mathbf{C}}$, $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$ and $\hat{\mathcal{I}} \in \mathbb{I}_{\mathbb{F}}$. Then, $\mathcal{A}[\![A]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})} \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![A]\!]_{\hat{\mathcal{I}}})$ and $\mathcal{D}[\![D]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})} \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{I}}})$.*

Proof. Let $A \in \mathbb{A}^{\Pi}_{\mathbf{C}}$ and $\hat{\mathcal{I}} \in \mathbb{I}_{\mathbb{F}}$, we show that $\mathcal{A}[\![A]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})} \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![A]\!]_{\hat{\mathcal{I}}})$ by structural induction on A.

Case $A = \mathsf{skip}$**.**

$$\begin{aligned}
\mathcal{A}[\![\mathsf{skip}]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})} &= \{(\mathit{true},\varnothing) \rightarrowtail \mathit{true} \cdots (\mathit{true},\varnothing) \rightarrowtail \mathit{true} \cdots\} \\
&\qquad [\,\text{by Definition 2 since } \boxtimes \vDash \mathit{true}\,] \\
&\sqsubseteq \{r \mid r \vDash \mathit{true}\} \\
&\qquad [\,\text{by Equation (3.1)}\,] \\
&= \gamma^{\mathbb{F}}(\mathit{true}) \\
&\qquad [\,\text{by Equation (3.4a)}\,] \\
&= \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![\mathsf{skip}]\!]_{\hat{\mathcal{I}}})
\end{aligned}$$

Case $A = \mathsf{tell}(c)$**.**

$$\begin{aligned}
\mathcal{A}[\![\mathsf{tell}(c)]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})} &= \{(\mathit{true},\varnothing) \rightarrowtail c \cdot (c,\varnothing) \rightarrowtail c \cdots (c,\varnothing) \rightarrowtail c \cdots\} \\
&\qquad [\,\text{by Definition 2}\,] \\
&\sqsubseteq \{r \mid r \vDash \bigcirc c\} \\
&\qquad [\,\text{by Equation (3.1)}\,] \\
&= \gamma^{\mathbb{F}}(\bigcirc c) \\
&\qquad [\,\text{by Equation (3.4b)}\,] \\
&= \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![\mathsf{tell}(c)]\!]_{\hat{\mathcal{I}}})
\end{aligned}$$

Case $A = \sum_{i=1}^{n} \mathsf{ask}(c_i) \to A_i$**.** Let $r \in \mathcal{A}[\![A]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$, we have to prove two cases.

1. Let $r := \underbrace{\mathit{stutt}(\{c_1,\dots,c_n\}) \dots \mathit{stutt}(\{c_1,\dots,c_n\})}_{k \text{ times}} \cdot (c_i,\varnothing) \rightarrowtail c_i \cdot (r_i{\downarrow_{c_i}})$ with $1 \leq i \leq n$ and $r_i \in \mathcal{A}[\![A_i]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$. From (3.2b) and (3.2d), it follows that $\mathit{stutt}(\{c_1,\dots,c_n\}) \vDash \dot{\neg} c_i$ for all $1 \leq i \leq n$. Thus, by (3.2e), for all $1 \leq j \leq k$ $r^j \vDash \dot{\bigwedge}_{i=1}^{n} \dot{\neg} c_i$. From (3.2b), it follows that $(c_i,\varnothing) \rightarrowtail c_i \vDash c_i$, and, by inductive hypothesis, $r_i \vDash \hat{\mathcal{A}}[\![A_i]\!]_{\hat{\mathcal{I}}}$. Therefore the sub-trace $(c_i,\varnothing) \rightarrowtail c_i \cdot (r_i{\downarrow_{c_i}})$ models the formula $c_i \mathbin{\dot{\wedge}} \bigcirc \hat{\mathcal{A}}[\![A_i]\!]_{\hat{\mathcal{I}}}$ and as a consequence models also $\dot{\bigvee}_{i=1}^{n} (c_i \mathbin{\dot{\wedge}} \bigcirc \hat{\mathcal{A}}[\![A_i]\!]_{\hat{\mathcal{I}}})$. Since this sub-trace is precede in r by the suffix $\mathit{stutt}(\{c_1,\dots,c_n\}) \dots \mathit{stutt}(\{c_1,\dots,c_n\})$, from (3.2g), it follows that $r \vDash (\dot{\bigwedge}_{i=1}^{n} \dot{\neg} c_i) \,\mathcal{U}\, \dot{\bigvee}_{i=1}^{n} (c_i \mathbin{\dot{\wedge}} \bigcirc \hat{\mathcal{A}}[\![A_i]\!]_{\hat{\mathcal{I}}})$ and, from (3.4c), we can conclude that $r \vDash \hat{\mathcal{A}}[\![A]\!]_{\hat{\mathcal{I}}}$.
2. Let $r := \mathit{stutt}(\{c_1,\dots,c_n\}) \cdots \mathit{stutt}(\{c_1,\dots,c_n\}) \cdots$. From (3.2c) and (3.2d) it follows that $\mathit{stutt}(\{c_1,\dots,c_n\}) \vDash \dot{\neg} c_i$ for all $1 \leq i \leq n$. Thus, by (3.2e), $\mathit{stutt}(\{c_1,\dots,c_n\}) \vDash \dot{\bigwedge}_{i=1}^{n} \dot{\neg} c_i$. Since r is an infinite replication of $\mathit{stutt}(\{c_1,\dots,c_n\})$, by definition of $\square$, $r \vDash \square \dot{\bigwedge}_{i=1}^{n} \dot{\neg} c_i$ and, by (3.4c) we can conclude that $r \vDash \hat{\mathcal{A}}[\![A]\!]_{\hat{\mathcal{I}}}$.

Case $A = \mathsf{now}\ c\ \mathsf{then}\ A_1\ \mathsf{else}\ A_2$**.** Let $r \in \mathcal{A}[\![A]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$. We show that $r \vDash (c \mathbin{\dot{\wedge}} \hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}}) \mathbin{\dot{\vee}} (\dot{\neg} c \mathbin{\dot{\wedge}} \hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}})$. We have to prove seven cases:

1. Let $r \coloneqq (c, \varnothing) \rightarrowtail c \cdots (c, \varnothing) \rightarrowtail c \cdots$ such that $(\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots \in \mathcal{A}[\![A_1]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$. From (3.2b) it follows that $r \vDash c$. By inductive hypothesis, $(\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots \in \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}})$. Moreover, true is the stronger formula that $(\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots$ can model. Thus, it follows that $\mathit{true} \dot{\rightarrow} \hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}}$. Since $\forall \phi \in \mathsf{csLTL}.\ \phi \dot{\rightarrow} \mathit{true} \dot{\wedge} \phi$, it holds that $r \vDash c \dot{\wedge} \hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}}$.
2. Let $r \coloneqq (\eta^+ \otimes c, \eta^-) \rightarrowtail d \otimes c \cdot (r' {\downarrow_c})$ such that $(\eta^+, \eta^-) \rightarrowtail d \cdot r' \in \mathcal{A}[\![A_1]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$, $d \otimes c \neq \mathit{false}$, $\forall c^- \in \eta^-.\ \eta^+ \otimes c \nvdash c^-$ and r' is c-compatible. By (3.2b), it follows that $r \vDash c$. By inductive hypothesis, we know that $(\eta^+, \eta^-) \rightarrowtail d \cdot r' \in \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}})$, and by (3.1), $(\eta^+, \eta^-) \rightarrowtail d \cdot r' \vDash \hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}}$. By hypothesis, $(\eta^+, \eta^-) \rightarrowtail d \cdot r'$ is compatible with c, thus $\hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}}$ cannot contain $\dot{\neg} c$. Furthermore, it can be noticed that r adds to $(\eta^+, \eta^-) \rightarrowtail d \cdot r'$ only the constraint c in the positive conditions and in the stores, thus, it follows that $r \vDash \hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}}$. By (3.2e) we conclude that $r \vDash c \dot{\wedge} \hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}}$.
3. Let $r \coloneqq (\eta^+ \otimes c, \eta^-) \rightarrowtail \mathit{false} \cdot (\mathit{false}, \varnothing) \rightarrowtail \mathit{false} \cdots (\mathit{false}, \varnothing) \rightarrowtail \mathit{false} \cdots$ such that $(\eta^+, \eta^-) \rightarrowtail d \cdot r' \in \mathcal{A}[\![A_1]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$, $d \otimes c = \mathit{false}$, $\forall c^- \in \eta^-.\ \eta^+ \otimes c \nvdash c^-$ and r' is c-compatible. By (3.2b), it follows that $r \vDash c$. By inductive hypothesis, $(\eta^+, \eta^-) \rightarrowtail d \cdot r' \in \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}})$ and, by (3.1), $(\eta^+, \eta^-) \rightarrowtail d \cdot r' \vDash \hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}}$. Reasoning similarly to Point 2 above, it can be noticed that $r \vDash \hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}}$. Thus, by (3.2e) we conclude that $r \vDash c \dot{\wedge} \hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}}$.
4. Let $r \coloneqq (c, \eta^-) \rightarrowtail c \cdot (r' {\downarrow_c})$ such that $\mathit{stutt}(\eta^-) \cdot r' \in \mathcal{A}[\![A_1]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$, $\forall c^- \in \eta^-.\ \eta^+ \otimes c \nvdash c^-$ and r' is c-compatible. It follows from (3.2b) that $r \vDash c$. By inductive hypothesis, $\mathit{stutt}(\eta^-) \cdot r' \in \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}})$, and, by (3.1), $\mathit{stutt}(\eta^-) \cdot r' \vDash \hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}}$. Reasoning as in Point 2 of this proof it follows that $r \vDash \hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}}$. Thus, $r \vDash c \dot{\wedge} \hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}}$.
5. Let $r \coloneqq (\mathit{true}, \{c\}) \rightarrowtail \mathit{true} \cdot (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots$ such that $(\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots \in \mathcal{A}[\![A_2]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$. By (3.2b) and (3.2d), $r \vDash \dot{\neg} c$. By inductive hypothesis, $(\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots \in \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}})$ and, reasoning as in Point 1 of this proof, it follows that $\mathit{true} \dot{\rightarrow} \hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}}$. Since $\forall \phi \in \mathsf{csLTL}.\ \phi \dot{\rightarrow} \mathit{true} \dot{\wedge} \phi$, it holds that $r \vDash \dot{\neg} c \dot{\wedge} \hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}}$.
6. Let $r \coloneqq (\eta^+, \eta^- \cup \{c\}) \rightarrowtail d \cdot r'$ such that $(\eta^+, \eta^-) \rightarrowtail d \cdot r' \in \mathcal{A}[\![A_2]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$ and $\eta^+ \nvdash c$. By (3.2b), $r \vDash \dot{\neg} c$. By inductive hypothesis, $(\eta^+, \eta^-) \rightarrowtail d \cdot r' \in \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}})$ and, by (3.1), $(\eta^+, \eta^-) \rightarrowtail d \cdot r' \vDash \hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}}$. It can be noticed that $\hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}}$ cannot imply the formula c, otherwise $(\eta^+, \eta^-) \rightarrowtail d \cdot r'$ would not be a model for $\hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}}$ since by hypothesis $\eta^+ \nvdash c$. Since r differs from $(\eta^+, \eta^-) \rightarrowtail d \cdot r'$ only in the presence of c in the first negative condition, it follows that $r \vDash \hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}}$. Thus, by (3.2e) we conclude that $r \vDash \dot{\neg} c \dot{\wedge} \hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}}$.
7. Let $r \coloneqq (\mathit{true}, \eta^- \cup \{c\}) \rightarrowtail \mathit{true} \cdot r'$ such that $\mathit{stutt}(\eta^-) \cdot r' \in \mathcal{A}[\![A_2]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$. By (3.2c), $r \vDash \dot{\neg} c$ and, by inductive hypothesis, $\mathit{stutt}(\eta^-) \cdot r' \vDash \hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}}$. Reasoning as in the previous Point 6 it can be noticed that $r \vDash \hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}}$ and, therefore, $r \vDash \dot{\neg} c \dot{\wedge} \hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}}$.

We have proven that for all $r \in \mathcal{A}[\![A]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$ either $r \vDash c \mathbin{\dot\wedge} \hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}}$ or $r \vDash \dot\neg c \mathbin{\dot\wedge} \hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}}$. Therefore, from (3.4d) we conclude that $r \vDash \hat{\mathcal{A}}[\![A]\!]_{\hat{\mathcal{I}}}$.

Case $A = A_1 \parallel A_2$**.** Let $r_1 \mathbin{\bar\parallel} r_2 \in \mathcal{A}[\![A_1 \parallel A_2]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$ such that $r_1 \in \mathcal{A}[\![A_1]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$ and $r_2 \in \mathcal{A}[\![A_2]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$. By inductive hypothesis, $r_1 \in \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}})$ and $r_2 \in \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![A_2]\!]_{\hat{\mathcal{I}}})$. It follows from Lemma 3 that $r_1 \mathbin{\bar\parallel} r_2 \in \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![A_1 \parallel A_2]\!]_{\hat{\mathcal{I}}})$.

Case $A = \exists x\, A_1$**.** Let $\bar\exists_x r_1 \in \mathcal{A}[\![\exists x\, A_1]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$ such that $r_1 \in \mathcal{A}[\![A_1]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$ and r_1 is x-closed. By inductive hypothesis, $r_1 \in \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![A_1]\!]_{\hat{\mathcal{I}}})$ and, by Lemma 4, it follows that $\bar\exists_x r_1 \in \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![\exists x\, A_1]\!]_{\hat{\mathcal{I}}})$.

Case $A = p(\vec{x})$**.** Let $r \coloneqq (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdot r' \in \mathcal{A}[\![p(\vec{x})]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}$ such that $r' \in \gamma^{\mathbb{F}}(\hat{\mathcal{I}}(p(\vec{x})))$. By (3.1), it follows that $r' \vDash \hat{\mathcal{I}}(p(\vec{x}))$. From (3.2f) we can conclude that $r \vDash \bigcirc \hat{\mathcal{I}}(p(\vec{x}))$ and, thus, $r \in \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![p(\vec{x})]\!]_{\hat{\mathcal{I}}})$.

Let $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$ and $\hat{\mathcal{I}} \in \mathbb{I}_{\mathbb{F}}$. We prove that $\mathcal{D}[\![D]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})} \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{I}}})$ by showing that for all $p(x) \coloneq A \in D$, $\mathcal{D}[\![D]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}(p(\vec{x})) \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{I}}}(p(\vec{x})))$.

$$\begin{aligned}
\mathcal{D}[\![D]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})}(p(\vec{x})) &= \bigsqcup_{p(x) :- A \in D} \mathcal{A}[\![A]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{I}})} \\
&\qquad [\,\text{by the soundness of } \hat{\mathcal{A}}\,] \\
&\sqsubseteq \bigsqcup_{p(x) :- A \in D} \gamma^{\mathbb{F}}(\hat{\mathcal{A}}[\![A]\!]_{\hat{\mathcal{I}}}) \\
&\qquad [\,\text{by the monotonicity of } \gamma^{\mathbb{F}} \text{ (Lemma 1)}\,] \\
&\sqsubseteq \gamma^{\mathbb{F}}(\dot\bigvee_{p(x) :- A \in D} \hat{\mathcal{A}}[\![A]\!]_{\hat{\mathcal{I}}}) \\
&= \gamma^{\mathbb{F}}(\hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{I}}})(p(\vec{x}))
\end{aligned}$$

Lemma 5. *For each $A \in \mathbb{A}^{\Pi}_{\mathbf{C}}$ and each $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$, $\hat{\mathcal{A}}[\![A]\!]$ and $\hat{\mathcal{D}}[\![D]\!]$ are monotonic.*

Proof. Consider $A \in \mathbb{A}^{\Pi}_{\mathbf{C}}$; we show that for each $\hat{\mathcal{I}}_1, \hat{\mathcal{I}}_2 \in \mathbb{I}_{\mathbb{F}}$ and for each $A \in \mathbb{A}^{\Pi}_{\mathbf{C}}$, $\hat{\mathcal{I}}_1 \mathbin{\dot\to} \hat{\mathcal{I}}_2 \Rightarrow \hat{\mathcal{A}}[\![A]\!]_{\hat{\mathcal{I}}_1} \mathbin{\dot\to} \hat{\mathcal{A}}[\![A]\!]_{\hat{\mathcal{I}}_2}$. Observe that the only case in which $\hat{\mathcal{A}}$ depends on the interpretation is the case of the process call. By definition of $\dot\to$, $\hat{\mathcal{I}}_1(p(\vec{x})) \mathbin{\dot\to} \hat{\mathcal{I}}_2(p(\vec{x}))$, thus:

$$\hat{\mathcal{A}}[\![p(\vec{x})]\!]_{\hat{\mathcal{I}}_1} = \bigcirc \hat{\mathcal{I}}_1(p(\vec{x})) \mathbin{\dot\to} \bigcirc \hat{\mathcal{I}}_2(p(\vec{x})) = \hat{\mathcal{A}}[\![p(\vec{x})]\!]_{\hat{\mathcal{I}}_2}$$

The monotonicity of $\hat{\mathcal{D}}[\![D]\!]$ follows directly from the monotonicity of $\hat{\mathcal{A}}[\![A]\!]$. □

Theorem 2. *Let $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$ and $\hat{\mathcal{S}} \in \mathbb{I}_{\mathbb{A}}$.*

1. *If there are no abstractly incorrect process declarations in D (i.e., $\hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{S}}} \mathbin{\dot\to} \hat{\mathcal{S}}$), then D is partially correct w.r.t. $\hat{\mathcal{S}}$.*
2. *Let D be partially correct w.r.t. $\hat{\mathcal{S}}$. If D has abstract uncovered elements then D is not complete.*

Proof. **Point 1** By hypothesis, $\hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{S}}} \dot{\to} \hat{\mathcal{S}}$. Since $\gamma^{\mathbb{F}}$ is monotonic (Lemma 1), $\gamma^{\mathbb{F}}(\hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{S}}}) \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{S}})$. By the soundness of $\hat{\mathcal{D}}$ (Theorem 1) and by transitivity it follows that $\mathcal{D}[\![D]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{S}})} \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{S}})$. It can be noticed that $\gamma^{\mathbb{F}}(\hat{\mathcal{S}})$ is a pre-fixpoint of $\mathcal{D}[\![D]\!]$, thus, from Knaster-Tarski's theorem it follows directly that $\mathit{lfp}\, \mathcal{D}[\![D]\!] \dot{\to} \gamma^{\mathbb{F}}(\hat{\mathcal{S}})$. The thesis follows directly from the definition of $\mathcal{F}[\![D]\!]$ ($\mathcal{F}[\![D]\!] = \mathit{lfp}\, \mathcal{D}[\![D]\!]$).

Point 2 By hypothesis, ϕ_t is such that $\phi_t \dot{\to} \hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{S}}}$ and $\phi_t \dot{\wedge} \hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{S}}} = \dot{\mathit{false}}$. Thus, it follows that $\gamma^{\mathbb{F}}(\hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{S}}}) \sqcap \gamma^{\mathbb{F}}(\hat{\mathcal{S}}) = \{\epsilon\}$. Since $\mathcal{D}[\![D]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{S}})} \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{D}}[\![D]\!]_{\hat{\mathcal{S}}})$, we have that $\mathcal{D}[\![D]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{S}})} \sqcap \gamma^{\mathbb{F}}(\hat{\mathcal{S}}) = \{\epsilon\}$. Suppose that $\gamma^{\mathbb{F}}(\mathcal{S}) \sqsubseteq \mathcal{F}[\![D]\!]$. Since by hypothesis $\mathcal{F}[\![D]\!] \sqsubseteq \gamma^{\mathbb{F}}(\mathcal{S})$, we have that $\mathcal{F}[\![D]\!] = \gamma^{\mathbb{F}}(\mathcal{S})$. It follows that $\mathcal{D}[\![D]\!]_{\mathcal{F}[\![D]\!]} \sqcap \mathcal{F}[\![D]\!] = \{\epsilon\}$, but this is a contradiction since $\mathcal{F}$ is a fixpoint. Thus, $\gamma^{\mathbb{F}}(\mathcal{S}^{\alpha}) \not\sqsubseteq \mathcal{F}[\![D]\!]$ and the thesis holds.

Theorem 3. *Let R be a process declaration for $p(\vec{x})$, $\mathcal{S}$ a concrete specification and $\hat{\mathcal{S}}$ a sound approximation for $\mathcal{S}$ (i.e., $\mathcal{S} \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{S}})$).*

1. *If $\mathcal{D}[\![\{R\}]\!]_{\mathcal{S}} \not\sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{S}})$ and it exists ϕ_t such that $\gamma^{\mathbb{F}}(\phi_t) \sqsubseteq \mathcal{D}[\![\{R\}]\!]_{\mathcal{S}}(p(\vec{x}))$ and $\phi_t \dot{\wedge} \hat{\mathcal{S}}(p(\vec{x})) = \dot{\mathit{false}}$, then R is abstractly incorrect (on ϕ_t) w.r.t. $\hat{\mathcal{S}}$.*
2. *If there exists an abstract uncovered element ϕ w.r.t. $\hat{\mathcal{S}}$, then there exists $r \in \gamma^{\mathbb{F}}(\phi)$ such that $r \notin \mathcal{D}[\![\{R\}]\!]_{\mathcal{S}}(p(\vec{x}))$.*

Proof. **Point 1** By hypothesis it exists ϕ_t such that $\gamma^{\mathbb{F}}(\phi_t) \sqsubseteq \mathcal{D}[\![\{R\}]\!]_{\mathcal{S}}$ and $\phi_t \dot{\wedge} \hat{\mathcal{S}} = \dot{\mathit{false}}$. Since $\mathcal{S} \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{S}})$, we have that $\mathcal{D}[\![\{R\}]\!]_{\mathcal{S}} \sqsubseteq \mathcal{D}[\![\{R\}]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{S}})}$ and, by Theorem 1, $\mathcal{D}[\![\{R\}]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{S}})} \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{D}}[\![\{R\}]\!]_{\hat{\mathcal{S}}})$. The thesis follows directly from Lemma 5, since $\phi_t \dot{\to} \hat{\mathcal{D}}[\![\{R\}]\!]_{\hat{\mathcal{S}}}$.

Point 2 By hypothesis $\phi_t \dot{\wedge} \hat{\mathcal{D}}[\![\{R\}]\!]_{\hat{\mathcal{S}}} = \dot{\mathit{false}}$. By Lemma 5 and since $\gamma^{\mathbb{F}}$ is $\bigsqcap$-distributive, it follows that $\gamma^{\mathbb{F}}(\phi_t) \sqcap \gamma^{\mathbb{F}}(\hat{\mathcal{D}}[\![\{R\}]\!]_{\hat{\mathcal{S}}}) = \{\epsilon\}$. By Theorem 1, $\mathcal{D}[\![\{R\}]\!]_{\gamma^{\mathbb{F}}(\hat{\mathcal{S}})} \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{D}}[\![\{R\}]\!]_{\hat{\mathcal{S}}})$ and, since $\mathcal{S} \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{S}})$, it follows that $\mathcal{D}[\![\{R\}]\!]_{\mathcal{S}} \sqsubseteq \gamma^{\mathbb{F}}(\hat{\mathcal{D}}[\![\{R\}]\!]_{\hat{\mathcal{S}}})$. Thus, $\gamma^{\mathbb{F}}(\phi_t) \sqcap \mathcal{D}[\![\{R\}]\!]_{\mathcal{S}} = \{\epsilon\}$ and this means that there exists $r \in \gamma^{\mathbb{F}}(\phi_t)$ such that $r \notin \mathcal{D}[\![\{R\}]\!]_{\mathcal{S}}$.

Now we present the proofs of the results presented in Section 4.1 together with some auxiliary definitions and results which are used in those proofs.

We first show that α- and β-formulas rules and the next operator preserve the satisfiability of a set of formulas.

Lemma 6. *Given a set of formulas Φ, an α-formula α and a β-formula β:*

1. *$\Phi \cup \{\alpha\}$ is satisfiable $\iff$ $\Phi \cup A(\alpha)$ is satisfiable;*
2. *$\Phi \cup \{\beta\}$ is satisfiable $\iff$ $\Phi \cup B_1(\beta)$ or $\Phi \cup B_2(\beta)$ is satisfiable;*
3. *if Φ is a set of elementary formulas, Φ is satisfiable $\iff$ $\mathsf{next}(\Phi)$ is satisfiable;*

Proof. We prove the three points separately.

1. Let us consider the rules for α-formulas in Fig. 3. Let Φ be a set of formulas, α an α-formula and $\phi, \phi_1, \phi_2 \in \mathsf{csLTL}$.

R1 Let $\alpha = \dot\neg \dot\neg \phi$, this case follows directly from the equivalence $\dot\neg \dot\neg \phi = \phi$.

R2 Let $\alpha = \phi_1 \dot\wedge \phi_2$, this case follows directly from Definition 2, in particular Equation (3.2e).

R3 Let $\alpha = \dot\neg \bigcirc \phi$, this case follows directly from the equivalence $\dot\neg \bigcirc \phi = \bigcirc \dot\neg \phi$.

2. Let us consider the rules for β-formulas in Fig. 3. Let Φ be a set of formulas, β a β-formula and $\phi_1, \phi_2 \in \mathsf{csLTL}$.

R4 Let $\beta = \dot\neg(\phi_1 \dot\wedge \phi_2)$. We show the two directions independently.

⇒ Assume that it exists $r \in \mathbf{M}$ such that $r \vDash \Phi \cup \{\dot\neg(\phi_1 \dot\wedge \phi_2)\}$. By applying De Morgan laws $r \vDash \Phi \cup \{\dot\neg \phi_1 \dot\vee \dot\neg \phi_2\}$. By Definition 2 it follows directly that $r \vDash \Phi \cup \{\dot\neg \phi_1\}$ or $r \vDash \Phi \cup \{\dot\neg \phi_2\}$.

⇐ Without lost of generality assume that it exists $r \in \mathbf{M}$ such that $r \vDash \Phi \cup \{\dot\neg \phi_1\}$. It follows that $r \vDash \Phi \cup \{\dot\neg \phi_1 \dot\vee \dot\neg \phi_2\}$ and by De Morgan laws $r \vDash \Phi \cup \{\dot\neg(\phi_1 \dot\wedge \phi_2)\}$.

R5 Let $\beta = \dot\neg(\phi_1 \,\mathcal{U}\, \phi_2)$.

⇒ Assume that it exists $r \in \mathbf{M}$ such that $r \vDash \Phi \cup \{\dot\neg(\phi_1 \,\mathcal{U}\, \phi_2)\}$. We build a model for at least one of the following sets: $\Phi \cup \{\phi_1, \dot\neg \phi_2, \dot\neg \bigcirc(\phi_1 \,\mathcal{U}\, \phi_2)\}$ and $\Phi \cup \{\dot\neg \phi_1, \dot\neg \phi_2\}$. We distinguish two cases.
In case $r \vDash \phi_1$, we have $r \vDash \Phi \cup \{\phi_1, \dot\neg(\phi_1 \,\mathcal{U}\, \phi_2)\}$, thus, by the fixpoint characterization of $\mathcal{U}$, $r \vDash \Phi \cup \{\phi_1, \dot\neg(\phi_2 \dot\vee \bigcirc(\phi_1 \,\mathcal{U}\, \phi_2))\}$. It can be notice that $\dot\neg(\phi_2 \dot\vee \bigcirc(\phi_1 \,\mathcal{U}\, \phi_2)) = \dot\neg \phi_2 \dot\wedge \dot\neg \bigcirc(\phi_1 \,\mathcal{U}\, \phi_2)$ and by Definition 2 it follows that $r \vDash \Phi \cup \{\phi_1, \dot\neg \phi_2, \dot\neg \bigcirc(\phi_1 \,\mathcal{U}\, \phi_2)\}$.
Otherwise, in case $r \nvDash \phi_1$, $r \vDash \Phi \cup \{\dot\neg \phi_1, \dot\neg(\phi_1 \,\mathcal{U}\, \phi_2)\}$. This means that $r \vDash \dot\neg \phi_1$, $r \vDash \dot\neg(\phi_1 \,\mathcal{U}\, \phi_2)$ and $r \vDash \Phi$. By definition of $\mathcal{U}$ it follows that $r \nvDash \phi_2$, otherwise $r \vDash \phi_1 \,\mathcal{U}\, \phi_2$ and this contradicts the hypothesis. Therefore, $r \vDash \dot\neg \phi_1$ and $r \vDash \dot\neg \phi_2$ and we can conclude that $r \vDash \Phi \cup \{\dot\neg \phi_1, \dot\neg \phi_2\}$.

⇐ Assume that it exists $r \in \mathbf{M}$ such that $r \vDash \Phi \cup \{\phi_1, \dot\neg \phi_2, \dot\neg \bigcirc(\phi_1 \,\mathcal{U}\, \phi_2)\}$. By definition of $\mathcal{U}$ if follows that $r \nvDash \phi_1 \,\mathcal{U}\, \phi_2$, since ϕ_2 and $\bigcirc(\phi_1 \,\mathcal{U}\, \phi_2)$ are not modeled by r. Thus, we can conclude that $r \vDash \Phi \cup \{\dot\neg(\phi_1 \,\mathcal{U}\, \phi_2)\}$.
Now assume that it exists $r \in \mathbf{M}$ such that $r \vDash \Phi \cup \{\dot\neg \phi_1, \dot\neg \phi_2\}$. Since neither ϕ_1 nor ϕ_2 are not modeled by r, it follows that $r \nvDash \phi_1 \,\mathcal{U}\, \phi_2$, thus, $r \vDash \Phi \cup \{\dot\neg(\phi_1 \,\mathcal{U}\, \phi_2)\}$.

R7 Let $\beta = \phi_1 \,\mathcal{U}\, \phi_2$ be an eventuality in the context Φ.

⇒ Assume that it exists $r \in \mathbf{M}$ such that $r \vDash \Phi \cup \{\phi_1 \,\mathcal{U}\, \phi_2\}$, we build a model for at least one of the following sets: $\Phi \cup \{\phi_2\}$ and $\Phi \cup \{\phi_1, \dot\neg \phi_2, \bigcirc((\Phi^* \dot\wedge \phi_1) \,\mathcal{U}\, \phi_2)\}$. Let $j \geq 0$ be the least j such that $r^j \vDash \phi_2$. If $j = 0$ then $r \vDash \phi_2$ and $r \vDash \Phi \cup \{\phi_2\}$. Otherwise, if $j > 0$, then $r \nvDash \phi_2$ and, by definition of $\mathcal{U}$, $r \vDash \phi_1$. Let i be the greatest index such that $0 \leq i < l$ and $r^i \vDash \Phi \cup \{\phi_1 \,\mathcal{U}\, \phi_2\}$. It follows that Φ or $\phi_1 \,\mathcal{U}\, \phi_2$ should not hold in the next time instant. Since ϕ_2 has not been reached yet we have that $r^{i+1} \vDash \phi_1 \,\mathcal{U}\, \phi_2$, thus, at least one $\phi \in \Phi$ should not be modeled by r^{i+1}. It follows that $r^i \vDash \bigcirc((\Phi^* \dot\wedge \phi_1) \,\mathcal{U}\, \phi_2)$.

⇐ We have to distinguish two cases. Assume that it exists $r \in \mathbf{M}$ such that $r \vDash \Phi \cup \{\phi_2\}$, thus, $r \vDash \Phi \cup \{\phi_1 \,\mathcal{U}\, \phi_2\}$.

Otherwise assume that it exists $r \in \mathbf{M}$ such that $r \vDash \varPhi \cup \{\phi_1, \dot{\neg}\,\phi_2, \bigcirc((\varPhi^* \dot{\wedge}\, \phi_1) \,\mathcal{U}\, \phi_2)\}$. Since $r \vDash \bigcirc((\varPhi^* \dot{\wedge}\, \phi_1) \,\mathcal{U}\, \phi_2)$ we have that $r \vDash \bigcirc(\phi_1 \,\mathcal{U}\, \phi_2)$. Thus, by definition of $\mathcal{U}$ we conclude that $r \vDash \varPhi \cup \{\phi_1 \,\mathcal{U}\, \phi_2\}$.

3. Consider the set $\varPhi = \{c_1, \ldots, c_n, \bigcirc\phi_1, \ldots, \bigcirc\phi_m, \dot{\neg}\bigcirc\psi_1, \ldots, \dot{\neg}\bigcirc\psi_k\}$, with $c_1, \ldots, c_n \in \mathcal{C}$ and $\phi_1, \ldots, \phi_m, \psi_1, \ldots, \psi_k \in \mathsf{csLTL}$. We show the two directions independently.
   - $\Rightarrow$ Assume that it exists $r \in \mathbf{M}$ such that $r \vDash \varPhi$. Let us recall that r^1 is suffix of r obtained by delete the first element of r. By Definition 2 it follows that $r \vDash c_i$ for $i = 1 \ldots n$, $r \vDash \bigcirc\phi_j$ for $j = 1 \ldots m$ and $r \nvDash \bigcirc\psi_l$ for $l = 1 \ldots k$. From monotonicity of r it follows that $r^1 \vDash c_i$ for $i = 1 \ldots n$. Moreover, by (3.2f), $r^1 \vDash \phi_j$ for $j = 1 \ldots m$ and $r^1 \nvDash \psi_l$ for $l = 1 \ldots k$. Thus it follows directly that $r^1 \vDash \mathsf{next}(\varPhi)$.
   - $\Leftarrow$ Now assume that it exists $r \in \mathbf{M}$ such that $r \vDash \mathsf{next}(\varPhi)$. Consider $C := c_1 \otimes \cdots \otimes c_n$. It is easy to notice that $r' := (C, \varnothing) \rightarrowtail C \cdot r$ is a monotone and consistent conditional trace, otherwise $r(1) \nvDash c$ and $r \nvDash \mathsf{next}(\varPhi)$. We show that $(C, \varnothing) \rightarrowtail C \cdot r$ is a model for $\varPhi$. By definition of C, is easy to notice that $(C, \varnothing) \rightarrowtail C \vDash c_i$ for $i = 1 \ldots n$. Furthermore, by (3.2f), $(C, \varnothing) \rightarrowtail C \cdot r \vDash \bigcirc\phi_j$ for $j = 1 \ldots m$ and $r \nvDash \psi_l$ for $l = 1 \ldots k$, thus $(C, \varnothing) \rightarrowtail C \cdot r \nvDash \dot{\neg}\bigcirc\psi_l$. Therefore, $(C, \varnothing) \rightarrowtail C \cdot r \vDash \varPhi$.

□

The correctness of the rule for the existential quantification derives from the following lemma, which shows that $\dot{\exists}_x\, \phi$ and ϕ are equi-satisfiable.

Lemma 7. *Let $\phi \in$ csLTL, $\dot{\exists}_x\, \phi$ is satisfiable $\iff$ ϕ satisfiable.*

Proof. We show the two directions independently.

$\Rightarrow$ This direction follows directly from (3.2h).

$$\begin{aligned}
\dot{\exists}_x\, \phi \text{ satisfiable} &\Rightarrow \text{it exists } r \in \mathbf{M}.\ r \vDash \dot{\exists}_x\, \phi \\
&\Rightarrow \text{it exists } r' \in \mathbf{M}.\ \bar{\exists}_x\, r' = \bar{\exists}_x\, r \text{ and } r' \vDash \phi \\
&\Rightarrow \phi \text{ satisfiable}
\end{aligned}$$

$\Leftarrow$ Let r be a model for ϕ, if we remove from r the information regarding x, we obtain a model $r' := \bar{\exists}_x\, r$ for $\dot{\exists}_x\, \phi$. Indeed, $\bar{\exists}_x\, r = \exists_x\, r$ ($\exists$ is idempotent) and $r \vDash \phi$, thus, by (3.2h) $r' \vDash \dot{\exists}_x\, \phi$.

□

Corollary 1. *Let $\varPhi \subseteq$ csLTL such that $x \in Var$ does not appear in $\varPhi$ and let $\phi \in$ csLTL. Then, $\varPhi \cup \{\dot{\exists}_x\, \phi\}$ is satisfiable $\iff$ $\varPhi \cup \{\phi\}$ is satisfiable.*

Proof. Follows directly from Lemma 7. x does not appear in $\varPhi$, thus the local variable x of ϕ is independent from any other variable in $\varPhi$. □

It can be noticed that the fact that x cannot appear in $\varPhi$ is not a real restriction since it is possible to perform a renaming in order to apply safely the $\dot{\exists}$ elimination without incurring in variable names crushes.

Cyclic branches in a tableau can be represented in a finite way by means of the notion of path.

Definition 10. *Let $b = n_0, n_1, \dots n_k$ be an open branch such that $L(n_k) = L(n_j)$ for $0 \leq j < k$, then b is cyclic and we define* $\mathsf{path}(b) = n_0, n_1, \dots, n_j, (n_{j+1}, \dots, n_k)^\omega$.

Every branch of a tableau is divided into stages. A *stage* is a sequence of consecutive nodes between two consecutive applications of the next operator.

Definition 11. *Given a branch b, every maximal subsequence $n_i, n_{i+1}, \dots n_j$ of* $\mathsf{path}(b)$ *is called a* stage *if, for all $i \leq l \leq j$, $L(n_l)$ is not formed only by elementary formulas or $L(n_l) \neq \mathsf{next}(L(n_{l-1}))$. We denote by* $\mathsf{stages}(b)$ *the sequence of the stages in b.*

We distinguish a particular class of stages called saturated.

Definition 12. *A stage s is saturated if and only if for every $\phi \in L(s)$:*

– *if ϕ is an α-formula then $A(\alpha) \subseteq L(s)$;*
– *if ϕ is an beta-formula then $B_1(\beta) \subseteq L(s)$ or $B_2(\beta) \subseteq L(s)$;*
– *if $\phi = \dot{\exists}_x \phi'$ with $x \in Var$ and $\phi' \in \mathsf{csLTL}$ then $\phi' \in L(s)$.*

Definition 13. *Let $\mathcal{T}_\Phi$ be a tableau and $S = s_0, s_1, \dots, s_n$ be a sequence of stages in $\mathcal{T}_\Phi$. Any eventuality $\phi_1 \,\mathcal{U}\, \phi_2 \in L(s_i)$ with $0 \leq i \leq n$ is said to be* fulfilled *in S if there exists $j \geq i$ such that $\phi_2 \in L(s_j)$.*

Intuitively, the formula is fulfilled in the path if we can reach (following the path) a node where ϕ_2 is true.

Definition 14. *A sequence of stages S is said to be* fulfilling *if and only if every eventuality occurring in S is fulfilled in S. A branch b is said to be* fulfilling *if and only if* $\mathsf{path}(\mathsf{stages}(b))$ *is fulfilling.*

Now we give the definition of expanded branch. Open expanded branches correspond to models of the initial set of formulas.

Definition 15. *An open branch b is* expanded *if and only if b is fulfilling and each stage in* $\mathsf{stages}(b)$ *is saturated.*

When constructing a tableau only non-expanded open branches are selected to be enlarged with the rules in Fig. 3. When all branches are closed or expanded the tableau cannot be further expanded.

Proposition 2. *Let $\mathcal{T}_\Phi$ be the systematic tableau for Φ, each stage s occurring in $\mathcal{T}_\Phi$ is saturated.*

Proof. By looking to Definition 9 it can be noticed that the algorithm applies any possible α-, β-rule and $\dot{\exists}$ elimination before applying the next operator to jump to the following stage. □

It can be proved that starting from a finite set of formulas $\varPhi$, the set of formulas which can occur in the construction of the systematic tableau $\mathcal{T}_{\varPhi}$ is finite. This result is the adaptation to the csLTL case of the corresponding result for PLTL shown in [GHL+09].

We denote as $\mathit{clo}(\varPhi)$ the closure of a set of formulas $\varPhi$ which contains all the formulas that can occur in any systematic tableau for $\varPhi$.

Let us first introduces some auxiliary sets of formulas which are used in the definition of $\mathit{clo}(\varPhi)$.

We denote as $\mathit{subf}(\varPhi)$ the set of sub-formulas in $\varPhi$ and their negations. $\mathit{preclo}(\varPhi)$ extends $\mathit{subf}(\varPhi)$ with the formulas that can be generated from $\mathit{subf}(\varPhi)$ by means of the rules in Fig. 3 (α, β rules and $\dot\exists$ elimination) except Rule R7.

$$\begin{aligned} \mathit{preclo}(\varPhi) := \mathit{subf}(\varPhi) \cup \{&\bigcirc(\phi_1\,\mathcal{U}\,\phi_2), \dot\neg\bigcirc(\phi_1\,\mathcal{U}\,\phi_2), \bigcirc\dot\neg(\phi_1\,\mathcal{U}\,\phi_2) \mid \phi_1\,\mathcal{U}\,\phi_2 \in \mathit{subf}(\varPhi)\} \\ &\{\bigcirc(\dot\neg\phi) \mid \dot\neg(\bigcirc\phi) \in \mathit{subf}(\varPhi)\} \cup \{\phi \mid \dot\exists_x\,\phi \in \mathit{subf}(\varPhi)\} \end{aligned}$$

$\mathit{clo}(\varPhi)$ captures the formulas generated by Rule R7 by using $\mathit{negctx}(\varPhi)$ which represents the conjunctions of negated contexts introduced by Rule R7.

$$\begin{aligned} \mathit{clo}(\varPhi) :=& \Big\{\dot{\bigwedge}\,\varDelta \mid \varDelta \subseteq \{\phi_1 \mid \phi_1\,\mathcal{U}\,\phi_2 \in \mathit{subf}(\varPhi)\} \cup \mathit{negctx}(\varPhi)\Big\} \\ &\text{where } \mathit{negctx}(\varPhi) := \{\varGamma^* \mid \varGamma \subseteq \mathit{preclo}(\varPhi)\} \end{aligned}$$

Definition 16. *Let $\varPhi$ be a set of formulas, the closure of $\varPhi$ is defined as*

$$\begin{aligned} \mathit{clo}(\varPhi) := &\,\mathit{preclo}(\varPhi) \cup \mathit{clo}(\varPhi) \\ &\cup \{(\phi_1 \dot\wedge \phi_2)\,\mathcal{U}\,\psi, \bigcirc((\phi_1 \dot\wedge \phi_2)\,\mathcal{U}\,\psi) \mid \phi\,\mathcal{U}\,\psi \in \mathit{subf}(\varPhi) \text{ and } \phi_1, \phi_2 \in \mathit{clo}(\varPhi)\} \end{aligned}$$

Proposition 3. *Let $\varPhi \subseteq$ csLTL be a finite set, then $\mathit{clo}(\varPhi)$ is also finite.*

Proof. It follows directly from Definition 16. □

The fact that $\mathit{clo}(\varPhi)$ is finite is not enough to guarantee that the algorithm terminates in a finite number of steps. It is necessary to assume that the algorithm uses a *fair strategy* to distinguish eventualities. this means that no eventuality formula in an open branch can remain non-distinguished indefinitely. A fair strategy guarantees the termination of the construction.

Let us recall some significant results shown in [GHL+09] about the handle of eventualities in the construction of the systematic tableau $\mathcal{T}_{\varPhi}$ for a set of formulas $\varPhi$.

Proposition 4. *Let s be a stage in a branch b of $\mathcal{T}_{\varPhi}$, if $\{\phi, \dot\neg\phi\} \subseteq L(s)$ then every branch prefixed by b is closed.*

Proof. It can be noticed that the application of the rules in Fig. 3 to two complementary formulas belonging to the same stage (but not necessarily to the same node) will generate two complementary formulas that belong to the same node. □

The following proposition states that non-satisfied undistinguished eventualities are kept in branches at least until they are fulfilled or they become distinguished.

Proposition 5. *Let b be a branch of $\mathcal{T}_\Phi$ and $s_0, s_1, \dots, s_k$ be a prefix of* $\mathsf{path}(\mathsf{stages}(b))$. *If $\phi_1 \,\mathcal{U}\, \phi_2 \in L(n_i)$ for some $0 \le i \le k$, $\phi_1 \,\mathcal{U}\, \phi_2$ is not distinguished in $s_i, \dots, s_k$ and $\phi_2 \notin L(s_i) \cup \dots \cup L(s_k)$, then $\{\phi_1, \dot{\neg}\phi_2, \bigcirc(\phi_1 \,\mathcal{U}\, \phi_2)\} \subseteq L(s_j)$ for all $i \le j \le k$.*

Proof. By the construction of $\mathcal{T}_\Phi$ since undistinguished eventualities are handled by Rule R7. □

The following proposition states that if a distinguished eventuality $\phi_1 \,\mathcal{U}\, \phi_2$ is not fulfilled in and expanded branch b, then b is closed, since the expansion of $\phi_1 \,\mathcal{U}\, \phi_2$ by using Rule R7, is in contradiction with the context.

Proposition 6. *Let b be a branch of $\mathcal{T}_\Phi$ and $s_0, s_1, \dots, s_k$ be a prefix of* $\mathsf{path}(\mathsf{stages}(b))$. *Consider the eventuality $\phi_1 \,\mathcal{U}\, \phi_2$, and let i be the least index such that the eventuality $\phi_1 \,\mathcal{U}\, \phi_2$ is distinguished in the stage s_i. If $\phi_2 \notin L(s_i) \cup \dots \cup L(s_k)$ then, for all $0 \le l \le k - i$, $\{\delta_l, \dot{\neg}\phi_2, \bigcirc(\delta_{l+1} \,\mathcal{U}\, \phi_2)\} \subseteq L(s_{i+l})$ where $\delta_0 = \phi_1$ and $\delta_{l+1} = \delta_l \mathbin{\dot{\wedge}} \chi$ for some $\chi \in negctx(\Phi)$.*

Moreover, if $\delta_l = \dot{\bigwedge}\Gamma$ for some Γ such that $\chi \in \Gamma$, then every maximal branch prefixed by $s_0, \dots, s_{i+l}$ is closed.

Proof. By construction of $\mathcal{T}_\Phi$, distinguished eventualities are handled by Rule R7. This rule gives rise to two branches: one containing $\{\gamma_l, \dot{\neg}\phi_2, \bigcirc(\gamma_{l+1} \,\mathcal{U}\, \phi_2)\}$ and the other containing ϕ_2. If $\bigcirc(\gamma_{l+1} \,\mathcal{U}\, \phi_2)$ is the distinguish eventuality in a successive node n on stage s_{i+l} then, in the next stage, s_{i+l} the distinguished eventuality is $\gamma_{l+1} \,\mathcal{U}\, \phi_2$ in a node n'. By Rule R7, $\gamma_0 = \phi_1$ and for all $j > 0$ $\gamma_j = \gamma_{j-1} \mathbin{\dot{\wedge}} \Delta^*_{j-1}$ where $\Delta^*_{j-1} \in negctx(\Phi)$ and Γ_{j-1} is the context $L(n) \setminus \{\bigcirc(\gamma_{l+1} \,\mathcal{U}\, \phi_2)\}$. Therefore, by induction on l, $\gamma_l \in clo(\Phi)$ for all $0 \le l \le k-1$.

Moreover we have that χ is the negation of the context of a node in s_{i+l}, if $\delta_l = \dot{\bigwedge}\Gamma$ for some Γ such that $\chi \in \Gamma$, then every branch prefixed by $s_0, \dots, s_{i+l}$ contains at the same stage two complementary formulas $\{\psi, \dot{\neg}\psi\}$. From Proposition 4 we can conclude every maximal branch prefixed by $s_0, \dots, s_{i+l}$ is closed. □

Corollary 2. *Every distinguish eventuality in a cyclic branch of $\mathcal{T}_\Phi$ is fulfilled.*

Proof. By Proposition 6 if a distinguish eventuality in a branch b is unfulfilled, then b is closed and it is not cyclic. □

Proposition 7. *Let b be a branch of $\mathcal{T}_\Phi$. b is open if and only if one of the following points holds:*

1. *the last node of b contains only constraint formulas;*
2. *b is cyclic and for every eventuality $\phi \in L(n)$ for a node occurring in b, ϕ is fulfilled in b.*

Proof. It follows directly from Point 3 and Point 7 in the algorithm of Definition 9 and from Proposition 4 and Corollary 2. □

Let us recall Lemma 2.

Lemma 2. *The algorithm of Definition 9 by using fair strategy given as input a finite set $\Phi \subseteq \mathsf{csLTL}$, terminates by building an expanded tableau for $\mathcal{T}_\Phi$.*

Proof. Suppose that the algorithm does not terminates. This means that $\mathcal{T}_\Phi$ contains an infinite branch $b = n_1, n_2, \dots, n_i \dots$. By Propositions 3, 6 and 7 this can happen only if b contains an eventuality that is never distinguished, which contradicts the fairness assumption.

The following proposition shows the behavior of negated eventualities. It is needed to prove completeness.

Proposition 8. *Let b be a branch in the systematic tableau $\mathcal{T}_\Phi$ for $\Phi \subseteq \mathsf{csLTL}$, and let s_j be a stage of the path p in the branch ($p = \mathsf{path}(b)$) such that $\dot{\neg}(\phi_1 \, \mathcal{U} \, \phi_2) \in L(s_j)$. Then, every finite subsequence of p of the form $\pi = s_j, s_{j+1}, \dots, s_k$ satisfies one of the following properties:*

1. *$\{\phi_1, \dot{\neg}\phi_2, \bigcirc \dot{\neg}(\phi_1 \, \mathcal{U} \, \phi_2)\} \subseteq L(s_i)$ for $j \leq i \leq k$.*
2. *There exists $j \leq i \leq k$ such that $\{\dot{\neg}\phi_1, \dot{\neg}\phi_2\} \subseteq L(s_i)$ and $\{\phi_1, \dot{\neg}\phi_2, \bigcirc \dot{\neg}(\phi_1 \, \mathcal{U} \, \phi_2)\} \subseteq L(s_l)$ for $j \leq l \leq i - 1$.*

Proof. We proceed by induction of $k - j$. In case $k = j$ the property follows directly from Rule R5 and since each stage of a systematic tableau is saturated (Proposition 2). In case $k > j$, by inductive hypothesis we have that $\pi' = s_j, \dots, s_{k-1}$ satisfies one of the two properties of Proposition 8. If π' satisfies Point 1 then by the saturation of the stage (Proposition 2) it follows that $\{\phi_1, \dot{\neg}\phi_2, \dot{\neg}(\phi_1 \, \mathcal{U} \, \phi_2)\} \subseteq L(s_k)$ or $\{\dot{\neg}\phi_1, \dot{\neg}\phi_2\} \subseteq L(s_k)$, thus π verifies Point 1 or Point 2 respectively. Otherwise, if π' satisfies Point 2 so does π. □

This proposition ensures that, if a node is labeled with a negated eventuality, then every node in a finite suffix of the path from that node, by construction, does not contain the second part of the eventuality (ϕ_2).

In order to prove completeness, we need an auxiliary lemma and the definition of the auxiliary function stores that, given a sequence of stages, builds a suitable conditional trace which join all the accumulated information in a stage at each time instant.

To define stores, we abuse of notation and write ϵ the empty sequence of stages. Recall that $\otimes$ is the join operation of the constraint system and $\bigotimes \emptyset = \mathit{true}$.

$$\begin{aligned}
&\mathsf{stores}(\epsilon) = \epsilon \\
&\mathsf{stores}(s \cdot S) = (C, \emptyset) \rightarrowtail C \cdot \mathsf{stores}(S) \\
&\qquad \text{where } C = \bigotimes \{c \mid c \in \mathbf{C}, c \in L(n), n \in s\}
\end{aligned}$$

By definition of next, which in our case propagates the constraints from one stage to the following, the conditional trace r resulting of applying stores to a sequence of stages S is monotone. Furthermore, since all the negative conditions are empty, r is also consistent.

We show that, a systematic tableau $\mathcal{T}_{\Phi}$ built for Φ, we can compute a model for Φ from every open branch b in $\mathcal{T}_{\Phi}$.

Lemma 8. *Let b be an* open expanded *branch in the systematic tableau $\mathcal{T}_{\Phi}$ for $\Phi \subseteq \mathsf{csLTL}$. Given the sequence of stages S in $\mathsf{path}(b)$, then $\mathsf{stores}(S) \models \Phi$.*

Proof. Let $r := \mathsf{stores}(S)$. To show that $r \models \Phi$, it is sufficient to show that for all $\phi \in \Phi$, $r \models \phi$. Note that, by Definition 9 and by the definition of stores, r contains, at each time instant, all the constraints in the labeling of the nodes in the corresponding stage. We proceed by induction on the structure of ϕ.

- Let $\phi = c$ with $c \in \mathbf{C}$; Since the first state in r contains c (which we know belongs to the labels in the first stage), then by the definition of $\models$ (Definition 2), $r \models c$.
- Let ϕ be of one of the following forms $\dot\neg\,\dot\neg\,\phi_1$, $\phi_1 \dot\wedge \phi_2$, $\dot\neg\,\phi_1 \dot\wedge \phi_2$, $\bigcirc\,\phi_1$, $\dot\neg \bigcirc \phi_1$ or $\dot\exists_x\,\phi_1$; Since every stage is saturated and by induction hypothesis on $\{\phi_1\}$, $\{\phi_1, \phi_2\}$, $\{\dot\neg\,\phi_1, \dot\neg\,\phi_2\}$, $\{\phi_1\}$, $\{\dot\neg\,\phi_1\}$ and $\{\phi_1\}$, respectively, $r \models \phi$.
- Let $\phi = \phi_1 \,\mathcal{U}\, \phi_2$, since b is an open extended branch, ϕ is fulfilled in b and, as a consequence, in $\mathsf{path}(S)$. Therefore, it exists a finite subsequence $s_0, s_1, \ldots, s_n$ of $\mathsf{path}(S)$ such that $\phi_2 \in L(s_n)$ and for all $0 \le i < n$, $\phi_1 \in L(s_i)$. By inductive hypothesis, $r^n \models \phi_2$ and for all $0 \le i < n$, $r^i \models \phi_1$. By (3.2g) in Definition 2, it follows that $r \models \phi_1 \,\mathcal{U}\, \phi_2$.
- Let $\phi = \dot\neg(\phi_1 \,\mathcal{U}\, \phi_2)$ By Proposition 8 it does not exist a finite subsequence $s_0, s_1, \ldots, s_n$ of $\mathsf{path}(\mathsf{stages}(b))$ such that $\phi_2 \in L(s_n)$ and for all $0 \le i < n$, $\phi_1 \in L(s_i)$. By inductive hypothesis, it follows that $r^n \not\models \phi_2$ or it exists $0 \le i < n$ such that $r^i \not\models \phi_1$. Thus, by (3.2g) in Definition 2 it follows that $r \not\models \phi_1 \,\mathcal{U}\, \phi_2$, and by (3.2d) $r \models \dot\neg(\phi_1 \,\mathcal{U}\, \phi_2)$.

□

Theorem 4. *$\Phi \subseteq \mathsf{csLTL}$ is unsatisfiable if and only if there exists a closed systematic tableau for Φ.*

Proof. $\Rightarrow$ Suppose that it does not exist a closed tableau for Φ, then the systematic tableau $\mathcal{T}_{\Phi}$ would be open. Let b be an open branch of $\mathcal{T}_{\Phi}$ and S its stages. By Lemma 8, $\mathsf{stores}(\mathsf{path}(S))$ is a model for Φ, thus Φ is satisfiable.

$\Leftarrow$ Let $\mathcal{T}_{\Phi}$ be the closed systematic tableau for Φ. This means that the set of formulas labeling each leaf is unsatisfiable. By the algorithm in Definition 9 and by Lemma 6, it follows that every node in $\mathcal{T}_{\Phi}$ is labeled with an unsatisfiable set of formulas. Thus, Φ is unsatisfiable.