Towards an Effective Decision Procedure for LTL formulas with Constraints*

M. Comini, L. Titolo, A. Villanueva

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

Abstract. This paper presents an ongoing work that is part of a more wide-ranging project whose final scope is to define a method to validate LTL formulas w.r.t. a program written in the timed concurrent constraint language *tccp*, which is a logic concurrent constraint language based on the concurrent constraint paradigm of Saraswat. Some inherent notions to *tccp* processes are non-determinism, dealing with partial information in states and the monotonic evolution of the information.
In order to check an LTL property for a process, our approach is based on the abstract diagnosis technique. The concluding step of this technique needs to check the validity of an LTL formula (with constraints) in an effective way.
In this paper, we present a decision method for the validity of temporal logic formulas (with constraints) built by our abstract diagnosis technique.

1 Introduction

The *ccp* paradigm 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 characteristics of the *ccp* framework is that programs can manifest non-monotonic behaviors, implying that standard approaches cannot be directly adapted. Within this family, [7] introduced the *Timed Concurrent Constraint Language* (*tccp*) 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.

Modeling and verifying concurrent systems by hand can be very complicated. Thus, the development of automatic formal methods is essential. One of the most known technique for formal verification is model checking, that was originally

* This work has been partially supported by the EU (FEDER) and the Spanish MEC/MICINN, ref. TIN 2010-21062-C02-0, and by Generalitat Valenciana, ref. PROMETEO2011/052.

introduced in [3,16] 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.

All the proposals of model checking have in common that a subset of the model of the (target) program has to be built, and sometimes the needed fragment is quite huge. Our final goal is to define a method that avoids the need to build the model of a system in order to check the validity of some temporal property. We propose an extension of the abstract diagnosis technique of [5] so that the abstract domain is formed by LTL formulas. The final step of the method consists in checking whether a given formula, built from the program (abstract) semantics and the specification, is valid.

In this work, we present a decision procedure to check the validity of such formulas. The linear temporal logic used in this work is an adaptation of the propositional LTL logic to the concurrent constraint framework, following the ideas of [15,8,9,18]. It is expressive enough to represent the abstract semantics of *tccp* with much precision and we provide a decision procedure based on the tableaux method of [11,13]. As we show through this paper, the considered logic has some differences w.r.t. the classic LTL logic due to the constraint nature and to the fact that models for *tccp* programs have some special characteristics (inherited from the *ccp* paradigm).

2 The small-step denotational semantics of *tccp*

The *tccp* language [7] is particularly suitable to specify both reactive and time critical systems. As the other languages of the *ccp* paradigm [17], 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 $\mathbf{C} = \langle \mathcal{C}, \le, \otimes, \mathit{true}, \mathit{false}, \mathit{Var}, \tilde{\exists} \rangle$ composed of a set of constraints $\mathcal{C}$ such that $(\mathcal{C}, \le)$ 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 $\tilde{\exists}$ existentially quantifies variables over constraints. The *entailment* $\vdash$ is the inverse of order $\le$.

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

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

where c, $c_1, \dots, c_n$ are finite constraints in $\mathbf{C}$; $p_{/m} \in \Pi$, $x \in \mathit{Var}$ and $\vec{x} \in \mathit{Var} \times \cdots \times \mathit{Var}$. A *tccp* program P 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 $p(\vec{x}) \coloneq A$.

[3] See [7,17] for more details on cylindric constraint systems.

The notion of time is introduced by defining a discrete and global clock. The ask, tell and process call agents take one time-unit to be executed.

Intuitively, the skip agent represents the successful termination of the agent computation. The $\mathsf{tell}(c)$ agent adds the constraint c to the current store and stops. It takes one time-unit, thus the constraint c is visible to other agents from the following time instant. The store is updated by means of the $\otimes$ operator of the constraint system. 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 holds in the current store; otherwise, if no guard is entailed by the store, the agent suspends. The agent $\mathsf{now}\ c\, \mathsf{then}\ A_1\ \mathsf{else}\ A_2$ behaves in the current time instant like A_1 (respectively A_2) if c is (respectively is not) entailed by the store. Entailment is checked by using the $\vdash$ operator of the constraint system. Note that this agent can process negative information: it can capture when some information is not present in the store since the agent A_2 is executed both when $\neg c$ is entailed, but also when neither c nor $\neg c$ are entailed. $A_1 \parallel A_2$ models the parallel composition of A_1 and A_2 in terms of maximal parallelism, i.e., all the enabled agents of A_1 and A_2 are executed at the same time. The agent $\exists x\, A$ is used to make variable x local to A. To this end, it uses the $\tilde{\exists}$ operator of the constraint system. Finally, the agent $p(\vec{x})$ takes from D a declaration of the form $p(\vec{x}) \coloneq A$ and executes A at the following time instant. We assume that the set D of declarations is closed w.r.t. parameter names.

In this work, we refer to the denotational concrete semantics defined in [5] which is fully-abstract w.r.t. the small-step behavior of *tccp*. Due to space limitations, we just introduce intuitively the most relevant aspects of such semantics and provide a global view of it by means of an example. The missing definitions, as well as the proofs of all the results, can be found in [6].

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). The set of all conditional timed traces is denoted with $\mathbf{M}$. Intuitively, conditional traces can be seen as hypothetical computations in which, for each time instant, a 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 $\mathit{stutt}(C)$, with $C \subseteq \mathbf{C} \setminus \{\mathit{true}\}$;
end of a process: the construct $\boxtimes$.

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 $\forall c^- \in \eta^-,\ \eta^+ \nvdash c^-$.

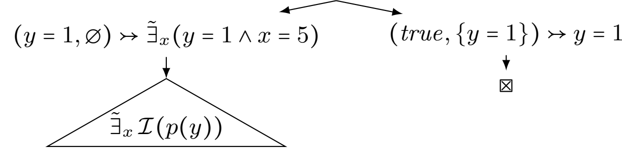

Fig. 1. Tree representation of $\mathcal{A}[\![A]\!]_{\mathcal{I}}$ of Example 1.

Conditional states in a conditional trace have to be 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 must not entail its associated negative conditions).

We abuse in notation and define as $\tilde{\exists}_x r$ the sequence resulting by removing from r all the information about the variable x. Moreover, $r \in \mathbf{M}$ is said to be *self-sufficient* if the first condition is $(true, \varnothing)$ 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.* $x \in \mathcal{V}$ (*x-self-sufficient*) if $\tilde{\exists}_{Var \setminus \{x\}} r$ is self-sufficient. Intuitively, for self-sufficient conditional traces, no additional information (from other agents) is needed in order to *complete* the computation.

Example 1. Let us consider a program with a single process declaration $D := \{p(y) :- A\}$, where

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

The semantics of A is graphically represented in Fig. 1. The left branch represents the computation when the information $y = 1$ is entailed by the store (the positive condition requires $y = 1$), thus the information added by the tell agent ($x = 5$) is joined to the store and, in the following state, since it is invoked a (recursive) process call, we find the interpretation of the process called (represented by the triangle labeled with the interpretation $\mathcal{I}$). Process calls do not modify the store when invoked, but they affect the store from the following time instant. The right branch is taken only if $y = 1$ does not hold in the current state, then it holds at the following time instant due to the tell agent.

3 Constraint System Linear Temporal Logic

In this section, we define a variation of the classical Linear Temporal Logic [14]. Following [15,8,9,18], the idea is to replace atomic propositions by constraints of the underlying constraint system. This logic is the basis for the definition of the abstract semantics needed in our abstract diagnosis technique.

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

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

We denote with csLTL *the set of all temporal formulas over* $\mathbf{C}$*.*

The formulas $\dot{true}$, $\dot{false}$, $\dot{\neg}\,\phi$, and $\phi_1 \mathbin{\dot\wedge} \phi_2$ 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, we use $\phi_1 \mathbin{\dot\vee} \phi_2$ as a shorthand for $\dot{\neg}(\dot{\neg}\,\phi_1 \mathbin{\dot\wedge} \dot{\neg}\,\phi_2)$; $\phi_1 \mathbin{\dot\rightarrow} \phi_2$ for $\dot{\neg}\,\phi_1 \mathbin{\dot\vee} \phi_2$; $\Diamond\, \phi$ for $\dot{true}\, \mathcal{U}\, \phi$ and $\Box\, \phi$ for $\dot{\neg}\, \Diamond\, \dot{\neg}\,\phi$. A *constraint formula* is an atomic formulas 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*.

The truth of a formula $\phi \in \mathsf{csLTL}$ is defined w.r.t. a trace $r \in \mathbf{M}$. As usually done in the context of temporal logics, we define the satisfaction relation $\models$ only for infinite conditional traces. We implicitly transform finite traces (which end in $\boxtimes$) by replicating the last store infinite times.

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

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

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

$$r \models \dot{true} \tag{3.2a}$$

$$r \not\models \dot{false} \tag{3.2b}$$

$$(\eta^+, \eta^-) \rightarrowtail d \cdot r' \models c \quad \textit{iff } \eta^+ \vdash c \tag{3.2c}$$

$$stutt(\eta^-) \cdot r' \models c \quad \textit{iff } \forall d^- \in \eta^- .\, c \nvdash d^- \textit{ and } r' \models c \tag{3.2d}$$

$$r \models \dot{\neg}\,\phi \quad \textit{iff } r \not\models \phi \tag{3.2e}$$

$$r \models \phi_1 \mathbin{\dot\wedge} \phi_2 \quad \textit{iff } r \models \phi_1 \textit{ and } r \models \phi_2 \tag{3.2f}$$

$$r \models \dot{\exists}_x\, \phi \quad \textit{iff } \exists r' \textit{ s.t.} \tilde{\exists}_x\, r' = \tilde{\exists}_x\, r,\ r' \textit{ x-self-sufficient, } r' \models \phi \tag{3.2g}$$

$$t \cdot r \models \bigcirc\, \phi \quad \textit{iff } r \models \phi \tag{3.2h}$$

$$r \models \phi_1\, \mathcal{U}\, \phi_2 \quad \textit{iff } \exists i \geq 1.\, \forall j < i.\ r^i \models \phi_2 \textit{ and } r^j \models \phi_1 \tag{3.2i}$$

We abuse of notation and we extend $\models$ to sets of formulas: $r \models \Phi \iff \forall \phi \in \Phi.\ r \models \phi$ A formula ϕ is said to be satisfiable *if there exists $r \in \mathbf{M}$ such that $r \models \phi$, while it is said to be* valid *if, for all $r \in \mathbf{M}$, $r \models \phi$.*

Example 2. The formula $\Box(x > 0 \mathbin{\dot\rightarrow} \bigcirc\, z > 1)$ expresses that forever, whenever $x > 0$ is entailed by the store, then $z > 1$ is entailed at the following time instant.

4 Abstract diagnosis of temporal properties

Abstract diagnosis is a semantic based method to identify bugs in programs. It was originally defined for logic programming [4] and then extended to other paradigms [1,2,10,5]. This technique is based on the definition of an abstract semantics for the program which must be a sound approximation of its behavior.

Then, given an abstract specification $\mathcal{S}$ of the expected behavior of the program, the abstract diagnosis technique automatically detects the errors in the program by checking if the result of one computation of the semantics evaluation function (where the procedure calls are interpreted over $\mathcal{S}$) is "contained" in the specification itself.

A first approach to the abstract diagnosis of *tccp* was presented in [5] by using as specifications sets of abstract traces. The main drawback of that proposal was that specifications were given in terms of traces, thus they can be tedious to write. In order to overcome that problem, we have defined an abstract semantics in terms of csLTL formulas. This allows us to express the intended behavior in a more compact way, by means of a csLTL formula.

The semantics evaluation function $\mathcal{A}[\![A]\!]$, given an agent A and an interpretation $\mathcal{I}$ (for the process symbols of A), builds a csLTL formula representing a correct approximation of the small-step behavior of A.The semantics of the declarations is given in terms of the fixpoint of a semantics evaluation function $\mathcal{D}[\![D]\!]$ which associates to each procedure declaration $p(\vec{x})$ the logic disjunction of the abstract semantics of every agent A such that $p(\vec{x}) :- A$ belongs to the declaration D (i.e., $\mathcal{D}[\![D]\!]_{\mathcal{I}}(p(\vec{x})) \coloneqq \dot{\bigvee}_{p(\vec{x}):-A\in D} \mathcal{A}[\![A]\!]_{\mathcal{I}}$).

The *abstract diagnosis* technique determines exactly the "originating" symptoms and, in the case of incorrectness, the faulty process declaration in the program. This is captured by the definitions of *abstractly incorrect process declaration* and *abstract uncovered element*. Informally, a process declaration D is abstractly incorrect if it derives a wrong abstract element $\phi_t \in$ csLTL from the intended semantics $\mathcal{S}$. Dually, ϕ_t is uncovered if the declarations cannot derive it from the intended semantics.

We show here the main result of abstract diagnosis, which determines the form of formulas that we need to check for validity.

Theorem 1. *Consider a set of declarations D and an abstract specification $\mathcal{S}$.*

1. *If there are no abstractly incorrect process declarations in D (i.e., $\mathcal{D}[\![D]\!]_{\mathcal{S}} \dot{\rightarrow} \mathcal{S}$), then D is partially correct w.r.t. $\mathcal{S}$ (the small-step denotational semantics of D is "contained "in the semantics of $\mathcal{S}$).*
2. *Let D be partially correct w.r.t. $\mathcal{S}$. If D has abstract uncovered elements then D is not complete.*

Therefore, in order to check partial correctness of a program, it is sufficient to check the implication $\mathcal{D}[\![D]\!]_{\mathcal{S}} \dot{\rightarrow} \mathcal{S}$.

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; therefore, no (manual) inspection is needed for declarations which are not signalled. Moreover, it happens that all concrete errors—that are "visible"—are detected, as they lead to an abstract incorrectness or abstract uncovered.

Example 3. Assume we need to check that the program in Example 1 satisfies that the constraint $y = 1$ is eventually entailed by the store. The intended specification for the process p corresponding to this property is $\mathcal{S}(p(y)) \coloneqq \Diamond(y = 1)$. The csLTL-semantics $\mathcal{D}$ for $p(y)$ with the given specification as interpretation is

$$\mathcal{D}[\![D]\!]_{\mathcal{S}}(p(y)) = \dot{\exists}_x \big((y = 1 \mathbin{\dot{\wedge}} \bigcirc x = 5 \mathbin{\dot{\wedge}} \bigcirc(\Diamond\, y = 1)) \mathbin{\dot{\vee}} (\dot{\neg}\, y = 1 \mathbin{\dot{\wedge}} \bigcirc y = 1)\big)$$

Note that the resulting formula has a clear correspondence with the behavior of the program. We have two disjuncts, one for each branch of the conditional in the body of the declaration. Moreover, since the conditional agent is in the scope of a local variable x, both disjuncts are enclosed within an existential quantification. The computation of the csLTL-semantics is compositional, based on the structure of the program. The first disjunct corresponds to the case when the guard of the conditional agent is satisfied ($y = 1$), thus in the following time instant two things happen: $x = 5$ is entailed, and also is entailed the interpretation for the process call p because a recursive call is run. This illustrates how the intended specification is used as the interpretation of process calls.

Following Theorem 1, to check whether the process $p(y)$ satisfies the property, it is sufficient to show that $\mathcal{D}[\![D]\!]_{\mathcal{S}}(p(y)) \mathbin{\dot{\rightarrow}} \mathcal{S}(p(y))$ is valid.

The fact that our technique is based on abstract diagnosis becomes explicit when two situations occur simultaneously. When the process that is being analyzed has more than one fixpoint (this essentially happens when D contains a loop which does not produce contributes at all) and the specification is an eventuality that is not inconsistent with the program behavior, then it may happen that the actual behaviour does not model a specification $\mathcal{S}$, which is a non-least fixpoint of $\mathcal{D}[\![D]\!]$, but we do not detect abstractly incorrect declarations since $\mathcal{S}$ is a fixpoint. This is natural in the context of the abstract diagnosis technique: what we are proving is that, if $\mathcal{S}(p(\vec{x}))$ is assumed to hold for each process $p(\vec{x})$ defined in D and $\mathcal{D}[\![D]\!]_{\mathcal{S}} \mathbin{\dot{\rightarrow}} \mathcal{S}$, then *the program* $\mathcal{F}[\![D]\!]$ satisfies $\mathcal{S}$.

In any other situation, a positive result of the abstract diagnosis corresponds to proving that the program *strongly* satisfies the temporal property. Also if we provide a specification that is in contradiction with the actual behavior of the process, our technique answers as expected in a verification context.

5 An automatic decision procedure for csLTL

In order to make our abstract diagnosis approach effective, we need to define an automatic decision procedure to check the validity of the csLTL formulas that show up when checking a property. In particular, we need to handle csLTL formulas of the form $\psi \mathbin{\dot{\rightarrow}} \phi$, where ψ corresponds to the computed approximated behavior of the program, and ϕ is the abstract intended behavior of the process.

We impose a restriction on the specification ϕ: we do not allow the use of existential quantifications. Actually, this restriction is quite natural 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. In contrast, negation can be applied to any formula ϕ (not only to constraints).

In this section, we extend the tableau construction for Propositional LTL (PLTL) of [11,13] in order to deal with csLTL formulas. We need to adapt the method to our context due to three issues:

1. The structures on which the logic is interpreted are different. In our case, traces (sequences of states) are monotonic, meaning that the information in each state always increases.
2. The logic itself is a bit different from PLTL since propositions are replaced by constraints in $\mathbf{C}$.
3. We have to handle existential quantification over variables of the underlying constraint system. This does not mean that we are dealing with a first-order logic as will become clear later

In the following, we first present the basic rules that are used during the construction of the tree associated to the tableau. Then we present the algorithm that implements the process of construction of the tree.

5.1 Basic rules for a csLTL tableau

Classic tableaux algorithms are based on the systematic construction of a graph which is used to check the satisfiability of the formula. In [11,13], the authors present an algorithm that does not need to use auxiliary structures such as graphs to decide about the satisfaction of the formula, and this makes this approach more suitable for automatization.

A tableau procedure is defined by means of rules that build a tree whose nodes are labeled with sets of formulas. If all branches of the tree are *closed*, then the formula has no models. Otherwise, we can obtain a model that satisfies the formula from the open branches. Let us introduce the basic rules for the csLTL case. As usual, we present just the minimal set of rules.

A tableau rule is applied to a node n labeled with the set of formulas $L(n)$. Each rule application requires a previous selection of a formula ϕ from $L(n)$. We call context to the set of formulas $L(n) \setminus \{\phi\}$ and we denote it with Γ. Conjunctions are α-formulas and disjunctions β-formulas. Fig. 2 presents the rules for $\alpha-$ and $\beta-$formulas.

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

	β	$B_1(\beta)$	$B_2(\beta)$
R3	$\dot{\neg}(\phi_1 \mathbin{\dot{\wedge}} \phi_2)$	$\{\dot{\neg}\,\phi_1\}$	$\{\dot{\neg}\,\phi_2\}$
R4	$\dot{\neg}(\phi_1 \,\mathcal{U}\, \phi_2)$	$\{\dot{\neg}\,\phi_1, \dot{\neg}\,\phi_2\}$	$\{\phi_1, \dot{\neg}\,\phi_2, \dot{\neg}\bigcirc(\phi_1 \,\mathcal{U}\, \phi_2)\}$
R5	$\phi_1 \,\mathcal{U}\, \phi_2$	$\{\phi_2\}$	$\{\phi_1, \dot{\neg}\,\phi_2, \bigcirc(\phi_1 \,\mathcal{U}\, \phi_2)\}$
R6	$\phi_1 \,\mathcal{U}\, \phi_2$	$\{\phi_2\}$	$\{\phi_1, \dot{\neg}\,\phi_2, \bigcirc((\Gamma^* \mathbin{\dot{\wedge}} \phi_1) \,\mathcal{U}\, \phi_2)\}$

Fig. 2. α- and β-formulas rules

Tables in Fig. 2 are interpreted as follows. Each row in a table represents a rule. Each time that an α–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 R6 uses the so-called context Γ^*, which is defined as $\Gamma^* := \dot{\bigvee}_{\gamma \in \Gamma} \dot{\neg} \gamma$. The use of contexts is the mechanism to detect the loops where no formula changes, thus allowing to mark branches containing eventually formulas as *open*. This kind of rules were first used in [12].

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 [11,13]. This operator allows one to identify *stages* in a tableau which represent time instants in the model.

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

Lemma 1. *Given a set of formulas Φ, an α-formula α and a β-formula β:*

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

A second main difference w.r.t. the PLTL case regards the existential quantification. The csLTL existential quantification 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: 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. Correctness of this rule derives from the following lemma, which shows that $\dot{\exists}_x \, \phi$ and ϕ are equi-satisfiable.

Lemma 2. *Let $\phi \in$ csLTL, $\dot{\exists}_x \, \phi$ is satisfiable $\iff$ ϕ satisfiable.*

5.2 Semantic csLTL tableaux

In this section, we present the notion of tableau for our csLTL formulas following the ideas of [11,13]. Since we borrow some definitions and notions from that work, in this section we skip some formal definitions.

A tableau $\mathcal{T}_\Phi$ for a set of formulas Φ is a tree-like structure where each node n is labeled with a set of csLTL formulas $L(n)$. The root is labeled with the

set of formulas Φ whose satisfiability/unsatisfiability is needed to check; Then, children of nodes are the result of applying the basic rules of Subsection 5.1. The algorithm in which these nodes are built is given in the following subsection. Nodes with no children are called *leaf* nodes.

Definition 3 (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 $b = n_0, \dots, n_i, n_{i+1}, \dots, n_k \in B$ and every $0 \le i < k$:*
   - *(a) $L(n_{i+1}) = \{A(\alpha)\} \cup L(n_i) \smallsetminus \{\alpha\}$ for an α-formula $\alpha \in L(n_i)$;*
   - *(b) $L(n_{i+1}) = \{B_1(\beta)\} \cup L(n_i) \smallsetminus \{\beta\}$ and there exists $b' = n_0, \dots, n_i, n'_{i+1}, \dots, n'_k$ such that $L(n'_{i+1}) = \{B_2(\beta)\} \cup L(n_i) \smallsetminus \{\beta\}$ for a β-formula $\beta \in L(n_i)$;*
   - *(c) $L(n_{i+1}) = \{\phi'\} \cup L(n_i) \smallsetminus \{\dot{\exists}_x \phi'\}$ for $\dot{\exists}_x \phi' \in L(n_i)$;*
   - *(d) $L(n_{i+1}) = \mathsf{next}(L(n_i))$ if $L(n_i)$ contains only elementary formulas.*

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

Definition 4. *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 couple of constraint formulas $c, \dot{\neg} c'$ such that $c \vdash c'$.*

The last condition for inconsistence of a node is particular to the *ccp* context. Since we are dealing with constraints that model partial information, it is possible to have an *implicit* inconsistence, in the sense that we need the entailment relation to detect it.

An inconsistent node does not accept any rule application. When a branch contains an inconsistent node, it is said to be closed, otherwise it is open.

By Lemma 1 and by Definition 3, it can be noticed that every closed branch contains only unsatisfiable sets of formulas. Open branches are not necessarily satisfiable since they could be prefixes of a closed one.

Similarly to the PLTL case, it exists only a finite number of different labels in a tableau. Thus, an infinite branch $b = n_0, n_1, \dots n_k \dots$ contains a cycle. These branches are called *cyclic branches* and can be finitely represented as $\mathsf{path}(b) = n_0, n_1, \dots, n_j, (n_{j+1}, \dots, n_k)^\omega$ when $L(n_k) = L(n_j)$ for $0 \le j < k$.

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. We abuse of notation and we say that the labeling of a stage s is the labeling of each node in that stage ($L(s) = \bigcup_{n \in s} L(n)$). Moreover, a stage s is *saturated* if no α-, β- or hiding rule can be applied to any of its nodes.

We borrow from [11,13] the characterization of *fulfilled* eventually formula in a path of the tableau, namely when it is satisfied. We say that, when an

eventually formula $\phi_1 \mathcal{U} \phi_2$ belongs to the labeling of a stage s in a path, 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 eventually formulas it its labeling are fulfilled in S and a branch b is fulfilling if the sequences of stages in its paths are fulfilling.

Finally, an open branch is expanded if it is fulfilling and all its stages are saturated. 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 5 (expanded csLTL tableau [11,13]). *A tableau is called expanded if every branch is expanded or closed. An expanded tableau is closed if every branch ends in an inconsistent node, otherwise it is open.*

5.3 A systematic csLTL tableaux construction

Definition 6 presents the algorithm to automatically build an expanded csLTL tableau (called systematic tableau [11,13]) for a given set of formulas Φ.

The construction consists in selecting at each step a non-expanded branch to be enlarge 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 which rule R5 or R6 has to be applied in a node, it is necessary to *distinguish* the eventuality that is being unfolded in the path. In this way, the rule R6 is applied only to *distinguished* eventualities when selected; when the selected eventuality is not the distinguished one, then rule R5 is used. If a node does not contain any distinguished eventuality, then the algorithm distinguishes one of them and rule R6 is chosen to be applied to it. Each node of the tableau has at most one distinguished eventuality.

The algorithm marks a node as *closed* when it is inconsistent and as *open* when it contains just constraint formulas or when it is the last node of an expanded branch (all the eventualities in the path are fulfilled).

Definition 6. *Given as input a finite set of formulas Φ, the algorithm repeatedly selects an unmarked leaf node l labelled with the set of formulas $L(l)$ and applies, in order, one of the points shown below.*

1. *If l is an inconsistent node, then mark it as closed ($\times$).*
2. *If $L(l)$ is a set of constraint formulas, mark l as open ($\odot$).*
3. *If $L(l) = L(l')$ for l' ancestor of l, take the oldest ancestor l'' of l that is labeled with $L(l)$ and check if each eventuality in the path between l'' and l is fulfilled in such path. If they are all fulfilled, then mark l as open ($\odot$).*
4. *Otherwise, choose $\phi \in L(l)$ such that ϕ is not a* next *formula. Then,*
   - *if ϕ is an α-formula (let $\phi = \alpha$), create a new node l' as a child of l and label it as $L(l') = (L(l) \setminus \{\alpha\}) \cup A(\alpha)$ by using the α–rules in Fig. 2,*
   - *if ϕ is a β-formula (let $\phi = \beta$), create two new nodes l' and l'' as children of l and label them as $L(l') = (L(l) \setminus \{\beta\}) \cup B_1(\beta)$ and $L(l'') = (L(l) \setminus \{\beta\}) \cup B_2(\beta)$ by using the corresponding rule in Fig. 2. Moreover, if β is an eventuality, we have three possible cases:*

- *if β is the distinguished eventuality in $L(l)$, then apply Rule R6 to β and distinguish the formula inside the* next *formula in $B_2(\beta)$;*
- *if β is not distinguished, but there is another distinguished formula, then apply Rule R5 to β and maintain the existing distinguished formula in $B_1(\beta)$ and $B_2(\beta)$;*
- *otherwise, distinguish β and apply Rule R6 to β and distinguish the formula inside the* next *formula in $B_2(\beta)$;*

– *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 $L(l') = (L(l) \setminus \{\phi\}) \cup \{\phi'\}$*

5. *If $L(l)$ is a set of elementary formulas, apply the* next *operator: create a new node l' as child of l and label it as $L(l') = \mathsf{next}(L(l))$.*

The construction terminates when every branch is marked.

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 exist 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. 2, 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 kept until they are fulfilled or they become distinguished.

One key result of the tableau in [13] and that we borrow is that if a distinguished eventuality is not fulfilled in an expanded branch b, then we can mark the branch as closed. This is because if we apply Rule R6, then we get a contradiction with the context of the eventuality.

As a conclusion, we have that every distinguished eventuality in a cyclic branch b of $\mathcal{T}_\Phi$ is fulfilled just since if it were unfulfilled, then b would be closed (thus not cyclic). Also, by construction and the above properties, b is open if and only if (1) the last node of b contains only constraint formulas, or (2) b is cyclic and all its eventualities are fulfilled in b.

Lemma 3. *By using a fair strategy and given as input a finite set $\Phi \subseteq$ csLTL, the algorithm of Definition 6, terminates and builds an expanded tableau for $\mathcal{T}_\Phi$.*

5.4 Soundness and completeness

Let us now show that the proposed algorithm is sound and complete for proving the satisfiability/unsatisfiability of csLTL formulas.

Theorem 2 (soundness). *If there exists a closed systematic tableau for $\Phi \subseteq$ csLTL, then Φ is unsatisfiable.*

In order to prove completeness, we need to define an auxiliary function stores that, given a sequence of stages, builds a suitable conditional trace which joins

all the accumulated information in a stage at each time instant. We abuse of notation and write ϵ the empty sequence of stages. Recall that $\otimes$ is the join operation of the constraint system and $\bigotimes \varnothing = \mathit{true}$.

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

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, given 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 4. *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 $\mathsf{stores}(S) \vDash \Phi$.*

Theorem 3 (refutational completeness). *If $\Phi \subseteq \mathsf{csLTL}$ is unsatisfiable, then there exists a closed tableau for Φ.*

Theorem 4 (completeness). *If $\Phi \subseteq \mathsf{csLTL}$ is satisfiable, then there exists a finite open tableau for Φ.*

5.5 Application of the Tableau

We are interested in checking the validity of a formula of the form $\psi \dot{\rightarrow} \phi$. Our strategy is to build the tableau for its negation $\mathcal{T}_{\dot{\neg}(\psi \dot{\rightarrow} \phi)}$ so that, if $\mathcal{T}_{\dot{\neg}(\psi \dot{\rightarrow} \phi)}$ is closed, meaning that $\dot{\neg}(\psi \dot{\rightarrow} \phi)$ is unsatisfiable, then our implication is valid.

Let us show two examples of construction of the systematic tableaux for two formulas of this kind.

Example 4. Consider the formula of our guiding example $\dot{\exists}_x \phi \dot{\rightarrow} \Diamond(y = 1)$, where $\phi = (y = 1 \mathrel{\dot{\wedge}} \bigcirc x = 5 \mathrel{\dot{\wedge}} \bigcirc(\Diamond y = 1)) \mathrel{\dot{\vee}} (\dot{\neg} y = 1 \mathrel{\dot{\wedge}} \bigcirc y = 1)$. The tableau in Fig. 3, with $L(\mathit{root}) = \dot{\exists}_x \phi \mathrel{\dot{\wedge}} \Box \dot{\neg}(y = 1)$ shows its validity. 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.

In the example, the first step uses the rule for the conjunction. Then, the second step involves the elimination of the existential quantification for $\dot{\exists}_x \phi$. Since the formula $\Box(\dot{\neg} y = 1)$, which represents the context, does not contain information about x, $\dot{\exists}_x \phi$ can be replaced with ϕ. The formula ϕ is then selected for a β step (disjunction). The branch on the left is closed after two steps since $y = 1$ and $\dot{\neg} y = 1$ both belong to the node labeling.

The branch on the right, first flattens the conjunction and then applies the next rule. Note that the negation of a constraint is not kept in the following time instant. We recall that negation means "not entailment" (in contrast to meaning that *the contrary is true*), thus, in the future, the constraint could become true.

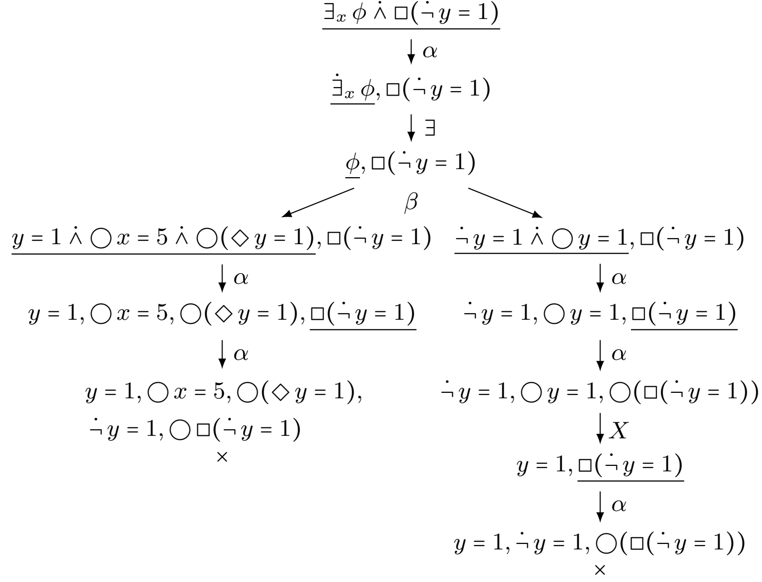

Fig. 3. Tableau for $\dot{\exists}_x\, \phi \mathrel{\dot{\rightarrow}} \Diamond\, y = 1$ of Example 4.

Since both branches are closed, we know that the formula $\dot{\exists}_x\, \phi \mathrel{\dot{\wedge}} \Box \dot{\neg}(y = 1)$ is not satisfiable, thus its negation $\dot{\exists}_x\, \phi \mathrel{\dot{\rightarrow}} \Diamond(y = 1)$ is valid.

In the context of abstract diagnosis, this proves that the program is abstractly correct w.r.t. the LTL specification.

Example 5. Now, suppose that we want to check, for the program introduced in Example 1, that the constraint $y = 1$ is always entailed by the store. The corresponding specification is $\mathcal{S}'(p(y)) = \Box(y = 1)$.

The csLTL-semantics $\mathcal{D}$ for $p(y)$ using the given specification as interpretation is $\dot{\exists}_x \big((y = 1 \mathrel{\dot{\wedge}} \bigcirc x = 5 \mathrel{\dot{\wedge}} \bigcirc(\Box\, y = 1)) \mathrel{\dot{\vee}} (\dot{\neg}\, y = 1 \mathrel{\dot{\wedge}} \bigcirc\, y = 1)\big)$. Let us abbreviate the body of the existential quantification as ϕ'. To check whether the process $p(y)$ is correct w.r.t. the property, we have to show that $\dot{\exists}_x\, \phi' \mathrel{\dot{\rightarrow}} \Box(y = 1)$ is valid.

Fig. 4 shows part of the (finite) tableau that proves the satisfiability of the formula $\dot{\exists}_x\, \phi' \mathrel{\dot{\wedge}} \Diamond(\dot{\neg}\, y = 1)$. This means that its negation, $\dot{\exists}_x\, \phi' \mathrel{\dot{\rightarrow}} \Box(y = 1)$, is not valid. In the context of abstract diagnosis, although the formula is actually not satisfied by the program, because of the loss of precision due to the approximation, this is only a warning about the possible incorrectness of the program w.r.t. the LTL specification.

6 How to handle streams

In *tccp* streams are used to model imperative-style variables. In this context, when specifying an intended behavior, we are interested in the *current* value in a stream S (i.e., the last instantiated value in the stream), denoted as $S \doteq \mathit{value}$.

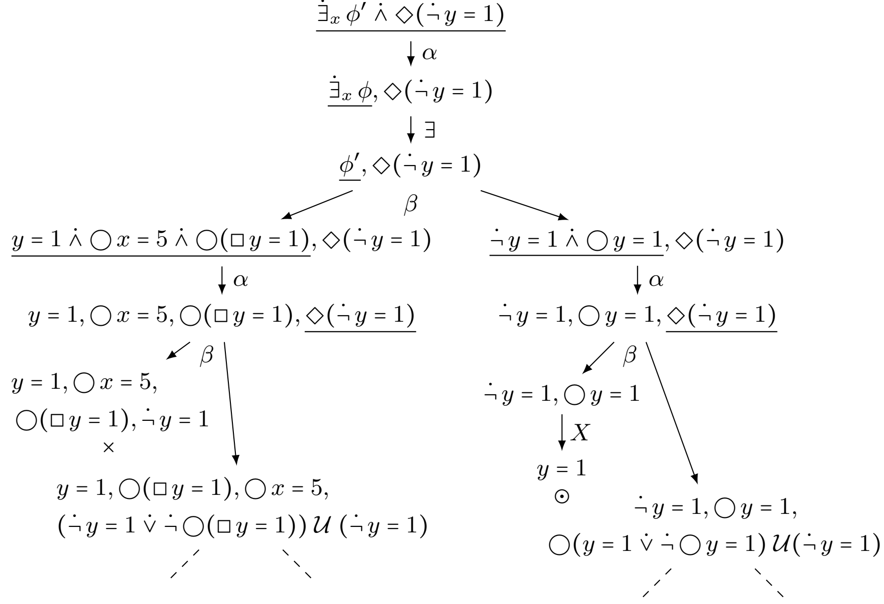

Fig. 4. Tableau for $\dot{\exists}_x \phi' \dot{\rightarrow} \Box y = 1$ of Example 5

In order to deal in our tableau with this special kind of constraint, we define a stream simplification function σ that transforms a formula ϕ into a new formula ϕ' that contains information only about the last instantiated value of the streams in ϕ. We show that the transformation preserves satisfiability.

The function σ uses two auxiliary functions. $dep(\phi)$ generates a set of dependencies on the form (S, S'), where S is a stream and S' its tail in the formula ϕ, while $head(S, D)$ returns, given a set of dependencies D, the first name given to the stream S. For instance, for $\phi = ((S = [v \mid S']) \dot{\wedge} (S' = [v' \mid S''])) \,\mathcal{U}\, (T = [u \mid T'])$, $dep(\phi) = \{(S, S'), (S', S''), (T, T')\}$ and $head(S'', dep(\phi)) = S$. Formally,

$$dep(\phi) \coloneqq \begin{cases} \varnothing & \text{if } \phi = \dot{true},\ \phi = \dot{false} \text{ or } \phi = c \\ \{(S, S')\} & \text{if } \phi = (S = [v \mid S']) \\ dep(\phi_1) & \text{if } \phi = \dot{\exists}_x \phi_1 \text{ or } \phi = \bigcirc \phi_1 \\ dep(\phi_1) \cup dep(\phi_2) & \text{if } \phi = \phi_1 \dot{\wedge} \phi_2 \text{ or } \phi = \phi_1 \,\mathcal{U}\, \phi_2 \end{cases}$$

$$head(S, D) \coloneqq \begin{cases} S & \text{if } D = \varnothing \\ head(S', D') & \text{if } D = \{(S', S)\} \cup D' \\ head(S, D') & \text{if } D = \{(T', T)\} \cup D' \text{ and } T \neq S \end{cases}$$

Definition 7. *Let $\phi, \phi_1, \phi_2 \in$ csLTL, S, S', R, R' streams and v a stream value. σ is defined inductively as follows.*

$$\sigma(\phi) := \begin{cases} \phi & \text{if } \phi = \dot{true},\ \phi = \dot{false} \text{ or } \phi = c \\ S \doteq c & \text{if } \phi = (S' = [v \mid S'']) \text{ and } head(S', dep(\phi)) = S \\ \dot{\neg}\, \sigma(\phi_1) & \text{if } \phi = \dot{\neg}\, \phi_1 \\ \sigma(\phi_1) \dot{\wedge} \sigma(\phi_2) & \text{if } \phi = \phi_1 \dot{\wedge} \phi_2 \\ \dot{\exists}_x\, \sigma(\phi_1) & \text{if } \phi = \dot{\exists}_x\, \phi_1 \\ \bigcirc \sigma(\phi_1) & \text{if } \phi = \bigcirc \phi_1 \\ \sigma(\phi_1)\, \mathcal{U}\, \sigma(\phi_2) & \text{if } \phi = \phi_1\, \mathcal{U}\, \phi_2 \end{cases}$$

Example 6. Consider $\phi = (C = [near \mid C'] \dot{\wedge} \bigcirc(C' = [out \mid C'']) \dot{\wedge} G = [down \mid G'])$, then $\sigma(\phi) = (C \doteq near \dot{\wedge} \bigcirc(C \doteq out) \dot{\wedge} G \doteq down)$.

The transformation σ preserves satisfiability, thus it can be used to preprocess the initial formula before applying the tableau method.

Lemma 5. *Let $\phi \in$ csLTL, ϕ is satisfiable $\iff$ $\sigma(\phi)$ is satisfiable.*

If we need to use this transformation, then the next operator must be slighly modified: $\mathsf{next}(\Phi) := \{\phi \mid \bigcirc \phi \in \Phi\} \cup \{\dot{\neg}\phi \mid \dot{\neg}\bigcirc \phi \in \Phi\} \cup \{c \mid c \in \Phi, c \in \mathbf{C}\} \cup \{S \doteq v_1 \mid S \doteq v_1 \in \Phi$ and $\nexists\, v_2$ such that $\bigcirc(S \doteq v_2) \in \Phi$ and $v_1 \neq v_2\}$. Intuitively, the constraints on the form $S \doteq v_1$ are propagated only if in the next time instant the tail of S has not be instantiated with a different constraint v_2. This definition of next preserves satisfiability of Lemma 1.

7 Conclusions and Future Work

In this paper, we have introduced a decision procedure for csLTL formulas. csLTL is a linear temporal logic that replaces propositional formulas by constraint formulas, thus in order to determine the validity of a formula with no temporal constructs, it uses the entailment relation of the underlying constraint system.

This decision procedure is the last step of a method to validate LTL formulas for *tccp* programs. It is an adaptation of the tableau defined in [11,13]. The main differences of our algorithm w.r.t. the propositional case are due to the constraint nature of the behavior of *tccp*.

A Constraint Linear Temporal Logic is defined in [18] 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. . Moreover, the verification results are given for the locally-independent fragment of *ntcc*, which avoids the non-monotonicity of the original language. In contrast, our abstract diagnosis technique checks temporal properties for the full *tccp* language.

As future work, we plan to implement this algorithm and integrate it with the verification method. We also plan to explore other instances of the method based on logics for which decision procedures or (semi)automatic tools exists.

A Proofs and results of Section 5

In the section we present the proofs of the results presented in Section 5 together with some auxiliary definitions and results which are used in those proofs.

In the following we recall Lemma 1 and Lemma 2 and we prove them.

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 (of Lemma 1). We prove the three points separately.

1. Let us consider the rules for α-formulas in Fig. 2. 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 $\alpha = \phi_1 \mathbin{\dot\wedge} \phi_2$, this case follows directly from Definition 2, in particular Equation (3.2f).
2. Let us consider the rules for β-formulas in Fig. 2. Let Φ be a set of formulas, β a β-formula and $\phi_1, \phi_2 \in \mathsf{csLTL}$.
   - **R3** Let $\beta = \dot\neg(\phi_1 \mathbin{\dot\wedge} \phi_2)$. We show the two directions independently.
     - $\Rightarrow$ Assume that it exists $r \in \mathbf{M}$ such that $r \vDash \Phi \cup \{\dot\neg(\phi_1 \mathbin{\dot\wedge} \phi_2)\}$. By applying De Morgan laws $r \vDash \Phi \cup \{\dot\neg\,\phi_1 \mathbin{\dot\vee} \dot\neg\,\phi_2\}$. By Definition 2 it follows directly that $r \vDash \Phi \cup \{\dot\neg\,\phi_1\}$ or $r \vDash \Phi \cup \{\dot\neg\,\phi_2\}$.
     - $\Leftarrow$ 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 $r \vDash \Phi \cup \{\dot\neg\,\phi_1 \mathbin{\dot\vee} \dot\neg\,\phi_2\}$ and by De Morgan laws $r \vDash \Phi \cup \{\dot\neg(\phi_1 \mathbin{\dot\wedge} \phi_2)\}$.
   - **R4** Let $\beta = \dot\neg(\phi_1 \,\mathcal{U}\, \phi_2)$.
     - $\Rightarrow$ 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 $\mathcal{U}$, $r \vDash \Phi \cup \{\phi_1, \dot\neg(\phi_2 \mathbin{\dot\vee} \bigcirc(\phi_1 \,\mathcal{U}\, \phi_2))\}$. It can be notice that $\dot\neg(\phi_2 \mathbin{\dot\vee} \bigcirc(\phi_1 \,\mathcal{U}\, \phi_2)) = \dot\neg\,\phi_2 \mathbin{\dot\wedge} \dot\neg \bigcirc(\phi_1 \,\mathcal{U}\, \phi_2)$ 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 $r \vDash \Phi \cup \{\dot\neg\,\phi_1, \dot\neg\,\phi_2\}$.

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

R5 Let $\beta = \phi_1 \mathcal{U} \phi_2$.

- $\Rightarrow$ Assume that it exists $r \in \mathbf{M}$ such that $r \models \Phi \cup \{\phi_1 \mathcal{U} \phi_2\}$. We build a model for at least one of the following sets: $\Phi \cup \{\phi_2\}$ and $\Phi \cup \{\phi_1, \dot{\neg}\phi_2, \bigcirc(\phi_1 \mathcal{U} \phi_2)\}$.
  Assume that $r \models \phi_2$, it follows immediately that $r \models \Phi \cup \{\phi_2\}$.
  Otherwise, if $r \not\models \phi_2$ we have $r \models \Phi \cup \{\dot{\neg}\phi_2, \phi_1 \mathcal{U} \phi_2\}$ and by the fixpoint characterization of $\mathcal{U}$, $a \models \Phi \cup \{\phi_1, \dot{\neg}\phi_2, \bigcirc(\phi_1 \mathcal{U} \phi_2)\}$.
- $\Leftarrow$ We need to distinguish two cases. Assume that it exists $r \in \mathbf{M}$ such that $r \models \Phi \cup \{\phi_2\}$, it follows directly that r is also a model for $\phi_1 \mathcal{U} \phi_2$.
  Now assume that it exists $r \in \mathbf{M}$ such that $r \models \Phi \cup \{\phi_1, \dot{\neg}\phi_2, \bigcirc(\phi_1 \mathcal{U} \phi_2)\}$, by the fixpoint characterization of $\mathcal{U}$ it follows that $r \models \Phi \cup \{\phi_1 \mathcal{U} \phi_2\}$.

R6 Let $\beta = \phi_1 \mathcal{U} \phi_2$ be an eventuality in the context Φ.

- $\Rightarrow$ Assume that it exists $r \in \mathbf{M}$ such that $r \models \Phi \cup \{\phi_1 \mathcal{U} \phi_2\}$, 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 $r^j \models \phi_2$. If $j = 0$ then $r \models \phi_2$ and $r \models \Phi \cup \{\phi_2\}$. Otherwise, if $j > 0$, then $r \not\models \phi_2$ and, by definition of $\mathcal{U}$, $r \models \phi_1$. Let i be the greatest index such that $0 \leq i < l$ and $r^i \models \Phi \cup \{\phi_1 \mathcal{U} \phi_2\}$. 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 $r^{i+1} \models \phi_1 \mathcal{U} \phi_2$, thus, at least one $\phi \in \Phi$ should not be modeled by r^{i+1}. It follows that $r^i \models \bigcirc((\Phi^* \dot{\wedge} \phi_1) \mathcal{U} \phi_2)$.
- $\Leftarrow$ We have to distinguish two cases. Assume that it exists $r \in \mathbf{M}$ such that $r \models \Phi \cup \{\phi_2\}$, thus, $r \models \Phi \cup \{\phi_1 \mathcal{U} \phi_2\}$.
  Otherwise assume that it exists $r \in \mathbf{M}$ such that $r \models \Phi \cup \{\phi_1, \dot{\neg}\phi_2, \bigcirc((\Phi^* \dot{\wedge} \phi_1) \mathcal{U} \phi_2)\}$. Since $r \models \bigcirc((\Phi^* \dot{\wedge} \phi_1) \mathcal{U} \phi_2)$ we have that $r \models \bigcirc(\phi_1 \mathcal{U} \phi_2)$. Thus, by definition of $\mathcal{U}$ we conclude that $r \models \Phi \cup \{\phi_1 \mathcal{U} \phi_2\}$.

3. Consider the set $\Phi = \{c_1, \ldots, c_n, \bigcirc\phi_1, \ldots, \bigcirc\phi_m, \dot{\neg}\bigcirc\psi_1, \ldots, \dot{\neg}\bigcirc\psi_k\}$, 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 $r \models \Phi$. Let us recall that r^1 is suffix of r obtained by delete the first element of r. By Definition 2 it follows that $r \models c_i$ for $i = 1 \ldots n$, $r \models \bigcirc\phi_j$ for $j = 1 \ldots m$ and $r \not\models \bigcirc\psi_l$ for $l = 1 \ldots k$. From monotonicity of r it follows that $r^1 \models c_i$ for $i = 1 \ldots n$. Moreover, by (3.2h), $r^1 \models \phi_j$ for $j = 1 \ldots m$ and $r^1 \not\models \psi_l$ for $l = 1 \ldots k$. Thus it follows directly that $r^1 \models \mathsf{next}(\Phi)$.
   - $\Leftarrow$ Now assume that it exists $r \in \mathbf{M}$ such that $r \models \mathsf{next}(\Phi)$. 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 $r(1) \not\models c$ and $r \not\models \mathsf{next}(\Phi)$. We

show that $(C, \varnothing) \rightarrowtail C \cdot r$ is a model for Φ. By definition of C, is easy to notice that $(C, \varnothing) \rightarrowtail C \vDash c_i$ for $i = 1 \dots n$. Furthermore, by (3.2h), $(C, \varnothing) \rightarrowtail C \cdot r \vDash \bigcirc \phi_j$ for $j = 1 \dots m$ and $r \nvDash \psi_l$ for $l = 1 \dots k$, thus $(C, \varnothing) \rightarrowtail C \cdot r \nvDash \dot{\neg} \bigcirc \psi_l$. Therefore, $(C, \varnothing) \rightarrowtail C \cdot r \vDash \Phi$.

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

Proof (of Lemma 2). We show the two directions independently.

- ⇒ This direction follows directly from (3.2g).

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

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

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

Definition 9. *Let $b = n_0, n_1, \dots n_k$ be an open branch such that $L(n_k) = L(n_j)$ for $0 \le j < k$, 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 operator next.

Definition 10. *Given a branch b, every maximal subsequence $n_i, n_{i+1}, \dots n_j$ of* path(b) *is called a* stage *if, for all $i \le l \le j$, $L(n_l)$ is not formed only by elementary formulas or $L(n_l) \neq$* next$(L(n_{l-1}))$*. We denote by* stages(b) *the sequence of the stages in b.*

We distinguish a particular class of stages called saturated.

Definition 11. *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 $\phi' \in$ csLTL then $\phi' \in L(s)$.*

Definition 12. *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 $0 \le i \le n$ is said to be* fulfilled *in S if there exists $j \ge i$ 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 13. *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* path(stages(b)) *is fulfilling.*

Now we give the definition of expanded branch. As we will see in Section 5.4, open expanded branches correspond to models of the initial set of formulas.

Definition 14. *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 of Subsection 5.1. When all branches are closed or expanded the tableau cannot be further expanded.

Proposition 1. *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 6 it can be noticed that the algorithm applies any possible α-, β-rule and $\dot{\exists}$ elimination before applying the operator next to jump to the following stage.

It can be proved that starting from a finite set of formulas Φ, the set of formulas which can occur in the construction of the systematic tableau $\mathcal{T}_\Phi$ is finite. This result is the adaptation to the csLTL case of the corresponding result for PLTL shown in [13].

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

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

We denote as $subf(\Phi)$ the set of subformulas in Φ and their negations. $preclo(\Phi)$ extends $subf(\Phi)$ with the formulas that can be generated from $subf(\Phi)$ by means of the rules in Subsection 5.1 (α, β rules and $\dot{\exists}$ elimination) except Rule R6.

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

$clo(\Phi)$ captures the formulas generated by Rule R6 by using $negctx(\Phi)$ which represents the conjunctions of negated contexts introduced by Rule R6..

$$\begin{aligned} clo(\Phi) :=& \big\{\dot{\bigwedge}\,\Delta \mid \Delta \subseteq \{\phi_1 \mid \phi_1\,\mathcal{U}\,\phi_2 \in subf(\Phi)\} \cup negctx(\Phi)\big\} \\ &\text{where } \; negctx(\Phi) := \{\Gamma^* \mid \Gamma \subseteq preclo(\Phi)\} \end{aligned}$$

Definition 15. *Let Φ be a set of formulas, the closure of Φ is defined as*

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

Proposition 2. *Let $\Phi \subseteq$ csLTL be a finite set, then $clo(\Phi)$ is also finite.*

Proof. It follows directly from Definition 15.

The fact that $clo(\Phi)$ 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 [13] about the handle of eventualities in the construction of the systematic tableau $\mathcal{T}_\Phi$ for a set of formulas Φ.

Proposition 3. *Let s be a stage in a branch b of $\mathcal{T}_\Phi$, 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. 2 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 4. *Let b be a branch of $\mathcal{T}_\Phi$ and $s_0, s_1, \ldots, s_k$ be a prefix of* $\mathsf{path}(\mathsf{stages}(b))$. *If $\phi_1 \,\mathcal{U}\, \phi_2 \in L(n_i)$ for some $0 \leq i \leq k$, $\phi_1 \,\mathcal{U}\, \phi_2$ is not distinguished in $s_i, \ldots, s_k$ and $\phi_2 \notin L(s_i) \cup \cdots \cup L(s_k)$, then $\{\phi_1, \dot{\neg}\phi_2, \bigcirc(\phi_1 \,\mathcal{U}\, \phi_2)\} \subseteq L(s_j)$ for all $i \leq j \leq k$.*

Proof. By the construction of $\mathcal{T}_\Phi$ since undistinguished eventualities are handled by Rule R5.

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 R6, is in contradiction with the context.

Proposition 5. *Let b be a branch of $\mathcal{T}_\Phi$ and $s_0, s_1, \ldots, s_k$ 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 $\phi_2 \notin L(s_i) \cup \cdots \cup L(s_k)$ then, for all $0 \leq l \leq k - i$, $\{\delta_l, \dot{\neg}\phi_2, \bigcirc(\delta_{l+1} \,\mathcal{U}\, \phi_2)\} \subseteq L(s_{i+l})$ where $\delta_0 = \phi_1$ and $\delta_{l+1} = \delta_l \dot{\wedge} \chi$ 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 $s_0, \ldots, s_{i+l}$ is closed.

Proof. By construction of $\mathcal{T}_\Phi$, distinguished eventualities are handled by Rule R6. 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 R6, $\gamma_0 = \phi_1$ and for all $j > 0$ $\gamma_j = \gamma_{j-1} \dot{\wedge} \Delta^*_{j-1}$ 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 $0 \leq l \leq k - 1$.

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 $s_0, \ldots, s_{i+l}$ contains at the same stage two complementary formulas $\{\psi, \dot{\neg}\psi\}$. From Proposition 3 we can conclude every maximal branch prefixed by $s_0, \ldots, s_{i+l}$ is closed.

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

Proof. By Proposition 5 if a distinguish eventuality in a branch b is unfulfilled, then b is closed and it is not cyclic.

Proposition 6. *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 2 and Point 3 in the algorithm of Definition 6 and from Proposition 3 and Corollary 2.

Let us recall Lemma 3.

Lemma 8. *The algorithm of Definition 6 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 (of Lemma 3). Suppose that the algorithm does not terminates. This means that $\mathcal{T}_\Phi$ contains an infinite branch $b = n_1, n_2, \ldots, n_i \ldots$. By Propositions 2, 5 and 6 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 refutational completeness

Proposition 7. *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 $\pi = s_j, s_{j+1}, \ldots, s_k$ 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 $j \le i \le k$.*
2. *There exists $j \le i \le k$ 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 $j \le l \le i-1$.*

Proof. We proceed by induction of $k-j$. In case $k = j$ the property follows directly from Rule R4 and since each stage of a systematic tableau is saturated (Proposition 1). In case $k > j$, by inductive hypothesis we have that $\pi' = s_j, \ldots, s_{k-1}$ satisfies one of the two properties of Proposition 7. If π' satisfies Point 1 then by the saturation of the stage (Proposition 1) 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.

In the following we show the proof of Lemma 5.

Proof (of Lemma 5). We show the two directions independently.

- $\Rightarrow$ We proceed by induction on the structure of ϕ. Assume that ϕ is satisfiable, then it exists $r \in \mathbf{M}$ such that $r \vDash \phi$. The only non direct case is when $\phi = (S = [c \mid S'])$. We have to distinguish two cases:
  - $\boldsymbol{r = (\eta^+, \eta^-) \rightarrowtail c \cdot r'}$ By (3.2c) it follows that $\eta^+ \vdash (S = [c \mid S'])$ and, as a consequence, $\eta^+ \vdash (S \doteq c)$. The thesis follows directly by noticing that $\sigma(S = [c \mid S']) = S \doteq c$.
  - $\boldsymbol{r = stutt(\eta^-) \cdot r'}$ From (3.2d) it follows that for all $d^- \in \eta^-$ $(S = [c \mid S']) \nvdash d^-$ and $r' \vDash (S = [c \mid S'])$. As a consequence, for all $d^- \in \eta^-$ $(S \doteq c) \nvdash d^-$. The thesis follows directly by noticing that $\sigma(S = [c \mid S']) = S \doteq c$.
- $\Leftarrow$ We proceed by induction on the structure of ϕ. As before the only interesting case is when $\phi = (S = [c \mid S'])$. Assume that $\phi = (S = [c \mid S'])$ and $\sigma(\phi)$ is satisfiable. Thus, it exists $r \in \mathbf{M}$ such that $r \vDash \sigma(\phi)$.
  - $\boldsymbol{r = (\eta^+, \eta^-) \rightarrowtail c \cdot r'}$ From (3.2c) it follows that $\eta^+ \vdash (S \doteq c)$. Consider $\bar{r} = (\eta^+ \otimes (S = [c \mid S']), \eta^-) \rightarrowtail c \cdot r'\downarrow_{(S=[c|S'])}$, it can be noticed that $(\eta^+ \otimes (S = [c \mid S']), \eta^-) \rightarrowtail c \vdash (S = [c \mid S'])$, thus $\bar{r} \vDash \phi$ and ϕ is satisfiable.
  - $\boldsymbol{r = stutt(\eta^-) \cdot r'}$ By (3.2d) it follows that for all $d^- \in \eta^-$ $(S \doteq c) \nvdash d^-$. Consider $\bar{r} = stutt(\eta^-) \cdot r'\downarrow_{(S=[c|S'])}$, it can be noticed that for all $d^- \in \eta^-$ $(S = [c \mid S']) \nvdash d^-$, thus $\bar{r} \vDash \phi$ and ϕ is satisfiable.

B Proofs and results of Section 4

In this section we presents some results about our abstract semantics which are necessary to show Theorem 1.

The following function is the core definition for the correct abstract semantics for *tccp* in the domain of csLTL formulas. Actually, this version of the semantics is an instance of the general framework in which we are restricted to a decidable subset of the csLTL logic. For this reason, the semantics for the choice agent is a correct, but not precise, semantics of the agent's behavior.

Definition 16 (csLTL abstract Semantics).
Given $A \in \mathbb{A}^{\Pi}_{\mathbf{C}}$ *and* $\mathcal{I} \in \mathbb{I}_{\mathbb{F}}$*, we define the* csLTL *semantics evaluation* $\mathcal{A}[\![A]\!]_{\mathcal{I}}$ *by structural induction as follows.*

$$\mathcal{A}[\![\mathsf{skip}]\!]_{\mathcal{I}} \coloneqq \mathit{true} \tag{B.1a}$$

$$\mathcal{A}[\![\mathsf{tell}(c)]\!]_{\mathcal{I}} \coloneqq \bigcirc c \tag{B.1b}$$

$$\mathcal{A}[\![\sum_{i=1}^{n} \mathsf{ask}(c_i) \to A_i]\!]_{\mathcal{I}} \coloneqq \dot{\bigvee}_{i=1}^{n} (c_i \mathbin{\dot\wedge} \bigcirc \mathcal{A}[\![A_i]\!]_{\mathcal{I}}) \mathbin{\dot\vee} (\dot{\bigwedge}_{i=1}^{n} \dot\neg c_i) \tag{B.1c}$$

$$\mathcal{A}[\![\mathsf{now}\ c\ \mathsf{then}\ A_1\ \mathsf{else}\ A_2]\!]_{\mathcal{I}} \coloneqq (c \mathbin{\dot\wedge} \mathcal{A}[\![A_1]\!]_{\mathcal{I}}) \mathbin{\dot\vee} (\dot\neg c \mathbin{\dot\wedge} \mathcal{A}[\![A_2]\!]_{\mathcal{I}}) \tag{B.1d}$$

$$\mathcal{A}[\![A_1 \parallel A_2]\!]_{\mathcal{I}} \coloneqq \mathcal{A}[\![A_1]\!]_{\mathcal{I}} \mathbin{\dot\wedge} \mathcal{A}[\![A_2]\!]_{\mathcal{I}} \tag{B.1e}$$

$$\mathcal{A}[\![\exists x\, A]\!]_{\mathcal{I}} \coloneqq \dot\exists_x \mathcal{A}[\![A]\!]_{\mathcal{I}} \tag{B.1f}$$

$$\mathcal{A}[\![p(\vec{x})]\!]_{\mathcal{I}} \coloneqq \bigcirc \mathcal{I}(p(\vec{x})) \tag{B.1g}$$

Let $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$. We define the (monotonic) immediate consequence operator $\mathcal{D}[\![D]\!]: \mathbb{I}_{\mathbb{F}} \to \mathbb{I}_{\mathbb{F}}$ as

$$\mathcal{D}[\![D]\!]_{\mathcal{I}}(p(\vec{x})) \coloneqq \dot{\bigvee} \{\mathcal{A}[\![A]\!]_{\mathcal{I}} \mid p(\vec{x}) \text{ :- } A \in D\}$$

Lemma 9. *The function $\gamma^{\mathbb{F}}$ is monotonic and injective.*

Proof. $\gamma^{\mathbb{F}}$ **is monotonic.** Let $\phi_1, \phi_2 \in \mathbb{F}$ such that $\phi_1 \dot{\to} \phi_2$. 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, $\phi_1 \dot{\leftrightarrow} \phi_2$.

Lemma 10 (Correctness of $\dot{\bigwedge}$).
Given $\phi_1, \phi_2 \in \mathbb{F}$, $\gamma^{\mathbb{F}}(\phi_1 \dot{\wedge} \phi_2) \sqsupseteq \bigsqcup\{r_1 \mathbin{\bar{\Vert}} r_2 \mid r_1 \in \gamma^{\mathbb{F}}(\phi_1),\ r_2 \in \gamma^{\mathbb{F}}(\phi_2),\ r_1 \mathbin{\bar{\Vert}} r_2$ is defined$\}$.

Proof. Consider $r_1 \in \gamma^{\mathbb{F}}(\phi_1)$ and $r_2 \in \gamma^{\mathbb{F}}(\phi_2)$ such that $r_1 \mathbin{\bar{\Vert}} r_2$ is defined. We show that $r_1 \mathbin{\bar{\Vert}} r_2 \in \gamma^{\mathbb{F}}(\phi_1 \dot{\wedge} \phi_2)$ (i.e., $r_1 \mathbin{\bar{\Vert}} r_2 \vDash \phi_1 \dot{\wedge} \phi_2$). Since $r_1 \mathbin{\bar{\Vert}} r_2$ 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 $(r_1 \mathbin{\bar{\Vert}} r_2) \vDash \phi_1$. Following a similar reasoning, we have that $(r_1 \mathbin{\bar{\Vert}} r_2) \vDash \phi_2$ and finally, from Equation (3.2f) we can conclude that $(r_1 \mathbin{\bar{\Vert}} r_2) \vDash \phi_1 \dot{\wedge} \phi_2$.

Lemma 11 (Correctness of $\dot{\exists}_x$).
Given $\phi \in \mathbb{F}$, $\gamma^{\mathbb{F}}(\dot{\exists}_x\, \phi) \sqsupseteq \bigsqcup\{\tilde{\exists}_x\, r \mid r \in \gamma^{\mathbb{F}}(\phi),\ r$ 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 (3.2g) it follows directly that $\tilde{\exists}_x\, r \vDash \dot{\exists}_x\, \phi$. By Equation (3.1) we can conclude that $\tilde{\exists}_x\, r \in \gamma^{\mathbb{F}}(\dot{\exists}_x\, \phi)$.

Theorem 5 (Correctness of $\mathcal{A}$). *Let $A \in \mathbb{A}^{\Pi}_{\mathbf{C}}$ and $\mathcal{I} \in \mathbb{I}_{\mathbb{F}}$. Then, $\mathcal{A}[\![A]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})} \sqsubseteq \gamma^{\mathbb{F}}(\mathcal{A}[\![A]\!]_{\mathcal{I}})$.*

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

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

$$\begin{aligned}
\mathcal{A}[\![\mathsf{skip}]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})} &= \{(\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots\} \\
&\qquad [\text{by Definition 2 since } \boxtimes \vDash \mathit{true}\,] \\
&\sqsubseteq \{r \mid r \vDash \mathit{true}\} \\
&\qquad [\text{by Equation (3.1)}\,] \\
&= \gamma^{\mathbb{F}}(\mathit{true}) \\
&\qquad [\text{by Equation (B.1a)}\,] \\
&= \gamma^{\mathbb{F}}(\mathcal{A}[\![\mathsf{skip}]\!]_{\mathcal{I}})
\end{aligned}$$

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

$$
\begin{aligned}
\mathcal{A}[\![\mathsf{tell}(c)]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})} &= \{(\mathit{true},\varnothing) \rightarrowtail c \cdot (c,\varnothing) \rightarrowtail c \cdots (c,\varnothing) \rightarrowtail c \cdots\} \\
&\qquad [\text{by Definition } 2] \\
&\sqsubseteq \{r \mid r \vDash \bigcirc c\} \\
&\qquad [\text{by Equation } (3.1)] \\
&= \gamma^{\mathbb{F}}(\bigcirc c) \\
&\qquad [\text{by Equation } (\text{B.1b})] \\
&= \gamma^{\mathbb{F}}(\mathcal{A}[\![\mathsf{tell}(c)]\!]_{\mathcal{I}})
\end{aligned}
$$

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

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

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

1. Let $r := (c,\varnothing) \rightarrowtail c \cdots (c,\varnothing) \rightarrowtail c \cdots$ such that $(\mathit{true},\varnothing) \rightarrowtail \mathit{true} \cdots (\mathit{true},\varnothing) \rightarrowtail \mathit{true} \cdots \in \mathcal{A}[\![A_1]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})}$. From (3.2c) it follows that $r \vDash c$. By inductive hypothesis, $(\mathit{true},\varnothing) \rightarrowtail \mathit{true} \cdots (\mathit{true},\varnothing) \rightarrowtail \mathit{true} \cdots \in \gamma^{\mathbb{F}}(\mathcal{A}[\![A_1]\!]_{\mathcal{I}})$. 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 $\mathit{true} \mathbin{\dot\rightarrow} \mathcal{A}[\![A_1]\!]_{\mathcal{I}}$. Since $\forall \phi \in \mathsf{csLTL}.\ \phi \mathbin{\dot\rightarrow} \mathit{true} \mathbin{\dot\wedge} \phi$, it holds that $r \vDash c \mathbin{\dot\wedge} \mathcal{A}[\![A_1]\!]_{\mathcal{I}}$.
2. Let $r := (\eta^+ \otimes c, \eta^-) \rightarrowtail d \otimes c \cdot (r'{\downarrow_c})$ such that $(\eta^+,\eta^-) \rightarrowtail d \cdot r' \in \mathcal{A}[\![A_1]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})}$, $d \otimes c \neq \mathit{false}$, $\forall c^- \in \eta^-.\ \eta^+ \otimes c \nvDash c^-$ and r' is c-compatible. By (3.2c), it follows that $r \vDash c$. By inductive hypothesis, we know that $(\eta^+,\eta^-) \rightarrowtail d \cdot r' \in \gamma^{\mathbb{F}}(\mathcal{A}[\![A_1]\!]_{\mathcal{I}})$, and by (3.1), $(\eta^+,\eta^-) \rightarrowtail d \cdot r' \vDash \mathcal{A}[\![A_1]\!]_{\mathcal{I}}$. By hypothesis, $(\eta^+,\eta^-) \rightarrowtail d \cdot r'$ is compatible with c, thus $\mathcal{A}[\![A_1]\!]_{\mathcal{I}}$ 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 $r \vDash \mathcal{A}[\![A_1]\!]_{\mathcal{I}}$. By (3.2f) we conclude that $r \vDash c \mathbin{\dot\wedge} \mathcal{A}[\![A_1]\!]_{\mathcal{I}}$.

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 $(\eta^+, \eta^-) \rightarrowtail d \cdot r' \in \mathcal{A}[\![A_1]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})}$, $d \otimes c = \mathit{false}$, $\forall c^- \in \eta^-.\ \eta^+ \otimes c \nvdash c^-$ and r' is c-compatible. By (3.2c), it follows that $r \vDash c$. By inductive hypothesis, $(\eta^+, \eta^-) \rightarrowtail d \cdot r' \in \gamma^{\mathbb{F}}(\mathcal{A}[\![A_1]\!]_{\mathcal{I}})$ and, by (3.1), $(\eta^+, \eta^-) \rightarrowtail d \cdot r' \vDash \mathcal{A}[\![A_1]\!]_{\mathcal{I}}$. Reasoning similarly to Point 2 above, it can be noticed that $r \vDash \mathcal{A}[\![A_1]\!]_{\mathcal{I}}$. Thus, by (3.2f) we conclude that $r \vDash c \mathbin{\dot\wedge} \mathcal{A}[\![A_1]\!]_{\mathcal{I}}$.
4. Let $r \coloneqq (c, \eta^-) \rightarrowtail c \cdot (r' {\downarrow_c})$ such that $\mathit{stutt}(\eta^-) \cdot r' \in \mathcal{A}[\![A_1]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})}$, $\forall c^- \in \eta^-.\ \eta^+ \otimes c \nvdash c^-$ and r' is c-compatible. It follows from (3.2c) that $r \vDash c$. By inductive hypothesis, $\mathit{stutt}(\eta^-) \cdot r' \in \gamma^{\mathbb{F}}(\mathcal{A}[\![A_1]\!]_{\mathcal{I}})$, and, by (3.1), $\mathit{stutt}(\eta^-) \cdot r' \vDash \mathcal{A}[\![A_1]\!]_{\mathcal{I}}$. Reasoning as in Point 2 of this proof it follows that $r \vDash \mathcal{A}[\![A_1]\!]_{\mathcal{I}}$. Thus, $r \vDash c \mathbin{\dot\wedge} \mathcal{A}[\![A_1]\!]_{\mathcal{I}}$.
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 $(\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots \in \mathcal{A}[\![A_2]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})}$. By (3.2c) and (3.2e), $r \vDash \dot\neg c$. By inductive hypothesis, $(\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots (\mathit{true}, \varnothing) \rightarrowtail \mathit{true} \cdots \in \gamma^{\mathbb{F}}(\mathcal{A}[\![A_2]\!]_{\mathcal{I}})$ and, reasoning as in Point 1 of this proof, it follows that $\mathit{true} \mathrel{\dot\rightarrow} \mathcal{A}[\![A_2]\!]_{\mathcal{I}}$. Since $\forall \phi \in \mathsf{csLTL}.\ \phi \mathrel{\dot\rightarrow} \mathit{true} \mathbin{\dot\wedge} \phi$, it holds that $r \vDash \dot\neg c \mathbin{\dot\wedge} \mathcal{A}[\![A_2]\!]_{\mathcal{I}}$.
6. Let $r \coloneqq (\eta^+, \eta^- \cup \{c\}) \rightarrowtail d \cdot r'$ such that $(\eta^+, \eta^-) \rightarrowtail d \cdot r' \in \mathcal{A}[\![A_2]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})}$ and $\eta^+ \nvdash c$. By (3.2c), $r \vDash \dot\neg c$. By inductive hypothesis, $(\eta^+, \eta^-) \rightarrowtail d \cdot r' \in \gamma^{\mathbb{F}}(\mathcal{A}[\![A_2]\!]_{\mathcal{I}})$ and, by (3.1), $(\eta^+, \eta^-) \rightarrowtail d \cdot r' \vDash \mathcal{A}[\![A_2]\!]_{\mathcal{I}}$. It can be noticed that $\mathcal{A}[\![A_2]\!]_{\mathcal{I}}$ cannot imply the formula c, otherwise $(\eta^+, \eta^-) \rightarrowtail d \cdot r'$ would not be a model for $\mathcal{A}[\![A_2]\!]_{\mathcal{I}}$ 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 $r \vDash \mathcal{A}[\![A_2]\!]_{\mathcal{I}}$. Thus, by (3.2f) we conclude that $r \vDash \dot\neg c \mathbin{\dot\wedge} \mathcal{A}[\![A_2]\!]_{\mathcal{I}}$.
7. Let $r \coloneqq (\mathit{true}, \eta^- \cup \{c\}) \rightarrowtail \mathit{true} \cdot r'$ such that $\mathit{stutt}(\eta^-) \cdot r' \in \mathcal{A}[\![A_2]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})}$. By (3.2d), $r \vDash \dot\neg c$ and, by inductive hypothesis, $\mathit{stutt}(\eta^-) \cdot r' \vDash \mathcal{A}[\![A_2]\!]_{\mathcal{I}}$. Reasoning as in the previous Point 6 it can be noticed that $r \vDash \mathcal{A}[\![A_2]\!]_{\mathcal{I}}$ and, therefore, $r \vDash \dot\neg c \mathbin{\dot\wedge} \mathcal{A}[\![A_2]\!]_{\mathcal{I}}$.

We have proven that for all $r \in \mathcal{A}[\![A]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})}$ either $r \vDash c \mathbin{\dot\wedge} \mathcal{A}[\![A_1]\!]_{\mathcal{I}}$ or $r \vDash \dot\neg c \mathbin{\dot\wedge} \mathcal{A}[\![A_2]\!]_{\mathcal{I}}$. Therefore, from (B.1d) we conclude that $r \vDash \mathcal{A}[\![A]\!]_{\mathcal{I}}$.

Case $A = A_1 \parallel A_2$**.** Let $r_1 \mathbin{\tilde\parallel} r_2 \in \mathcal{A}[\![A_1 \parallel A_2]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})}$ such that $r_1 \in \mathcal{A}[\![A_1]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})}$ and $r_2 \in \mathcal{A}[\![A_2]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})}$. By inductive hypothesis, $r_1 \in \gamma^{\mathbb{F}}(\mathcal{A}[\![A_1]\!]_{\mathcal{I}})$ and $r_2 \in \gamma^{\mathbb{F}}(\mathcal{A}[\![A_2]\!]_{\mathcal{I}})$. It follows from Lemma 10 that $r_1 \mathbin{\tilde\parallel} r_2 \in \gamma^{\mathbb{F}}(\mathcal{A}[\![A_1 \parallel A_2]\!]_{\mathcal{I}})$.

Case $A = \exists x\, A_1$**.** Let $\tilde\exists_x r_1 \in \mathcal{A}[\![\exists x\, A_1]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})}$ such that $r_1 \in \mathcal{A}[\![A_1]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})}$ and r_1 is x-closed. By inductive hypothesis, $r_1 \in \gamma^{\mathbb{F}}(\mathcal{A}[\![A_1]\!]_{\mathcal{I}})$ and, by Lemma 11, it follows that $\tilde\exists_x r_1 \in \gamma^{\mathbb{F}}(\mathcal{A}[\![\exists x\, A_1]\!]_{\mathcal{I}})$.

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

Lemma 12. *For each $A \in \mathbb{A}^{\Pi}_{\mathbf{C}}$ and each $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$, $\mathcal{A}[\![A]\!]$ and $\mathcal{D}[\![D]\!]$ are monotonic.*

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

$\mathcal{I}_2(p(\vec{x}))$, thus:

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

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

Theorem 6 (Correctness of $\mathcal{D}$). *Let $D \in \mathbb{D}^{\Pi}_{\mathbf{C}}$ and $\mathcal{I} \in \mathbb{I}_{\mathbb{F}}$. Then, $\mathcal{D}[\![D]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})} \sqsubseteq \gamma^{\mathbb{F}}(\mathcal{D}[\![D]\!]_{\mathcal{I}})$.*

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

$$\begin{aligned}
\mathcal{D}[\![D]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})}(p(\vec{x})) &= \bigsqcup_{p(x):-A\in D} \mathcal{A}[\![A]\!]_{\gamma^{\mathbb{F}}(\mathcal{I})} \\
&\quad [\,\text{by Theorem 5}\,] \\
&\sqsubseteq \bigsqcup_{p(x):-A\in D} \gamma^{\mathbb{F}}(\mathcal{A}[\![A]\!]_{\mathcal{I}}) \\
&\quad [\,\text{by the monotonicity of } \gamma^{\mathbb{F}} \text{ (Lemma 9)}\,] \\
&\sqsubseteq \gamma^{\mathbb{F}}(\dot{\bigvee}_{p(x):-A\in D} \mathcal{A}[\![A]\!]_{\mathcal{I}}) \\
&= \gamma^{\mathbb{F}}(\mathcal{D}[\![D]\!]_{\mathcal{I}})(p(\vec{x}))
\end{aligned}$$

Let us recall Theorem 1 and prove it.

Theorem 7. *Consider a set of declarations D and an abstract specification $\mathcal{S}$.*

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

Proof (of Theorem 1).

Point 1 By hypothesis, $\mathcal{D}[\![D]\!]_{\mathcal{S}} \dot{\to} \mathcal{S}$. Since $\gamma^{\mathbb{F}}$ is monotonic (Lemma 9), $\gamma^{\mathbb{F}}(\mathcal{D}[\![D]\!]_{\mathcal{S}}) \sqsubseteq \gamma^{\mathbb{F}}(\mathcal{S})$. By the soundness of $\mathcal{D}$ (Theorem 6) and by transitivity it follows that $\mathcal{D}[\![D]\!]_{\gamma^{\mathbb{F}}(\mathcal{S})} \sqsubseteq \gamma^{\mathbb{F}}(\mathcal{S})$. It can be noticed that $\gamma^{\mathbb{F}}(\mathcal{S})$ is a pre-fixpoint of $\mathcal{D}[\![D]\!]$, thus, from Knaster-Tarski's theorem it follows directly that $\mathit{lfp}\,\mathcal{D}[\![D]\!] \dot{\to} \gamma^{\mathbb{F}}(\mathcal{S})$. 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 $\phi_t \dot{\to} \mathcal{D}[\![D]\!]_{\mathcal{S}}$ and $\phi_t \dot{\wedge} \mathcal{D}[\![D]\!]_{\mathcal{S}} = \dot{\mathit{false}}$. Thus, it follows that $\gamma^{\mathbb{F}}(\mathcal{D}[\![D]\!]_{\mathcal{S}}) \sqcap \gamma^{\mathbb{F}}(\mathcal{S}) = \{\epsilon\}$. Since $\mathcal{D}[\![D]\!]_{\gamma^{\mathbb{F}}(\mathcal{S})} \sqsubseteq \gamma^{\mathbb{F}}(\mathcal{D}[\![D]\!]_{\mathcal{S}})$, we have that $\mathcal{D}[\![D]\!]_{\gamma^{\mathbb{F}}(\mathcal{S})} \sqcap \gamma^{\mathbb{F}}(\mathcal{S}) = \{\epsilon\}$. 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.

References

1. M. Alpuente, M. Comini, S. Escobar, M. Falaschi, and S. Lucas. Abstract Diagnosis of Functional Programs. In M. Leuschel, editor, *Logic Based Program Synthesis and Transformation – 12th International Workshop, LOPSTR 2002, Revised Selected Papers*, volume 2664 of *Lecture Notes in Computer Science*, pages 1–16, Berlin, 2003. Springer-Verlag.
2. G. Bacci and M. Comini. Abstract Diagnosis of First Order Functional Logic Programs. In M. Alpuente, editor, *Logic-based Program Synthesis and Transformation, 20th International Symposium*, volume 6564 of *Lecture Notes in Computer Science*, pages 215–233, Berlin, 2011. Springer-Verlag.
3. 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.
4. M. Comini, G. Levi, M. C. Meo, and G. Vitiello. Abstract Diagnosis. *Journal of Logic Programming*, 39(1-3):43–93, 1999.
5. 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.
6. 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.
7. F. S. de Boer, M. Gabbrielli, and M. C. Meo. A Timed Concurrent Constraint Language. *Information and Computation*, 161(1):45–83, 2000.
8. 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.
9. F. S. de Boer, M. Gabbrielli, and M. C. Meo. Proving correctness of Timed Concurrent Constraint Programs. *CoRR*, cs.LO/0208042, 2002.
10. 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.
11. J. Gaintzarain, M. Hermo, P. Lucio, and M. Navarro. Systematic semantic tableaux for pltl. *Electronic Notes in Theoretical Computer Science*, 206:59–73, 2008.
12. 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.
13. 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.
14. Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems - specification*. Springer, 1992.
15. 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.

16. 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.
17. V. A. Saraswat. *Concurrent Constraint Programming.* The MIT Press, Cambridge, Mass., 1993.
18. F. D. Valencia. Decidability of infinite-state timed ccp processes and first-order ltl. *Theoretical Computer Science*, 330(3):577–607, 2005.