Journal of Universal Computer Science, vol. 12, no. 11 (2006), 1551-1573
submitted: 1/5/06, accepted: 15/10/06, appeared: 28/11/06 © *J.UCS*

Verifying Real–Time Properties of tccp Programs

María Alpuente
(Technical University of Valencia, Spain
alpuente@dsic.upv.es)

María del Mar Gallardo
(University of Malaga, Spain
gallardo@lcc.uma.es)

Ernesto Pimentel
(University of Malaga, Spain
ernesto@lcc.uma.es)

Alicia Villanueva
(Technical University of Valencia, Spain
villanue@dsic.upv.es)

Abstract: The size and complexity of software systems are continuously increasing, which makes them difficult and labor-intensive to develop, test and evolve. Since concurrent systems are particularly hard to verify by hand, achieving effective and automated verification tools for concurrent software has become an important topic of research. *Model checking* is a popular automated verification technology which allows us to determine the properties of a software system and enables more thorough and less costly testing. In this work, we improve the *model–checking* methodology previously developed for the *timed concurrent constraint programming* language tccp so that more sophisticated, real–time properties can be verified by the model–checking tools. The contributions of the paper are twofold. On the one hand, we define a timed extension of the tccp semantics which considers an explicit, discrete notion of the passing of time. On the other hand, we consistently define a real–time extension of the linear–time temporal logic that is used to specify and analyze the software properties in tccp. Both extensions fit into the tccp framework perfectly in such a way that with minor modifications any tccp model checker can be reused to analyze real–time properties. Finally, by means of an example, we illustrate the improved ability to check real–time properties.
Key Words: timed concurrent constraint paradigm, model checking, temporal logic
Category: D.2.4, D.3.2

1 Introduction

The ever–growing size and sophistication of software systems requires more powerful, automated analysis and verification tools that are able to improve the reliability of programs. When considering concurrent software, the problem of verification becomes even more acute, since correctness is more elusive to capture by any but very precise formal tools. Linear temporal logic,

as it is used in model–checking procedures [Clarke, Emerson and Sistla 1993, Manna and Pnueli 1992], has been proven to be very appropriate for the verification of concurrent software. One of its attractive features is the qualitative representation of time, which is based more on the notion of precedence of events than on metric description.

The main problem that model–checking methodologies have to face is the traditional state–explosion problem that makes them inapplicable to large size systems. In the *concurrent constraint* paradigm ccp [Saraswat 1993], the notion of *store as valuation* is substituted by the notion of *store as constraint*, so that very compact state representations are obtained. This makes tccp (the *timed concurrent constraint programming* language of [de Boer, Gabbrielli and Meo 1999] which extends ccp with a discrete notion of time and a suitable mechanism to model timeouts and preemptions) especially appropriate for specifying and analyzing timing properties of concurrent systems by model checking. Two important features of tccp are the monotonicity of the store and the maximal parallelism of processes in execution. In other words, the store expands monotonically through time, and all parallel processes are run concurrently at each time instant.

In order to specify the desired temporal properties of the systems, both the tccp model–checking framework defined in [Falaschi and Villanueva 2006] and the optimized frameworks based on the symbolic and abstract algorithms of [Alpuente *et al.* 2005a] and [Alpuente *et al.* 2004a, Alpuente *et al.* 2005b] rely on the linear temporal logic LTL of [de Boer, Gabbrielli and Meo 2001], which is especially tailored to reason with constraints. The main limitation of this framework is caused by the monotonicity inherent to tccp stores. Roughly speaking, information is incrementally recorded in a disordered bag (the store) so that the relative order in which two pieces of information are stored is unavoidably lost. Moreover, it is also difficult to recognize whether two variables correspond to the same incarnation of recursive procedure calls, unless some ad-hoc "packaging" predicates are explicitly introduced in the tccp code, as in [Alpuente *et al.* 2005b]. As a consequence of these shortages of the previous model–checking framework, it is very hard to deal with quantitative temporal properties regarding the relative precedence among tccp events such as: "from the time instant in which $y = 2$ on, x will always be positive".

In this paper, we improve the existing model–checking technology for tccp so that sophisticated timing properties regarding temporal ordering can be naturally verified. First, we define an extension of the tccp semantics which considers an explicit, discrete notion of time. The main idea is to supply the global store with a suitable structure which allows us to recognize the pieces of information that are added at each time instant. A new notion of constraint entailment is also provided for structured stores so that the resulting computational model is

proven equivalent to the original one.

In order to improve the verification power of our model, we then introduce new temporal operators for the logic LTL that better exploit the structure of stores. In contrast to the *metric* temporal logic of [Alur and Henzinger 1994] that annotates next–state and strong–until operators with nonnegative integer time points, we introduce discrete–time marks to formulae instead, which suffices to model synchronous real–time systems. This also makes it different from [Alur and Henzinger 1994], which was devised to model dense time with discrete clocks.

By means of an example, we also compare the advantages of using tccp specifications w.r.t. a different approach for modeling and verifying real–time systems which is based on *timed automata*. Timed automata [Alur and Dill 1994] are an extension of Büchi automata with real–valued variables that model the clocks that describe the passing of time and are also used to postpone certain transitions until the corresponding deadline has been reached. Several model checkers, such as UPPAAL [Larsen, Petterson and Yi 1997] and Kronos [Yovine 1997], have been developed to analyze timed automata w.r.t. a particular class of real–time temporal logic formulae. As we show, the typical time restrictions of real–time systems can be described in tccp by using the original language sentences. More specifically, clocks can be implemented as time constraints, and then handled as ordinary constraints by the program. Thus, real time is introduced in tccp in a natural manner, which makes feasible extending tccp model checkers to analyze real–time properties in a simple way.

The paper is organized as follows. In Section 2, we briefly introduce the essentials of model–checking methods for tccp, as defined in [Alpuente *et al.* 2004a, Alpuente *et al.* 2005a, Alpuente *et al.* 2005b, Falaschi and Villanueva 2006]. In Section 3, we introduce the notion of structured store and formalize the new operational semantics for tccp augmented with time. In Section 4, we extend the LTL logic of [de Boer, Gabbrielli and Meo 2001] so that the ability to reason about temporal events in structured stores is improved. This is achieved by introducing the notion of *just entailed constraint* as well as a suitable notation for the streams which are used in tccp to record the change of state: each single variable is associated to a stream (implemented as a logical list); that is, each element of the list represents the value of the variable at a given time instant. Section 5 formulates our method to verify real–time properties in tccp. Finally, in Section 6 we conclude. Proofs of all technical results of the paper are given in Appendix A.

2 Model Checking for tccp

Model checking is an automatic technique for verifying finite state concurrent systems [Manna and Pnueli 1992]. It consists of three main tasks:

1. Modeling the system to be analyzed using a *modeling language*. Since the model built represents a non–deterministic concurrent system, its execution will typically explore many different paths, which are usually formalized by using a trace–based operational semantics given to the modeling language.

2. Specifying the desirable properties that the model must fulfill. Temporal logics such as linear temporal logic (LTL) or branching time logic (CTL) are frequently used to express both safety and liveness properties.

3. Running an automatic verification technique to check the correctness of the model w.r.t. a specific temporal property. Model–checking algorithms work by exhaustively inspecting the state space associated to the model, searching for traces that do not satisfy the desirable property.

The main limitation of model checking is known as the *state explosion problem*, which occurs when the model to be verified generates too many states to be recorded by the model–checking tool. Using tccp as a modeling language partially mitigates this problem since tccp permits very compact state representations thanks to the use of constraints.

In [de Boer, Gabbrielli and Meo 1999], the *Timed Concurrent Constraint* language (tccp for short) was defined as an extension of the Concurrent Constraint Programming language ccp [Saraswat 1993]. In the cc paradigm, the notion of *store as valuation* is replaced by the notion of *store as constraint*. The computational model is based on a global store where constraints are accumulated and on a set of agents that interact with the store. The model is parametric w.r.t. a cylindric constraint system $\mathcal{C}$ defined as follows.

Definition 1. Let $\langle \mathcal{C}, \leq, \sqcup, \mathit{true}, \mathit{false} \rangle$ be a complete algebraic lattice where $\sqcup$ is the lub operation, and *true*, *false* are the least and the greatest elements of $\mathcal{C}$, respectively. Assume that *Var* is a denumerable set of variables, and for each $x \in \mathit{Var}$, there exists a function $\exists_x : \mathcal{C} \to \mathcal{C}$ such that, for each $u, v \in \mathcal{C}$:

1. $\exists_x u \leq u$
2. $u \leq v$ then $\exists_x u \leq \exists_x v$
3. $\exists_x(u \sqcup \exists_x v) = \exists_x u \sqcup \exists_x v$
4. $\exists_x(\exists_y u) = \exists_y(\exists_x u)$

Then, $\langle \mathcal{C}, \leq, \sqcup, \mathit{true}, \mathit{false}, \mathit{Var}, \exists \rangle$ is a *cylindric constraint system*.

We will use the entailment relation $\vdash$ instead of its inverse relation $\leq$. Formally, given $u, v \in \mathcal{C}, u \leq v \iff v \vdash u$.

A *set of diagonal elements* for a cylindric constraint system consists of a family of elements $\{\delta_{xy} \in \mathcal{C} | x, y \in \mathit{Var}\}$ such that

1. $\mathit{true} \vdash \delta_{xx}$
2. If $y \neq x, z$ then $\delta_{xz} = \exists_y(\delta_{xy} \sqcup \delta_{yz})$.
3. If $x \neq y$ then $\delta_{xy} \sqcup \exists_x(v \sqcup \delta_{xy}) \vdash v$.

Diagonal elements allow us to hide local variables, as well as to implement parameter passing among predicates. Thus, quantifier $\exists_x$ and diagonal elements δ_{xy} allow us to properly deal with variables in constraint systems.

In tccp, a new conditional agent (now c then A else A) is introduced (w.r.t. ccp) which makes it possible to model behaviors where the absence of information can cause the execution of a specific action. Intuitively, the execution of a tccp program evolves by asking and telling information to the store. Let us briefly recall the syntax of the language:

$$A ::= \mathsf{stop} \mid \mathsf{tell}(c) \mid \textstyle\sum_{i=1}^{n} \mathsf{ask}(c_i) \rightarrow A_i \mid \mathsf{now}\, c\, \mathsf{then}\, A\, \mathsf{else}\, A \mid A||A \mid \exists x\, A \mid \mathsf{p}(x)$$

where c, c_i are *finite constraints* (i.e., atomic propositions) of $\mathcal{C}$. A tccp *process* P is an object of the form $D.A$, where D is a set of procedure declarations of the form $\mathsf{p}(x) :- A$, and A is an agent.

Intuitively, the stop agent finishes the execution of the program and $\mathsf{tell}(c)$ adds the constraint c to the store. Conditions in agent choice are key to implement synchronization in tccp. Agent $\sum_{i=1}^{n} \mathsf{ask}(c_i) \rightarrow A_i$ consults the store and non–deterministically executes the agent A_i in the following time instant, provided the store satisfies the condition c_i; otherwise, if no condition c_i is entailed by the store, the agent *suspends*. The conditional agent (now c then $A1$ else $A2$) can detect *negative information* in the sense that, if the store satisfies c, then the agent $A1$ is executed; otherwise $A2$ is executed. $A1||A2$ executes the two agents $A1$ and $A2$ in parallel. The $\exists x\ A$ agent is used to hide the information regarding x. Finally, $\mathsf{p}(x)$ is the procedure call agent.

The notion of time is introduced by defining a global clock which synchronizes all agents. In the semantics of tccp, the only agents that consume time are the tell, choice and procedure call agents. Note that, since stores grow monotonically, it is not possible to change the value of a given variable. If we want to model the evolution of variable values along the time, we have to deal with *streams*, which allow us to handle imperative variables in the same way as logical lists are used in concurrent logic languages. We write $X = [Y|Z]$ for denoting a stream X recording the current value Y of the considered variable and the stream Z of future values of the same variable. Streams are also used in tccp as explicit communication channels between tccp agents as illustrated by the example given in Section 3.3.

3 Introducing Explicit Time in tccp

In this section, we propose a new computational model for the language. First, we define the new notion of store and entailment relation, and then we extend the original operational semantics of the language to the new formulation.

3.1 The Structured Store

The store used by tccp can be viewed as a blackboard where information is continuously written and never canceled. As we have shown above, the problem is that we add information without keeping track of the insertion order so that we cannot recover the time instant when a constraint has been added, which is essential to analyzing temporal properties.

The following definition provides a structure to the notion of store by using the simple notion of time as a "state counter". Intuitively, a structured store consists of a timed sequence of stores. Each store represents only the information added at a given time instant by the processes that are run concurrently. Thus, we can observe and analyze the evolution of the structured store through time.

Definition 2. We define a *structured store* as an infinite indexed sequence of stores, i.e., an element of the domain $\textsc{Store} = \mathcal{C}^\omega$. We denote the i^{th} component of a structured store st as st_i, and it represents the store at time i.

Now, to work with the new structure, we redefine the notion of entailment relation and the least upper bound (lub) of constraints. Intuitively, the information stored up to a given time instant t is the lub of all the stores st_i in the sequence $0 \leq i \leq t$.

Definition 3. Given a constraint $c \in \mathcal{C}$ and a structured store $st \in \textsc{Store}$, the new entailment relation $\vdash_t$ is defined as

$$st \vdash_t c \Leftrightarrow (\sqcup_{0 \leq i \leq t} st_i) \vdash c$$

We also need to adapt the mechanism for updating the store, since we want to add the information to the right time instant.

Definition 4. Given a structured store st and a constraint $c \in \mathcal{C}$, the addition of c to the store st at the time instant t, $st \sqcup_t c$, is the structured store st', where each component st'_i is defined as $st_t \sqcup c$ if $i = t$ and st_i otherwise.

Intuitively, the updated structured store coincides with the old one in all the components except for component t, where constraint c is added. Moreover, we define the union of structured stores as $(st \sqcup st')_i = st_i \sqcup st'_i \ \forall i \geq 0$.

3.2 Operational Semantics Augmented with Time

Now we instrument the original operational semantics of the language with the new notions of store and constraint entailment given in definitions 2 and 3.

In Figure 1, we show the new transition relation $\longrightarrow \in (A \times \textsc{Store} \times \mathbb{N})^2$ where A is the set of tccp agents given in Section 2, and $\mathbb{N}$ is the domain of

natural numbers.We have augmented the configurations handled by the semantics with a parameter that represents the current time instant. As will become apparent later, the introduction of this parameter is possible because tccp agents are totally synchronized. The idea is that, at each time instant, we introduce the constraint generated by the agents into the right component of the structured store. Specifically, when a tell agent adds a constraint to the store, we update the structured store by introducing that piece of information into the component that corresponds to the subsequent time instant. [1] In the figure, symbol $\not\longrightarrow$ is used to indicate that it is not possible a step using relation $\longrightarrow$, i. e., the corresponding agent suspends.

Given a tccp program P, an agent A_0, and an initial structured store $st^0 = st_0^0 \cdot true^\omega \in \textsc{Store}$, [2] the *timed operational semantics* of P w.r.t. the initial configuration $\langle A_0, st^0\rangle$, is

$$\begin{aligned}\mathcal{O}_T(P)[\langle A_0, st^0\rangle] = \{&st = st_0^0 \cdot st_1^1 \cdot \cdots \in \textsc{Store}| \\ &\langle A_i, st^i\rangle_i \longrightarrow \langle A_{i+1}, st^{i+1}\rangle_{i+1} \text{ for } i \geq 0\}\end{aligned}$$

Thus, for each $st^i \in \textsc{Store}$ incrementally built during the execution, the semantics only records its i^{th} component st_i^i, which corresponds to the constraints added at the time instant i. We assume that each trace in $\mathcal{O}_T(P)[\langle A_0, st^0\rangle]$ is infinite (the last configuration is repeated indefinitely if necessary).

It is immediate for a particular implementation to optimize the representation of structured stores by getting rid of redundant information. However, we prefer to carry on with the proposed structure in our theoretical development in order to keep our formulation simpler.

Recall that the original tccp operational semantics (showed in Appendix A), which we denote as $\mathcal{O}$, produces monotonic sequences of stores, that is, traces of the form $s = s_0 \cdot s_1 \cdots$ where each $s_i \in \mathcal{C}$ and $\forall i \geq 0. s_{i+1} \vdash s_i$. In contrast, the sequences of stores produced by the instrumented operational semantics given in Figure 1 are non monotonic. As commented above, the i^{th} component only contains the set of constraints added at the time instant i.

Let us explain how to transform structured stores into standard stores. Given the structured store $st = st_0 \cdot st_1 \cdots$, then the corresponding monotonic sequence of stores $mon(st)$ is given by the sequence $s_0 \cdot s_1 \cdots s_i \cdots$, where $s_0 = st_0$ and $s_i = \sqcup_{j \leq i} st_j$. On the other hand, a sequence of monotonic stores produced by the standard trace semantics of tccp determines a structured store corresponding to the timed semantics.

The following theorem establishes the soundness and completeness of the new operational semantics with respect to the original one.

[1] Note that, in the original semantics, the information added by the tell agent is also available only in the subsequent time instant.

[2] Note that st_0^0 represents the first component of the structured store st^0.

R1	$\langle \mathsf{tell}(c), st\rangle_t \longrightarrow \langle \mathsf{stop}, st \sqcup_{t+1} c\rangle_{t+1}$	
R2	$\langle \sum_{i=0}^{n} \mathsf{ask}(c_i) \rightarrow A_i, st\rangle_t \longrightarrow \langle A_j, st\rangle_{t+1}$	if $0 \leq j \leq n$ and $st \vdash_t c_j$
R3	$\dfrac{\langle A, st\rangle_t \longrightarrow \langle A', st'\rangle_{t+1}}{\langle \mathsf{now}\, c\, \mathsf{then}\, A\, \mathsf{else}\, B, st\rangle_t \longrightarrow \langle A', st'\rangle_{t+1}}$	if $st \vdash_t c$
R4	$\dfrac{\langle B, st\rangle_t \longrightarrow \langle B', st'\rangle_{t+1}}{\langle \mathsf{now}\, c\, \mathsf{then}\, A\, \mathsf{else}\, B, st\rangle_t \longrightarrow \langle B', st'\rangle_{t+1}}$	if $st \nvdash_t c$
R5	$\dfrac{\langle A, st\rangle_t \not\longrightarrow}{\langle \mathsf{now}\, c\, \mathsf{then}\, A\, \mathsf{else}\, B, st\rangle_t \longrightarrow \langle A, st\rangle_{t+1}}$	if $st \vdash_t c$
R6	$\dfrac{\langle B, st\rangle_t \not\longrightarrow}{\langle \mathsf{now}\, c\, \mathsf{then}\, A\, \mathsf{else}\, B, st\rangle_t \longrightarrow \langle B, st\rangle_{t+1}}$	if $st \nvdash_t c$
R7	$\dfrac{\langle A, st\rangle_t \longrightarrow \langle A', st'\rangle_{t+1} \text{ and } \langle B, st\rangle_t \longrightarrow \langle B', st''\rangle_{t+1}}{\langle A \| B, st\rangle_t \longrightarrow \langle A' \| B', st' \sqcup st''\rangle_{t+1}}$	
R8	$\dfrac{\langle A, st\rangle_t \longrightarrow \langle A', st'\rangle_{t+1} \text{ and } \langle B, st\rangle_t \not\longrightarrow}{\langle A \| B, st\rangle_t \longrightarrow \langle A' \| B, st'\rangle_{t+1}}$	
R9	$\dfrac{\langle A, st_1 \sqcup \exists x\, st_2\rangle_t \longrightarrow \langle A', st'\rangle_{t+1}}{\langle \exists^{st_1} x\, A, st_2\rangle_t \longrightarrow \langle \exists^{st'} x\, A', st_2 \sqcup \exists x\, st'\rangle_{t+1}}$	
R10	$\langle p(x), st\rangle_t \longrightarrow \langle A, st\rangle_{t+1}$	if $p(x) : -A \in D$

Figure 1: Augmented operational semantics of the language

Theorem 5. *Consider a tccp agent A and an initial store $s_0 \in \mathcal{C}$. Let the structured store $st^0 = s_0 \cdot true^\omega$. Then*

1. *If $st \in \mathcal{O}_T(P)[\langle A, st^0\rangle]$, then $mon(st) \in \mathcal{O}(P)[\langle A, s_0\rangle]$.*
2. *If $s \in \mathcal{O}(P)[\langle A, s_0\rangle]$, then $\exists st \in \mathcal{O}_T(P)[\langle A, st^0\rangle]$ such that $\forall i \geq 0. s_i = \sqcup_{j \leq i} st_j$.*

3.3 Case Study: A Railway Crossing

We illustrate the use of tccp as a language to specify real–time systems by modeling the railway crossing example described in [Schneider 2000]. The system consists of three agents: `train`, `gate` and gate `controller`. Each agent behaves as follows:

`train` Sends `near` message to the `controller` when it is approaching the crossing. It also sends the message `out` when it has passed through the crossing.

`controller` When it receives the `near` message from the `train`, it sends the message `down` to the crossing `gate` and waits for the confirmation. When it receives the `out` message, it sends the message `up` and waits for the confirmation again.

`gate` When it receives the `down` message from the `controller` agent, it changes its state to `down` and responds properly. It behaves similarly when it receives the message `up`.

```
train(toC,T) :- ∃ toC',toC", T',T"(
      ask(true) → train(toC,T) +
      ask(true) →
         tell(toC = [near|toC'])||
         ask (true)^300 → tell(T = [enter|T']) ||
                          ask (true)^20 → tell(T' = [leave|T"]) ||
                                          tell(toC' = [out|toC"]) ||
                                          train(toC",T"))

controller(toC,toG,fromG) :- ∃ toC', toG',fromG'(
      ask(toC=[near|-]) →
         tell(toC=[near|toC']) || tell(toG=[down|toG']) ||
         ask(fromG=[confirm|-]) → tell(fromG=[confirm|fromG']) ||
                                  controller(toC',toG',fromG')
      +
      ask(toC=[out|-]) →
         tell(toC=[out|toC']) || tell(toG=[up|toG']) ||
         ask(fromG=[confirm|-]) → tell(fromG=[confirm|fromG']) ||
                                  controller(toC',toG',fromG'))

gate(fromG,toG,G):- ∃ fromG',toG',G'(
      ask(toG = [down|-]) → tell(toG=[down|toG']) ||
                            ask (true)^100 → tell(G = [down|G']) ||
                                             tell(fromG=[confirm|fromG']) ||
                                             gate(fromG',toG',G')
      +
      ask(toG = [up|-]) → tell(toG=[up|toG']) ||
                          ask (true)^100 → tell(G = [up|G']) ||
                                           tell(fromG=[confirm|fromG']) ||
                                           gate(fromG',toG',G'))

init:- ∃ toC,T,ToG,fromG,G (train(toC,T)|| controller(toC,toG,fromG)||
                            gate(fromG,toG,G))
```

Figure 2: tccp model for a railway crossing

This problem can be modeled in tccp as shown in Figure 2. The timing information encoded in the example is the following: a) the `train` takes at least 300 seconds to reach the crossing since the `near` message was sent; b) the `train` takes at least 20 seconds to cross the crossing; and, c) the `gate` takes 100 seconds to change its position following an instruction.

In order to simplify our example, we have implemented this timing information using `ask` sentences. For instance, $\texttt{ask(true)}^{100}$ represents a delay of 100 time units for the `gate` agent.

It is also interesting to note how streams are used in the example to update the values of the variables that store the position of the train and the gates at

each time instant. These streams are used as communication channels between processes, as illustrated in procedure `init`. In particular, stream `toC` is the communication channel from agent `train` to `controller`. Similarly, `toG` and `fromG` model the communication from `controller` to `gate` and viceversa, respectively.

Observe that the `tccp` implementation given in Figure 2 is a direct translation of the description given above. Agents communicate through the channels by adequately instantiating the corresponding streams. For instance, when `train` is approaching the crossing, in order to send message `near` to agent `controller`, it binds value `near` to variable `toC` by means of agent `tell(toC = [near|toC'])`. On the other hand, `controller` and `gate` are suspended in choice agents until they receive messages from the other agents to proceed consequently.

Note that the passing of time is implicit in the model. In the example, agents ignore the exact time when they are executing. Although the model could be extended to make the time explicit, it is not necessary for this example.

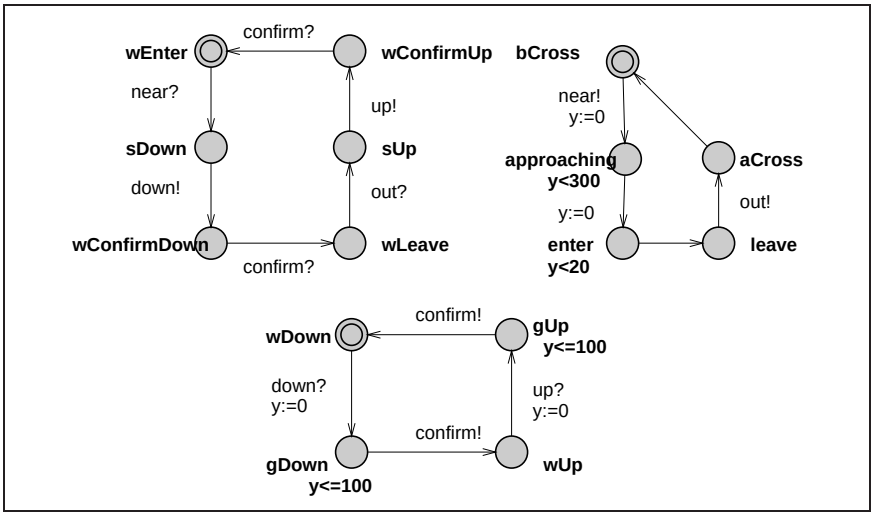

Figure 3: Timed automata for the `controller`, `train` and `gate` processes.

In Figure 3, we show the timed automata corresponding to the agents of the railway crossing example produced by UPPAAL. Variables y in processes `train` and `gate` are *local* clocks used to store the current value of time. Observe that clocks may be assigned values to fix a local initial time which is subsequently (implicitly) incremented. The automaton may use the clock values in the constraints attached to the states to temporally suspend the firing of a transition. Automata synchronize through input and output actions `near`?/`near`!, `down`?/`down`!, etc.

In Section 5.2, we will show how we have extended the considered temporal logic so that we are able to fix the current time instant when the formula is evaluated and then easily check time constraints in tccp.

4 Introducing Explicit Time in the Logic LTL

The linear temporal logic defined in [de Boer, Gabbrielli and Meo 2001] uses modalities in order to distinguish between the information *assumed* by agents prior to their execution (the *belief* information, which is supposed to be produced by the environment), and the information produced by the execution of agents (the *known* information). However, when analyzing programs by model checking, it is usual to assume that models are completely specified, i.e., the environment is considered a part of the model to be analyzed. Therefore, in this paper, we consider a simplified version of [de Boer, Gabbrielli and Meo 2001] where we get rid of modalities.

Given a constraint system $(\mathcal{C}, \vdash)$, the syntax of the temporal formulae is

$$\phi ::= c \mid \neg\phi \mid \phi \wedge \phi \mid \exists x\phi \mid \bigcirc \phi \mid \phi \,\mathcal{U}\, \phi$$

The rest of the standard propositional connectives and linear temporal operators are defined in terms of the above operators in the usual way: $\phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2)$, $\phi \rightarrow \psi = \neg\phi \vee \psi$, $\Diamond\phi = true\,\mathcal{U}\,\phi$ and $\Box\phi = \neg\Diamond\neg\phi$. Observe that, as formally defined below, the existential quantifier in the temporal formula $\exists x\phi$ is used to make variable x local to ϕ.

Definition 6. Consider the constraint system $(\mathcal{C}, \vdash)$, and a sequence of stores $s = s_0 \cdot s_1 \cdots$. The truth value of temporal formulae is defined as follows, where $s^{(i)} = s_i \cdot s_{i+1} \cdots$ is the suffix sequence of s starting at store s_i:

(1) $s^{(i)} \models c$ iff $s_i \vdash c$
(2) $s^{(i)} \models \neg\phi$ iff $s^{(i)} \not\models \phi$
(3) $s^{(i)} \models \phi_1 \wedge \phi_2$ iff $s^{(i)} \models \phi_1$ and $s^{(i)} \models \phi_2$
(4) $s^{(i)} \models \exists x\phi$ iff $s' \models \phi$, for some s' such that $\exists_x s^{(i)} = \exists_x s'$
(5) $s^{(i)} \models \bigcirc\phi$ iff $s^{(i+1)} \models \phi$
(6) $s^{(i)} \models \phi_1 \mathcal{U} \phi_2$ iff $\exists k \geq i . s^{(k)} \models \phi_2$ and $\forall i \leq j < k$, $s^{(j)} \models \phi_1$

Next, we refine the LTL logic of [de Boer, Gabbrielli and Meo 2001] by means of three improvements specially tailored to work within tccp. First, we adapt the LTL logic to deal with structured stores. Then, we formulate a notation for streams which eases the reasoning about a new, convenient class of constraints that we call *just entailed constraints*.

4.1 Augmented LTL Logic

In the following, we take advantage of structured stores in order to extend the expressiveness of LTL when modeling tccp program properties.

Definition 7. Given $t \in \mathbb{N}$, consider the constraint system $(\mathcal{C}^\omega, \vdash_t)$ and a structured store st. We define the timed satisfaction relation $\models_t$ as follows:

(1') $st \models_t c$ iff $st \vdash_t c$
(2') $st \models_t \neg\phi$ iff $st \not\models_t \phi$
(3') $st \models_t \phi_1 \wedge \phi_2$ iff $st \models_t \phi_1$ and $st \models_t \phi_2$
(4') $st \models_t \exists x\phi$ iff $st' \models_t \phi$, for some st' such that $\exists_x st = \exists_x st'$
(5') $st \models_t \bigcirc\phi$ iff $st \models_{t+1} \phi$
(6') $st \models_t \phi_1 \mathcal{U} \phi_2$ iff $\exists i \geq t. st \models_i \phi_2$ and $\forall t \leq j < i$, $st \models_j \phi_1$

Note that subindex t in $\models_t$ is variable; it represents the time instant where the temporal formula is evaluated. In the original logic, formulae are evaluated making a recursion on stores. This is possible since traces contain stores which grow monotonically. However, in order to retrieve the computed information, in the new logic, we need all the stores in the sequence. For this reason, we evaluate temporal formulae by making a recursion on time.

The following proposition demonstrates that the new satisfaction relation $\models_t$ is *equivalent* to the original one while being able to handle time in temporal formulae, as we will show later.

Proposition 8. *Given a tccp sequence of stores $s = s_0 \cdot s_1 \cdots$, a structured store st, $i \in \mathbb{N}$ and a temporal formula ϕ, then*

1. $s^{(i)} \models \phi$ *iff* $s \models_i \phi$
2. $st \models_i \phi$ *iff* $mon(st)^{(i)} \models \phi$

The new satisfaction relation $\models_t$ will be used for two main tasks: 1) to ask the accumulated store at a given time instant, as seen above; 2) to refine the constraint processing within the structured stores, as described in Section 4.3.

Let us illustrate the new satisfaction relation by means of an intuitive example. In [Alpuente *et al.* 2004a], we used LTL to express the properties of interest by using the constraints of the underlying constraint system as the atomic propositions of the logic. Assume that Figure 4 shows a structured store that is produced by the execution of a program that runs under the new operational semantics given in Figure 1. As already explained, streams are used to represent variables whose values may change during the program execution. In the example, streams X and Z range over $\mathbb{Z}$, and Y ranges on natural numbers. Moreover, $0 < n < m$ is the range of indices of the structured store.

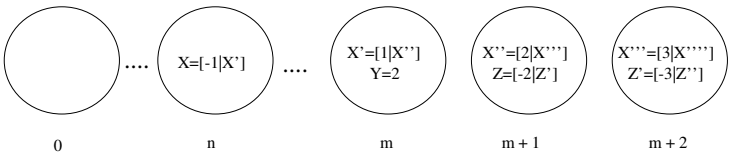

Figure 4: A Structured Store

Note that LTL allows us to reason about computation paths in terms of sequences of stores, which is a notion that (almost) coincides with the notion of structured store. If we use the new entailment relation ($\vdash_t$) over structured stores, the two notions actually do coincide. In other words, the structured store, which is represented as an array, can be seen as a sequence of stores representing a specific execution.

Let us now consider the original, non–timed tccp semantics $\mathcal{O}$ and the property P which establishes: "if $Y = 2$ then, from the next time instant on, the value of X will always be positive". This property P holds for the structured store shown in the example. However, it is difficult to express this property in temporal logic unless we introduce an explicit notion of time, since X is a stream. For instance, we could naively try to write P as the formula F given by

$$((Y = 2) \rightarrow \bigcirc\Box(\neg\exists X', N(X' = [N|_] \wedge N \leq 0)))$$

However, this expression does not match the property for two main reasons:

- F does not hold since there exists an old value of X that is negative (specifically the one given by X in the example). This happens because the store grows in a monotonic way; hence, within the original logical framework, we cannot distinguish if that instantiation occurred before or after the time instant m (when $Y = 2$).
- In addition, formula F is too restrictive since it imposes that the values of all streams (including those modeling variables that are different from variable X) must be positive from the time instant $m + 1$. This also makes F false since variable Z has a negative value in the time instant $m + 1$.

The augmented satisfaction relation $\models_t$ empowers the logic with the ability to effectively handle streams. For instance, in the case of the previous property P, we can simply express it as $Y = 2 \rightarrow \bigcirc\Box X > 0$.

If we want to model the evolution of variables values along the time, in pure tccp we have to deal with streams. In the next section, we introduce a mechanism that drastically simplifies the handling of streams in tccp. In particular, we show how to handle them as imperative variables, that is, the current value of the variable is directly accessible without recurring over the list.

4.2 Modeling Streams

When streams are used to represent variables whose values may change, then, at each time instant, the current value of a stream is the last value added to its tail, as formalized in the next definition.

Definition 9. Let X be a stream, st a structured store and $t \in \mathbb{N}$. Then, A is *the value of X in st at instant t*, denoted by $st \models_t X = [\cdots A|As]$ (or simply $st \models_t X = A$), iff $\exists m > 0$ such that:

$$st \models_t \exists A_1 \cdots \exists A_{m-1} \exists As.X = [A_1, \cdots, A_{m-1}, A|As] \; and$$
$$st \not\models_t \exists A' \exists As'.As = [A'|As']$$

In addition, under these conditions, we also say that the length of stream X in st at time t is m, in symbols $len(X, st, t) = m$.

For instance, if we call st the trace shown in Figure 4, then it holds that $st \models_m X = 1$ and $st \models_{m+1} Z = -2$.
The following notation is helpful to easily express constraints involving streams.

Definition 10. Assume that $cons(X_1, \cdots, X_n)$ is a constraint regarding the current values of streams $X_1, \cdots, X_n$, and st is a structured store. Then, st satisfies $cons(X_1, \cdots, X_n)$ in the time instant t, in symbols $st \models_t cons(X_1, \cdots, X_n)$, iff $st \models_t X_1 = A_1 \wedge \cdots \wedge X_n = A_n \wedge cons(A_1, \cdots, A_n)$.

For instance, considering the sequence of stores in Figure 4, it holds that $st \models_{m+1} X + Z = 0$ and $st \models_{m+1} \bigcirc(X + Z = 0)$.

Note that, in the temporal formulae above, we use the original names for streams X and Z, that is, the same variable identifiers that were used when variables were created. The auxiliary names created during the execution are hidden, which clearly simplifies the representation of the temporal formulae.

Now, considering the trace in Figure 4 again, it holds that $st \models_m Y = 2 \rightarrow \bigcirc\Box X > 0$ and this formula represents the property P described above in a very concise and exact way.

4.3 Just Entailed Constraints

Unfortunately, with this new logic, we cannot yet distinguish whether a given property is satisfied for the first time at a given time instant; i.e., whether or not it is a consequence of the information added to the store at a previous time instant. This is a desirable feature to have since we sometimes want to detect whether a specific situation is true as a consequence of an agent's action that has just been executed. This might help us to prevent an invalid behavior of the system when this event is detected in advance. On the other hand, by recording the time instant when a constraint has been added or entailed, we will be able to verify real–time properties, as shown in the next section.

The problem we face here is that, even if we have structured the store in such a way that the sequence of stores is not monotonic, we are dealing with a notion of entailment ($\vdash_t$) that makes the whole store monotonic again. This is because we accumulate all the information added to the store up to a given instant of time. In order to overcome this problem, we define a more refined version of the simple constraints. These new constraints are, in some sense, more demanding and more difficult to fulfill than the standard ones. Formally, given a constraint $c \in \mathcal{C}$, we introduce the new constraint $\overline{c}$, which represents that constraint c is now true for the first time. Note that when $t = 0$, all constraints are *just entailed constraints*. Thus, we extend Definition 7 by adding the following two rules:

$$(1'') \quad st \models_t \overline{c} \text{ iff } st \models_t c \text{ and } st \not\models_{t-1} c$$

$$(1''') \quad st \models_t \overline{cons(X_1, \cdots, X_n)} \text{ iff } st \models_t cons(X_1, \cdots, X_n) \text{ and } \exists 1 \leq i \leq n. len(X_i, st, t) > len(X_i, st, t-1)$$

The above definition establishes that it is possible for a constraint c that was just entailed in the previous time instant, to be again just entailed in the subsequent time instant whenever a variable occurring in the constraint changes its value. This is because we are interested in modeling all interactions among processes through constraints, including the fact that a computation is eventually redone.

Note that, by definition, constraints that are just entailed at a certain instant then hold at that instant. However, the opposite is clearly not true.

5 Analysis of Real–Time Properties in tccp

In this section, we first propose a simple real–time extension for the refined LTL logic given so far, and show how to use it for the analysis of real–time properties.

5.1 A Simple Real–Time Logic

Let us first introduce some helpful definitions. Consider the *timed cylindric constraint system* $\langle \mathcal{C}_T, \leq_T, true, false, Var_T, \exists \rangle$ where $\mathcal{C}_T$ is a set of (unstructured)

stores that introduces a distinguished class of timing constraints that consists of boolean expressions with the usual arithmetic operators. Var_T is an infinite set of variables used only to record times, such that $Var \cap Var_T = \emptyset$, with Var being the set of variables used in tccp programs. Let us denote with $\vdash_T$ the corresponding entailment relation. Roughly speaking, $\mathcal{C}_T$ stores will be constructed along the evaluation of temporal formulae to record the precise time instants where certain constraints of interest are proven. They will typically include expressions of the form $t = m$, or $t \leq t' + m$ where $t, t' \in Var_T$ and $m \in \mathbb{N}$.

Definition 11. Let $\mathcal{F}$ be the set of temporal formulae constructed with the elements of $\mathcal{C} \cup \overline{\mathcal{C}}$, the usual boolean connectives and the temporal operators. An *annotated formula* is an element of $\mathcal{E} = \mathcal{F} \times \mathcal{C}_T \times Var_T$, where the first component is a classic temporal formula, the second one is a timing constraint to be evaluated together with the temporal formula, and the last one is used to record the time instant when the temporal formula is proven.

The semantics of annotated temporal formulae is formalized as follows. Consider $\langle st, \tau \rangle$, where st is a structured store, $\tau \in \mathcal{C}_T$, and $\langle \phi, r, t \rangle \in \mathcal{E}$. Then,

$$\begin{array}{l}
\langle st, \tau \rangle \models_m \langle \phi, r, t \rangle \iff st \models_m \phi \text{ and } \tau \sqcup \{t = m\} \vdash_T r \\
\langle st, \tau \rangle \models_m \neg\langle \phi, r, t \rangle \iff \langle st, \tau \rangle \not\models_m \langle \phi, r, t \rangle \\
\langle st, \tau \rangle \models_m \langle \phi_1, r_1, t_1 \rangle \wedge \langle \phi_2, r_2, t_2 \rangle \iff \langle st, \tau \rangle \models_m \langle \phi_1, r_1, t_1 \rangle \text{ and} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle st, \tau \rangle \models_m \langle \phi_2, r_2, t_2 \rangle \\
\langle st, \tau \rangle \models_m \exists x \langle \phi, r, t \rangle \iff \langle st', \tau \rangle \models_m \langle \phi, r, t \rangle \text{ for some } st' \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{such that } \exists_x st = \exists_x st'. \\
\langle st, \tau \rangle \models_m \bigcirc \langle \phi, r, t \rangle \iff \langle st, \tau \sqcup \{t = m\} \rangle \models_{m+1} \langle \phi, r, t' \rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{where } t' \in Var_T \text{ is a fresh variable} \\
\langle st, \tau \rangle \models_m \langle \phi, r, t_m \rangle \mathcal{U} \langle \phi', r', t' \rangle \iff \exists k \geq m. \forall m \leq j < k. \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle st, \tau \sqcup_{i=m}^{j-1} (t_i = i) \rangle \models_j \langle \phi, r, t_m \rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle st, \tau \sqcup_{i=m}^{k-1} (t_i = i) \models_k \langle \phi', r', t' \rangle
\end{array}$$

The following proposition establishes the precise relation between LTL and the real–time extension that we introduced.

Proposition 12. *Consider* $st \in \text{STORE}$, $\tau \in \mathcal{C}_T$, $\phi \in \mathcal{F}$, $t \in Var_T$, $m \in \mathbb{N}$ *and* $r_1, r_2 \in \mathcal{C}$. *Then,*

(a) $\langle st, \tau \rangle \models_m \langle \phi, true, t \rangle \iff st \models_m \phi$

(b) $r_1 \vdash r_2, \langle st, \tau \rangle \models_m \langle \phi, r_1, t \rangle \Rightarrow \langle st, \tau \rangle \models_m \langle \phi, r_2, t \rangle$

Note that, as an easy consequence of (a) and (b) above, if $r \vdash true$ and $\langle st, \tau \rangle \models_m \langle \phi, r, t \rangle$, then $st \models_m \phi$.

5.2 Case study: A railway crossing (II)

Consider again the tccp model for the railway crossing example given in Section 3.3. We can now specify different timed properties for this model by using the real–time logic outlined in Section 5. For instance:

Property 1: "When the train is near the crossing, it takes less than 300 seconds to lower the gate".

$$\Box\langle\overline{toC = \texttt{near}}, true, t\rangle \rightarrow \bigcirc\Diamond\langle\overline{G = \texttt{down}}, t' \leq t + 300, t'\rangle$$

Observe that in this annotated formula we are imposing two important real–time constraints which are difficult to express with other formalisms. On the one hand, the state where `gate` is `down` must occur *after* `train` sends message `near` (`down` must be a new value just given to stream `G`). On the other hand, even if many future states may exist where `gate` has just got down, we are only interested in a future state occurring before time passes 300 seconds.

Property 2: "When the train enters the crossing, the gate is down, and it remains down at least 20 seconds".

$$\Box(\langle\overline{T = \texttt{enter}}, true, t_0\rangle \rightarrow \langle G = \texttt{down}, true, t_1\rangle \mathcal{U} \langle\overline{G = \texttt{up}}, t_0 + 20 \leq t_2, t_2\rangle)$$

Note that time variables in these examples correspond to clocks in the timed automata–based approach. In our case, time variables store the time instant when a particular formula is evaluated, and we are able to reason about time constraints regarding their values in a very natural way.

In order to verify these properties on tccp programs, it suffices to extend the standard model–checking algorithms by recording how every layer of the structured store represents a clock tick.

5.3 Verification of Real–Time Properties by Model Checking

It is mostly technical and not difficult to adapt the model–checking methodology introduced in [Falaschi and Villanueva 2006] to deal with the new tccp model and real–time formulae. The explicit model–checking algorithm described there works as follows:

1. A tccp program following the syntax defined in Section 2 is translated into a tccp Structure, which is similar to a Kripke structure.

2. The temporal formula to be verified, following the syntax given in Definition 6 is used to construct an extension of the tccp Structure called the *model–checking graph*.

3. This graph is explored while looking for a counterexample of the formula.

Some aspects which are specific to tccp must be considered. tccp programs can have an infinite number of states. For a specific kind of infinite–state systems and properties, the algorithm is able to find the counterexample in a finite number of steps. However, in some cases, the algorithm might not finish. For this reason, a time bound is introduced so that when such a bound is reached, the execution of the algorithm is stopped. For a detailed description, we refer to [Falaschi and Villanueva 2006].

The construction of the tccp Structure relies on the operational semantics of the language. When we consider the real–time extension proposed in this work we need to deal with structured stores and then substitute the traditional entailment relation by the new one augmented with time. Nevertheless, both modifications are straightforward due to the fact that the states of the tccp Structure are explicit. Once we have introduced the temporal component in the model (i.e., in the tccp Structure), and by using a constraint system empowered to deal with arithmetic constraints as we described in Definition 11, the construction of the model–checking graph is merely technical. The rest of procedures such as computing the strong connected components and finding the counterexample remain unchanged.

6 Conclusions

In this work, we propose an approach to deal with real–time properties within the *timed concurrent constraint programming* framework tccp. This allows us to improve the *model–checking* methodology previously developed for tccp so that more sophisticated, quantitative properties can be verified by model–checking tools. Since we find that the essential drawback of the original tccp framework is that tccp stores are unstructured, first we structured the store and then consistently adapted the operational semantics of the language as well as the LTL logic to deal with time instant marks. We have also proposed a simple methodology to handle streams within LTL formulae which is based on the notion of *just entailed* constraint. Finally, we have refined the LTL logic in order to model real–time properties, and we have illustrated our method by means of a leading example.

We are currently working on a prototype implementation of our methodology. As future work, we plan to explore the possibility of also dealing with past real–time formulae.

Acknowledgments

This work has been partially supported by the EU (FEDER) and the Spanish MEC under grant TIN2004-7943-C04, and the ICT for EU-India Cross-Cultural Dissemination ALA/95/23/2003/077-054 project.

References

[Alpuente *et al.* 2004a] M. Alpuente, M.M. Gallardo, E. Pimentel, and A. Villanueva. Abstract Model Checking of tccp programs. In *Proc. of the 2nd Works. on Quantitative Aspects of Programming Languages (QAPL 2004)*, volume 112 of *ENTCS*, pages 19–36. Elsevier Science, 2004.

[Alpuente *et al.* 2005a] M. Alpuente, M. Falaschi, and A. Villanueva. A Symbolic Model checker for tccp Programs. In *Proc. of the Int. Works. on Rapid Integration of Software Ingeneering techniques (RISE'04)*, LNCS 3475, pages 45–56. Springer Verlag, 2005.

[Alpuente *et al.* 2005b] M. Alpuente, M.M. Gallardo, E. Pimentel, and A. Villanueva. A Semantic Framework for the Abstract Model Checking of tccp programs. *Theoretical Computer Science*, 346:58–95, 2005.

[Alur and Dill 1994] R. Alur and D. L. Dill. A theory of timed automata *Journal of Theoretical Computer Science*, 126(2):183–235, 1994.

[Alur and Henzinger 1994] R. Alur and T. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

[de Boer, Gabbrielli and Meo 1999] F. S. de Boer, M. Gabbrielli, and M. C. Meo. A Timed Concurrent Constraint Language. *Information and Computation*, 161:45–83, 2000.

[de Boer, Gabbrielli and Meo 2001] F. S. de Boer, M. Gabbrielli, and M. C. Meo. A Temporal Logic for reasoning about Timed Concurrent Constraint Programs. In *Proc. of 8th Int. Symp. on Temporal Representation and Reasoning*, pages 227–233. IEEE Computer Society Press, 2001.

[Clarke, Emerson and Sistla 1993] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. In *Proc. of the 10th ACM Symp. on Princ. of Progr. Languages*, pages 117–126. ACM Press, 1983.

[Falaschi and Villanueva 2006] M. Falaschi and A. Villanueva. Automatic Verification of Timed Concurrent Constraint programs. *Theory and Practice of Logic Programming*, 6(3):265–300, 2006.

[Larsen, Petterson and Yi 1997] K.G. Larsen, P. Petterson, and W. Yi. UPPAAL in a nutshell *Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[Manna and Pnueli 1992] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems - Specification*. Springer-Verlag, New York, 1992.

[Saraswat 1993] V. A. Saraswat. *Concurrent Constraint Programming Languages*. The MIT Press, Cambridge, MA, 1993.

[Schneider 2000] S. Schneider. *Concurrent and Real–time Systems. The CSP Approach*. Wiley, 2000.

[Yovine 1997] S. Yovine. KRONOS: A Verification Tool for Real–Time Systems *Journal on Software Tools for Technology Transfer*, 1(1-2):123–133, 1997.

A Proofs

This appendix contains the proofs of all technical results, and the original operational semantics of tccp (Figure 5) used to prove Theorem 5.

r1	$\langle \mathsf{tell}(c), s\rangle \longrightarrow \langle \mathsf{stop}, s \sqcup c\rangle$	
r2	$\langle \sum_{i=0}^{n} \mathsf{ask}(c_i) \rightarrow A_i, s\rangle \longrightarrow \langle A_j, s\rangle$	if $0 \leq j \leq n$ and $s \vdash c_j$
r3	$\dfrac{\langle A, s\rangle \longrightarrow \langle A', s'\rangle}{\langle \mathsf{now}\, c\, \mathsf{then}\, A\, \mathsf{else}\, B, s\rangle \longrightarrow \langle A', s'\rangle}$	if $s \vdash c$
r4	$\dfrac{\langle B, s\rangle \longrightarrow \langle B', s'\rangle}{\langle \mathsf{now}\, c\, \mathsf{then}\, A\, \mathsf{else}\, B, s\rangle \longrightarrow \langle B', s'\rangle}$	if $s \nvdash c$
r5	$\dfrac{\langle A, s\rangle \not\longrightarrow}{\langle \mathsf{now}\, c\, \mathsf{then}\, A\, \mathsf{else}\, B, s\rangle \longrightarrow \langle A, s\rangle}$	if $s \vdash c$
r6	$\dfrac{\langle B, s\rangle \not\longrightarrow}{\langle \mathsf{now}\, c\, \mathsf{then}\, A\, \mathsf{else}\, B, s\rangle \longrightarrow \langle B, s\rangle}$	if $s \nvdash c$
r7	$\dfrac{\langle A, s\rangle \longrightarrow \langle A', s'\rangle \text{ and } \langle B, s\rangle \longrightarrow \langle B', s''\rangle}{\langle A \| B, s\rangle \longrightarrow \langle A' \| B', s' \sqcup s''\rangle}$	
r8	$\dfrac{\langle A, s\rangle \longrightarrow \langle A', s'\rangle \text{ and } \langle B, s\rangle \not\longrightarrow}{\langle A \| B, s\rangle \longrightarrow \langle A' \| B, s'\rangle}$	
r9	$\dfrac{\langle A, s_1 \sqcup \exists x\, s_2\rangle \longrightarrow \langle A', s'\rangle}{\langle \exists^{s_1} x\, A, s_2\rangle \longrightarrow \langle \exists^{s'} x\, A', s_2 \sqcup \exists x\, s'\rangle}$	
r10	$\langle p(x), s\rangle \longrightarrow \langle A, s\rangle$	if $p(x) :- A \in D$

Figure 5: Original operational semantics of tccp

Theorem 5 Consider a tccp agent A and an initial store $s_0 \in \mathcal{C}$. Let $st^0 = s_0 \cdot true^\omega$. Then

1. If $st \in \mathcal{O}_T(P)[\langle A, st^0\rangle]$, then $mon(st) \in \mathcal{O}(P)[\langle A, s_0\rangle]$.
2. If $s \in \mathcal{O}(P)[\langle A, s_0\rangle]$, then $\exists st \in \mathcal{O}_T(P)[\langle A, st^0\rangle]$ such that $\forall i \geq 0. s_i = \sqcup_{j \leq i} st_j$.

Proof. 1. The proof follows trivially from the definitions of $\vdash_t$ and $mon(s)$ given above.

2. If $s = s_0 \cdot \ldots s_n \cdots \in \mathcal{O}(P)[\langle A, s_0\rangle]$ then there exists a sequence of agents $A = A^0, A^1, \ldots$, such that $\langle A^0, s_0\rangle \longrightarrow \langle A^1, s_1\rangle \longrightarrow \ldots$ is an execution trace produced by the original semantics of tccp.

   Let us prove the following assertion: Given $st^n = st_0^0 \cdot st_1^1 \cdot \cdots \cdot st_n^n \cdot true^\omega$ such that $\forall i \leq n. s_i = \sqcup_{j \leq i} st_j^j$ then there exists $c \in \mathcal{C}$ such that we have a transition $\langle A^n, st^n\rangle_n \longrightarrow \langle A^{n+1}, st^{n+1}\rangle_{n+1}$ in the timed semantics of tccp where $st^{n+1} = st_0^0 \cdot st_1^1 \cdot \cdots \cdot st_n^n \cdot c \cdot true^\omega$ and $s_{n+1} = \sqcup_{i \leq n+1} st_i^i$.

Note that since $st^0 = s_0 \cdot true^\omega$, the previous result allows us to construct an structured store $st = st_0^0 \cdot st_1^1 \cdot \cdots \in \mathcal{O}_T(P)[\langle A, st^0\rangle]$ satisfying that $\forall i \geq 0. s_i = \sqcup_{j \leq i} st_i$ as stated in the Theorem.

We prove the previous assertion by induction structural induction on A^n.

$A^n = \mathbf{tell}(c)$**.** Then, following the classical operational semantics, $s_{n+1} = s_n \sqcup c$. Now, using rule **R1** of the timed semantics, we have that $st^{n+1} = st_0^0 \ldots st_n^n \cdot c \cdot true^\omega$. Since, by hypothesis $s_n = \sqcup_{i \leq n} st_i^i$, we obtain that $s_{n+1} = s_n \sqcup c = \sqcup_{i \leq n+1} st_i^i$.

$A^n = \sum_{j=0}^{n} \mathbf{ask}(c_j) \rightarrow A_j$**.** Let us assume that store s_n entails any of the guards c_j. The original tccp semantics gives us that $s_{n+1} = s_n$. Now, since by hypothesis $s_n = \sqcup_{i \leq n} st_i^i$ and $s_n \vdash c_j$, we deduce that $st^n \vdash_t c_j$. Thus, applying rule **R2** of the timed tccp semantics, we have that $st^{n+1} = st_0^0 \cdot \cdots \cdot st_n^n \cdot true \cdot true^\omega$, and clearly $s_{n+1} = s_n = \sqcup_{i \leq n+1} st_i^i$.

$A^n = \mathbf{now}\, c\, \mathbf{then}\, A_1\, \mathbf{else}\, A_2$**.** We may consider two cases for this agent. One is the case when the store entails the constraint c, and the other is the opposite (the constraint is not entailed). However, since both cases are proved similarly, we only deal with one of them.

Let us assume that $s_n \vdash c$, and that $\langle A_1, s_n\rangle \longrightarrow \langle A^{n+1}, s_{n+1}\rangle$. Then, by structural induction, there exists $d \in \mathcal{C}$ such that $\langle A_1, st_n^n\rangle \longrightarrow \langle A^{n+1}, st^{n+1}\rangle$ where $st^{n+1} = st_0^0 \cdot \cdots \cdot st_n^n \cdot d \cdot true^\omega$ and $s_{n+1} = \sqcup_{i \leq n+1} st_i^i$. Since by hypothesis, $s_n = \sqcup_{i \leq n} st_i^i$, we have that $st^n \vdash_t c$ too. Thus, applying rule **R3** of the timed semantics, we obtain the desired result.

The case when $s_n \vdash c$ and $\langle A_1, s_n\rangle \not\longrightarrow$ is proved as in the choice agent above.

$A^n = A_1 \,||\, A_2$**.** We have two cases, one when both A_1 and A_2 progress, and the second case when only one of the agents does. As before, we describe the reasoning for the first case since the second one is similar. Thus, let us assume that $\langle A_1, s_n\rangle \longrightarrow \langle A_1', s_{n+1}^{A_1}\rangle$ and that $\langle A_2, s_n\rangle \longrightarrow \langle A_2', s_{n+1}^{A_2}\rangle$. Then applying the standard semantics of tccp, we deduce that $A^{n+1} = A_1' || A_2'$ and $s_{n+1} = s_{n+1}^{A_1} \sqcup s_{n+1}^{A_2}$. By structural induction on A_1 and A_2, we have that there exist $c_1, c_2 \in \mathcal{C}$ such that $\langle A_1, st^n\rangle_n \longrightarrow \langle A_1', st_{A_1}^{n+1}\rangle_{n+1}$, $\langle A_2, st^n\rangle_n \longrightarrow \langle A_2', st_{A_2}^{n+1}\rangle_{n+1}$ and for $i = 1, 2$ $st_{A_i}^{n+1} = st_0^0 \cdot \cdots \cdot st_n^n \cdot c_i \cdot true^\omega$, and $s_{n+1}^{A_i} = \sqcup_{j \leq n} st_j^j \sqcup c_i$. Now, applying rule **R7** of the timed semantics, we obtain that $\langle A^n, st^n\rangle_n \longrightarrow \langle A^{n+1}, st^{n+1}\rangle_{n+1}$, where $st^{n+1} = st_{A_1}^n \sqcup st_{A_2}^n$. Thus, by definition of $\sqcup$ for structured stores, we have that $st^{n+1} = st_0^0 \cdot \cdots \cdot st_n^n \cdot c_1 \sqcup c_2 \cdot true^\omega$, and since $s_{n+1} = s_{n+1}^{A_1} \sqcup s_{n+1}^{A_2}$, using the expressions obtained for $s_{n+1}^{A_1}$ and $s_{n+1}^{A_2}$ by the induction step, we deduce that $s_{n+1} = \sqcup_{i \leq n+1} st_i^i$.

$A^n = \exists x A_1$**.** Let us assume that the information generated by the execution of $A_1[y/x]$ (y fresh variable) is s_{A_1}. Then, following the standard semantics, the configuration at instant $n+1$ is $\langle A_1', s_n \sqcup \exists_x s_{A_1}\rangle$. By structural induction hypothesis, the same information is produced by the agent A_1 in the structured semantics, thus the structured configuration at instant $n+1$ will be $\langle A_1', st_{n+1}\rangle$ where $st_{n+1} = st_{n+1}^{n+1} = st_{n+1}^{n} \sqcup \exists_x s_{A_1}\rangle$.

$A^n = p(x)$**.** This case is trivial since we consider the same program under two different semantics. By induction hypothesis, A^n are identical, thus we obtain a configuration where the property does hold.

Proposition 8 Given a tccp sequence of stores $s = s_0 \cdot s_1 \cdots$, a structured store st, $i \in \mathbb{N}$ and a temporal formula ϕ, then

1. $s^{(i)} \models \phi$ iff $s \models_i \phi$
2. $st \models_i \phi$ iff $mon(st)^{(i)} \models \phi$

Proof. 1. We proceed by structural induction on the formula ϕ.

$\phi \equiv c$**:** By definition, we know that $s \models_i c \Leftrightarrow s \vdash_i c \Leftrightarrow \sqcup_{0\leq j\leq i} s_j \vdash c$. Since sequence $s_0 \cdot s_1 \cdot \ldots$ is monotonic $\sqcup_{0\leq j\leq i} s_j = s_i$. Thus, $\sqcup_{0\leq j\leq i} s_j \vdash c \Leftrightarrow s_i \models c \Leftrightarrow s^{(i)} \models c$.

$\phi \equiv \neg\psi$**:** By definition, we know that $s^{(i)} \models \neg\psi \Leftrightarrow s^{(i)} \not\models \psi$ and that $s \models_i \neg\psi \Leftrightarrow s \not\models_i \psi$. We assume by induction hypothesis that $s^{(i)} \models \psi \Leftrightarrow s \models_i \psi$, thus the property trivially holds.

$\phi \equiv \psi_1 \wedge \psi_2$**:** By definition, $s^{(i)} \models \psi_1 \wedge \psi_2 \Leftrightarrow s^{(i)} \models \psi_1$ and $s^{(i)} \models \psi_2$. On the other hand, $s \models_i \psi_1 \wedge \psi_2 \Leftrightarrow s \models_i \psi_1$ and $s \models_i \psi_2$. By applying induction hypothesis the property trivially holds.

$\phi \equiv \exists x\psi$**:** Let us assume that $s \models_i \exists\psi$. Then, by definition, there exists a structure store s' such that $\exists_x s' = \exists_x s$, and $s' \models_i \psi$. By induction hypothesis, this implies that $s'^{(i)} \models \psi$. Now, since $\exists_x s' = \exists_x s$ we deduce that $\exists_x s'^{(i)} = \exists_x s^{(i)}$, which, by definition, implies that $s^{(i)} \models \exists x\psi$.

Inversely, let us assume that $s^{(i)} \models \exists x\psi$ then, by definition, $s' \models \psi$ for some s' such that $\exists_x s^{(i)} = \exists_x s'$. Let us construct the structured store $s'' = s_0 \cdot s_1 \cdots s_{i-1} \cdot s_0' \cdot s_1' \cdots$, that is, the first $i-1$ stores of s'' are the first $i-1$ stores of s, and the suffix s''^i is s'. By construction, $\exists_x s = \exists_x s''$, and $s'' \models_i \psi$, which by definition, means that $s \models_i \psi$.

$\phi \equiv \bigcirc\psi$**:** By definition $s^{(i)} \models \bigcirc\psi \Leftrightarrow s^{(i+1)} \models \psi$ and $s \models_i \bigcirc\psi \Leftrightarrow s \models_{i+1} \psi$. Therefore, it suffices to see that $s^{(i+1)} \models \psi \Leftrightarrow s \models_{i+1} \psi$, which is true by induction hypothesis.

$\phi \equiv \psi_1 \mathcal{U} \psi_2$**:** We know that $s^{(i)} \models \psi_1 \mathcal{U} \psi_2 \Leftrightarrow \exists j \geq i, s^{(j)} \models \psi_2$ and $\forall i \leq k < j, s^{(k)} \models \psi_1$. By induction hypothesis, we have that $\exists j \geq i, s \models_j \psi_2$ and $\forall i \leq k < j, s \models_k \psi_1$, or equivalently that $s \models_i \psi_1 \mathcal{U} \psi_2$.

2. By definition of *mon*, the proof of this claim is similar to the previous one.

Proposition 12 Consider $st \in \textsc{Store}$, $\tau \in \mathcal{C}_T$, $\phi \in \mathcal{F}$, $t \in Var_T$, $m \in \mathbb{N}$ and $r_1, r_2 \in \mathcal{C}$. Then,

(*a*) $\langle st, \tau \rangle \models_m \langle \phi, \mathit{true}, t \rangle \iff st \models_m \phi$

(*b*) $r_1 \vdash r_2, \langle st, \tau \rangle \models_m \langle \phi, r_1, t \rangle \Rightarrow \langle st, \tau \rangle \models_m \langle \phi, r_2, t \rangle$

Proof. a) $\Rightarrow$ holds by definition. Inversely, if $st \models_m \phi$, since $\tau \sqcup \{t = m\} \vdash_T \mathit{true}$, we have that $\langle st, \tau \rangle \models_m \langle \phi, \mathit{true}, t \rangle$.

b) By definition, $\langle st, \tau \rangle \models_m \langle \phi, r_1, t \rangle$ implies $st \models_m \phi$ and $\tau \sqcup \{t = m\} \vdash_T r_1$. Since $r_1 \vdash_T r_2$, we deduce that $\tau \sqcup \{t = m\} \vdash_T r_2$ which, by definition, means that $\langle st, \tau \rangle \models_m \langle \phi, r_2, t \rangle$