

A Semantic Framework for the Abstract Model Checking of `tccp` programs *

M. Alpuente M.M. Gallardo, E. Pimentel A. Villanueva

U. Politécnic de Valencia

U. de Málaga

U. Politécnic de Valencia

Camino de Vera s/n

Campus de Teatinos s/n

Camino de Vera s/n

46022 Valencia

29071 Málaga

46022 Valencia

alpuente@dsic.upv.es

{gallardo,ernesto}@lcc.uma.es

villanue@dsic.upv.es

Abstract

The *Timed Concurrent Constraint* programming language (`tccp`) introduces time aspects into the Concurrent Constraint paradigm. This makes `tccp` especially appropriate for analyzing timing properties of concurrent systems by model checking. However, even if very compact state representations are obtained thanks to the use of constraints in `tccp`, large state spaces can still be generated, which may prevent model-checking tools from verifying `tccp` programs completely. In this work, we introduce an abstract methodology which is based on over- and under-approximating `tccp` models and which mitigates the state explosion problem that is common to traditional model-checking algorithms. We ascertain the conditions for the correctness of the abstract technique. We complete our methodology by approximating the temporal properties that must be verified.

1 Introduction

In the past few years, some extensions of the concurrent constraint paradigm [3, 9] have been defined in order to model reactive systems. All these extensions introduce a quantitative notion of time that makes it possible to model the typical ingredients of these sys-

tems, such as timeouts, preemptions, etc. In this work, we develop a suitable approximation methodology that is based on abstract interpretation (AI) [6] in order to drastically reduce the state space of model checking `tccp`.

Applying abstract model checking involves the abstraction of both the model to be analyzed and the properties to be checked within the model. In the classic abstract model-checking literature, the abstract model is an over-approximation of the concrete model M . However, due to the double, logical as well as temporal dimension of `tccp`, inaccurate abstract models would be obtained in our context by simple over-approximation. In order to achieve fine accuracy, in this paper we define *abstract operations* that over-approximate the original ones (see [7]), and combine over- and under-approximation in the abstraction of `tccp` operators.

Applying AI in presence of quantifiable information such as time, raises other specific problems which are related to the process synchronization. In `tccp`, processes are totally synchronized meaning that, at each time instant, all enabled agents are simultaneously carried out. Unfortunately, the loss of information caused by the abstraction affects the suspension behavior of processes: the suspension of a process in the original model does not generally imply that the process abstractly suspends; hence synchronization in the abstract model might be damaged, and we have to refine the abstract semantics in order to overcome the suspension mismatch.

*This work has been partially supported by the EU (FEDER) and the Spanish MEC under grant TIN 2004-7943-C04, the Generalitat Valenciana under grant GV03/25, and the ICT for EU-India Cross-Cultural Dissemination ALA/95/23/2003/077-054 project.

We have developed in [1, 2] a complete methodology for abstracting **tccp** programs. Summarizing, in these works, 1) we propose an abstract model-checking methodology that mitigates the state explosion problem in **tccp** model checking; 2) We present the first formal proof for the total correctness of a refined abstract semantics which that models the suspension behavior of processes; 3) We develop a source-to-source transformation for **tccp** programs that is the basis for a natural implementation of our method; and 4) we develop an approximation technique for checking the satisfiability of the temporal properties that must be verified. For space reasons, we only present here a selection of these contributions.

2 The **tccp** language

In [3], the *Timed Concurrent Constraint* language (**tccp**) was defined as an extension of the Concurrent Constraint programming language **ccp** [8]. 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. cylindric constraint system \mathcal{C} . In **tccp**, a new (w.r.t. **ccp**) conditional agent **now** c **then** A **else** B is introduced which makes it possible to model situations where the absence of information can cause the execution of a specific action.

In the full version of the paper we show the definition of cylindric constraint system which is a particular class of simple constraint system. In this summary we refer only to simple constraint systems:

Definition 1 A simple constraint system is a structure $\langle \mathcal{C}, \vdash \rangle$ where \mathcal{C} is the set of atomic constraints and relation $\vdash \subseteq \wp(\mathcal{C}) \times \mathcal{C}$ satisfies

- C1. $u \vdash C$, for all $C \in u$.
- C2. $u \vdash C$, if $\forall C' \in v, u \vdash C'$ and $v \vdash C$.

Relation \vdash can be extended to $\vdash \subseteq \wp(\mathcal{C}) \times \wp(\mathcal{C})$ as $u \vdash v \iff \forall C \in v, u \vdash C$ which is reflexive and transitive.

During **tccp** computations, stores are represented by elements of $\Theta = \wp(\mathcal{C})$. Let us recall the **tccp** syntax for agents:

$$A ::= \text{stop} \mid \text{tell}(c) \mid \sum_{i=0}^n \text{ask}(c_i) \rightarrow A_i \mid \text{now } c \text{ then } A \text{ else } A \mid A \parallel A \mid \exists x A \mid \mathbf{p}(x)$$

where c, c_i is a finite set of constraints of \mathcal{C} . A **tccp process** P has the form $D.A$, where D is a set of procedure declarations $\mathbf{p}(x):-B$, and B is an agent.

3 Abstract **tccp** programs

An *abstract interpretation* (an *abstraction*) of the constraint system $\langle \mathcal{C}, \vdash \rangle$ is given by an *upper closure operator* (uco) $\rho : \wp(\Theta) \rightarrow \wp(\Theta)$. The intuition of this definition is that each $st \in \Theta$ is abstracted by its closure $\rho(\{st\})$.

Using abstract interpretation terminology, $\rho(\{st\})$ is the most precise abstraction of the store $st \in \Theta$ and, if $\rho(\{st\}) \subseteq sst$, then sst is also an abstraction of st . Thus roughly speaking, an abstract store is a set of concrete stores.

Definition 2 introduces two dual entailment relations for abstracting constraint systems.

Definition 2 Let $\langle \mathcal{C}, \vdash \rangle$ be a simple constraint system and ρ be a constraint abstraction. Then, we define the over- and under-approximated constraint systems $\langle \Theta, \vdash_{\rho}^{+} \rangle$ and $\langle \Theta, \vdash_{\rho}^{-} \rangle$ where $\vdash_{\rho}^{+}, \vdash_{\rho}^{-} \subseteq \wp(\Theta) \times \wp(\Theta)$, by:

$$\begin{aligned} sst_1 \vdash_{\rho}^{+} sst_2 &\iff \exists u \in \rho(sst_1), \exists v \in sst_2. u \vdash v \\ sst_1 \vdash_{\rho}^{-} sst_2 &\iff \forall u \in \rho(sst_1), \exists v \in sst_2. u \vdash v \end{aligned}$$

The names given above are justified because

- (1) If $u \vdash v$, then $\{u\} \vdash_{\rho}^{+} \{v\}$.
- (2) If $\{u\} \vdash_{\rho}^{-} \{v\}$, then $u \vdash v$.

The abstract **tccp** semantics developed below makes use of the abstract union operator $\sqcup^{\rho} : \wp(\Theta) \rightarrow \wp(\Theta)$ defined as $sst_1 \sqcup^{\rho} sst_2 = \rho(\{u \cup v \mid u \in sst_1, v \in sst_2, u \cup v \not\vdash \text{false}\})$.

3.1 Abstract Semantics

As it is shown in [10], the **ask-tell** paradigm introduces some problems when we deal with abstraction. When dealing with **tccp**, abstraction is more difficult to apply due to the temporal dimension and the maximal parallelism.

In Figure 1 we formalize a preliminary abstract operational semantics of **tccp** programs. A *configuration* is a pair $\langle \Gamma, sst \rangle$ where Γ is an agent and $sst \in \wp(\Theta)$ is an abstract store. In the following, we assume that an abstraction operator $\rho : \wp(\Theta) \rightarrow \wp(\Theta)$ has been provided. We drop the subindex ρ from \vdash_{ρ}^{+} , \vdash_{ρ}^{-} and \sqcup^{ρ} in order to simplify the presentation.

(0) $\langle \text{stop}^\alpha, sst \rangle \longrightarrow_\alpha \langle \text{stop}^\alpha, sst \rangle$
(1) $\langle \text{tell}^\alpha(c), sst \rangle \longrightarrow_\alpha \langle \text{stop}^\alpha, sst \sqcup \{c\} \rangle$
(2) $\frac{\exists j. sst \vdash^+ \{c_j\}}{\langle \sum_{i=0}^n \text{ask}^\alpha(c_i) \rightarrow A_i, sst \rangle \longrightarrow_\alpha \langle A_j, sst \rangle}$
(3) $\frac{sst \not\vdash^- \{c_0, \dots, c_n\}}{\langle \sum_{i=0}^n \text{ask}^\alpha(c_i) \rightarrow A_i, sst \rangle \longrightarrow_\alpha \langle \sum_{i=0}^n \text{ask}^\alpha(c_i) \rightarrow A_i, sst \rangle}$
(4) $\frac{\exists j. sst \vdash^+ \{c_j\}, \langle A_j, sst \rangle \longrightarrow_\alpha \langle A'_j, sst' \rangle}{\langle \sum_{i=0}^n \text{ask}!(c_i) \rightarrow A_i, sst \rangle \longrightarrow_\alpha \langle A'_j, sst' \rangle}$
(5) $\frac{\langle A, sst \rangle \longrightarrow_\alpha \langle A', sst' \rangle, sst \vdash^- c}{\langle \text{now}^\alpha c \text{ then } A \text{ else } B, sst \rangle \longrightarrow_\alpha \langle A', sst' \rangle}$
(6) $\frac{\langle B, sst \rangle \longrightarrow_\alpha \langle B', sst' \rangle, sst \not\vdash^- c}{\langle \text{now}^\alpha c \text{ then } A \text{ else } B, sst \rangle \longrightarrow_\alpha \langle B', sst' \rangle}$
(7) $\frac{\langle A, sst \rangle \longrightarrow_\alpha \langle A', sst'_1 \rangle, \langle B, sst \rangle \longrightarrow_\alpha \langle B', sst'_2 \rangle}{\langle A B, sst \rangle \longrightarrow_\alpha \langle A' B', sst'_1 \sqcup sst'_2 \rangle}$
(8) $\frac{\langle A, sst_1 \sqcup \exists x sst_2 \rangle \longrightarrow_\alpha \langle A', sst' \rangle}{\langle \exists sst_1 x A, sst_2 \rangle \longrightarrow_\alpha \langle \exists sst' x A', sst_2 \sqcup \exists x sst' \rangle}$
(9) $\langle p(x), sst \rangle \longrightarrow_\alpha \langle A, sst \rangle \quad \text{if } p(x) :- A \in D$

Figure 1: Correct abstract operational semantics

Let us explain the main differences w.r.t. the concrete **tccp** semantics defined in [3]. First, observe that in both semantics, each transition consumes a time instant, and that all agents are completely synchronized (rule (7) in Figure 1). The abstract version of agent A is denoted by A^α in this figure. The main points of the abstract semantics are the new **ask!** agent and the use of the two abstract entailment relations. There is one completely new rule (4), which defines the semantics for the instantaneous choice agent (**ask!**), which is similar to **ask** $^\alpha$ except for it does not consume time. The new agent is used to correctly abstract agent **now** as shown below in Figure 2. In addition, in order to simulate suspension in the abstract semantics, when a configuration containing an **ask** agent suspends in the concrete semantics, the corresponding abstract configuration is replicated in the new abstract semantics (rule (3)).

We give a first step towards a source-to-source transformation of **tccp** programs into the corresponding abstract programs. For each **tccp** agent A , we inductively construct a corresponding abstract **tccp** $^\alpha$ agent $\alpha(A)$ as is shown in Figure 2.

Stop agent. $\alpha(\text{stop}) = \text{stop}^\alpha$.
Tell agent. $\alpha(\text{tell}(c)) = \text{tell}^\alpha(c)$.
Choice agent. $\alpha(\sum_{i=0}^n \text{ask}(c_i) \rightarrow A_i) = \sum_{i=0}^n \text{ask}^\alpha(c_i) \rightarrow \alpha(A_i)$.
Conditional agent. $\alpha(\text{now } c \text{ then } A \text{ else } B) =$ $\text{now}^\alpha c \text{ then } \alpha(A) \text{ else } \text{ask}!(c) \rightarrow \alpha(A)$ $+ \text{ask}!(\text{true}) \rightarrow \alpha(B)$
Parallel agent. $\alpha(A B) = \alpha(A) \alpha(B)$.
Hiding agent. $\alpha(\exists x A) = \exists x \alpha(A)$, where x is a variable.
Procedure Call agent. $\alpha(p(x)) = p(x)$ where x is a variable of the constraint system and there exists a declaration $p(x) :- A$.

Figure 2: α -transformation for **tccp** programs

3.2 Correctness

Given a **tccp** program $P = D.\Gamma_0$, a *trace* t of P starting at $\langle \Gamma_0, st_0 \rangle$ is a sequence of configurations $t = \langle \Gamma_0, st_0 \rangle \longrightarrow \dots$ which is built by applying the transition relation rules \longrightarrow defined in [3]. Let $\mathcal{O}(P)(\langle \Gamma_0, st_0 \rangle)$ denote the corresponding standard operational semantics. We say that a concrete trace $t = \langle \Gamma_0, st_0 \rangle \longrightarrow \dots \in \mathcal{O}(P)(\langle \Gamma_0, st_0 \rangle)$ is *erroneous* iff $\exists i \geq 0. st_i$ is not consistent, that is, iff $\exists i \geq 0. st_i \vdash \text{false}$.

Similarly, given an abstraction ρ , let $\mathcal{A}_\rho(P^\alpha)(\langle \Gamma_0, sst_0 \rangle)$ denote the set of abstract traces generated by the abstract program P^α by using the abstract operational semantics given by Figure 1.

Given a trace $t = \langle \Gamma_0, st_0 \rangle \longrightarrow \dots \in \mathcal{O}(P)(\langle \Gamma_0, st_0 \rangle)$, we denote with $\alpha(t)$ the abstract trace obtained by point-wise applying the transformation α presented previously (Figure 2) to the agents in the configurations of t , and abstracting the corresponding stores using ρ ; that is, $\alpha(t) = \langle \alpha(\Gamma_0), \rho(\{st_0\}) \rangle \longrightarrow_\alpha \dots$. Given two abstract traces of the form $t_1^\alpha = \langle \Gamma_0^\alpha, sst_{01} \rangle \longrightarrow_\alpha \dots$ and $t_2^\alpha = \langle \Gamma_0^\alpha, sst_{02} \rangle \longrightarrow_\alpha \dots$, we write $t_1^\alpha \sqsubseteq t_2^\alpha$ whenever $sst_{i1} \subseteq sst_{i2}$, for all $i \geq 0$.

Theorem 1 *Given a **tccp** program $P = D.\Gamma_0$, and an abstraction ρ , for each non-erroneous trace $t \in \mathcal{O}(P)(\langle \Gamma_0, st_0 \rangle)$, there exists an abstract trace $t^\alpha \in \mathcal{A}_\rho(P^\alpha)(\langle \alpha(\Gamma_0), \rho(\{st_0\}) \rangle)$ such that $\alpha(t) \sqsubseteq t^\alpha$.*

4 Abstracting properties

In order to check temporal properties in the abstract model, we need to provide a suitable

approximation for them.

The syntax of the temporal logic of [4] is

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

The truth value of temporal formulae is defined with respect to a sequence of constraints s and the constraint system $\langle \mathcal{C}, \vdash \rangle$. Each element in the sequence represents the store at a time instant. Given a sequence $s = c_0 \cdot c_1 \cdots$, for all $i \geq 0$, we define $s^i = c_i \cdot c_{i+1} \cdots$. Following [4, 5], given temporal formulas ϕ , ϕ_1 and ϕ_2 , the satisfaction relation \models is defined below where $\exists_x s$ means $\exists_x c_0 \cdot \exists_x c_1 \cdots$:

- (1) $s^i \models c$ iff $c_i \vdash c$
- (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 for some $k \geq i$. $s^k \models \phi_2$ and for all $i \leq j < k$. $s^j \models \phi_1$

The temporal logic defined above is parameterized w.r.t. the underlying constraint system $\langle \mathcal{C}, \vdash \rangle$. Given an abstraction ρ , and using the systems $\langle \Theta, \vdash_\rho^+ \rangle$ and $\langle \Theta, \vdash_\rho^- \rangle$ given in Section 3, we may also define two abstract relations \models_ρ^+ and \models_ρ^- which, respectively, over- and under-approximate \models .

The main difficulty in abstracting the satisfiability relation \models is in dealing with the satisfiability of negated formulae (case (2) above). In fact, we may prove the following result which basically means that in order to check a negated formula, we have to interchange the abstract satisfaction relations:

$$\begin{aligned} s^\alpha \models_\rho^+ \neg\phi &\iff s^\alpha \not\models_\rho^- \phi \\ s^\alpha \models_\rho^- \neg\phi &\iff s^\alpha \models_\rho^+ \phi \end{aligned}$$

Theorem 2 proves the preservation results of our abstract model-checking methodology.

Theorem 2 *Given a tccp program $P = D.\Gamma_0$, an abstraction ρ , and a temporal formula ϕ :*

1. *If $\alpha(P) \models_\rho^- \phi$ then $P \models \phi$.*
2. *If $\alpha(P) \not\models_\rho^+ \phi$ then $P \models \neg\phi$.*

5 Conclusions

For space reasons, in this work we have presented only some results of [1, 2]. In particular, we have shown the refined abstract model-checking methodology that mitigates the state explosion problem in tccp model checking. We

have proved its correctness and we have developed an approximation technique for checking the satisfiability of the temporal properties that must be verified, which completes our methodology. As future work, an implementation of the framework proposed has already started, and we expect to get some feedback from the experiments that will enable further improvements in our method.

References

- [1] M. Alpuente, M.M. Gallardo, E. Pimentel, and A. Villanueva. Abstract Model Checking of tccp programs. In *Proc. of the 2nd Ws. on Quantitative Aspects of Prog. Lang., ENTCS* 112:19–36, 2004.
- [2] M. Alpuente, M.M. Gallardo, E. Pimentel, and A. Villanueva. A Semantic Framework for the Abstract Model Checking of tccp programs. *Theoretical Computer Science*, 2005. To appear.
- [3] F.S. de Boer, M. Gabbrielli, and M.C. Meo. A Timed Concurrent Constraint Language. *Information and Computation*, 161:45–83, 2000.
- [4] 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'l Symp. on Temporal Representation and Reasoning*, pages 227–233. IEEE, 2001.
- [5] F.S. de Boer, M. Gabbrielli, and M.C. Meo. Proving Correctness of Timed Concurrent Constraint Programs. *ACM Transactions on Computational Logic*, 5(4), 2004.
- [6] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM Int'l Symp. POPL*, pages 238–252, 1977.
- [7] M.M. Gallardo, P. Merino, and E. Pimentel. A generalized semantics of promela for abstract model checking. *Formal Aspects of Computing*, 16, 166–193, (2004).
- [8] V. A. Saraswat. *Concurrent Constraint Programming Languages*. The MIT Press, 1993.
- [9] V.A. Saraswat, R. Jagadeesan, and V. Gupta. Foundations of Timed Concurrent Constraint Programming. In *Proc. 9th Annual IEEE Symp. on Logic in Computer Science*, pages 71–80, 1994.
- [10] E. Zaffanella, R. Giacobazzi, and G. Levi. Abstracting Synchronization in Concurrent Constraint Programming. *J. of Functional and Logic Programming*, (6), 1997.