A Simulation Tool for *tccp* Programs*

María-del-Mar Gallardo Leticia Lavado
Laura Panizo
Universidad de Málaga, Andalucía Tech, Dept. Lenguajes y Ciencias de la Computación, España.
[gallardo,leticialavmu,laurapanizo]@lcc.uma.es

The Timed Concurrent Constraint Language *tccp* is a declarative synchronous concurrent language, particularly suitable for modelling reactive systems. In *tccp*, agents communicate and synchronise through a global constraint store. It supports a notion of discrete time that allows all non-blocked agents to proceed with their execution simultaneously.

In this paper, we present a modular architecture for the simulation of *tccp* programs. The tool comprises three main components. First, a set of basic abstract instructions able to model the *tccp* agent behaviour, the memory model needed to manage the active agents and the state of the store during the execution. Second, the agent interpreter that executes the instructions of the current agent iteratively and calculates the new agents to be executed at the next time instant. Finally, the constraint solver components which are the modules that deal with constraints. In this paper, we describe the implementation of these components and present an example of a real system modelled in *tccp*.

The full version of this paper [4] has been published in the proceedings of the 24th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2016).

Key Words: Timed Concurrent Constraint Language (*tccp*), Simulation tool, Abstract *tccp* instructions

1 Introduction

It is well known that many critical applications in different domains, such as health [13], railways [10] or automotive [8] have a reactive and concurrent behaviour that is difficult to model and analyse. Unfortunately, certain errors in these applications may have highly negative consequences and, therefore, it is essential to detect failures in software in the early design phases. This is why most modelling languages for these complex systems are supported by simulation and verification tools that guarantee the software's safety and reliability with respect to the critical properties.

Several formalisms have been developed to solve the problem of describing and analysing concurrent systems. In this paper, we focus on the Concurrent Constraint Paradigm (*ccp*) [11] characterised by the use of store-as-constraint instead of the classical store-as-value paradigm. Specifically, *tccp* [3] is a language suitable for describing reactive systems within this paradigm. As opposed to the interleaving composition of processes supported by most concurrent modelling languages, *tccp* makes use of the synchronous composition of processes. Synchronous languages have proved to be very useful for modelling hardware and software systems. Some successful examples are *Lustre* [7] or *SIGNAL* [5]. The synchronous management of processes clearly simplifies the scheduling tasks, although it complicates the memory use. The declarative and synchronous character of *tccp* makes it particularly suitable not only for describing but also for analysing complex concurrent systems.

There are a few tools for *tccp* proposed in the literature [9, 12]. In this paper, we present a modular framework for *tccp* with the aim of overcoming the lack of simulation and analysis tools. Classically, the

*This work has been supported by the Andalusian Excellence Project P11-TIC7659.

Submitted to:
PROLE 2017

implementation of logic languages has been based on the definition of the so-called *abstract machines* which provide an abstraction layer on the ultimate device that will execute the programs. Warren Abstract Machine (WAM) [2] is the first and most well-known abstract machine for logic languages. In the context of concurrent logic languages, there exist other proposals such as the abstract machine based on the construction of an AND/OR tree for the implementation of *Parlog* [6], or the Parallel Inference Machine (PMI) for language *KL1* [14].

We have built a simulation tool for *tccp* programs following the abstract machine philosophy but with some differences derived from the special features of the language[1]. The construction of a tool for executing *tccp* implies dealing with its declarative, constraint-based and synchronous character. For instance, the logic and concurrent nature of *tccp* involves the creation of a large number of fine-grain agents with a well-delimited variable scope. In addition, the use of constraints as data makes the integration of constraint solvers in the tool necessary. Furthermore, to correctly deal with the synchronisation, all agents executing in parallel must have a consistent view of the global memory (called *store* in *tccp*).

The implemented tool comprises different components to successfully solve the aforementioned problems. The core of the tool is formed by a set of abstract instructions and a memory model that is able to represent the state of the *tccp* program (that is, the current agent and the state of the global store) during the execution. In addition, the tool includes an interpreter that executes the current active agent iteratively. Finally, there is a module with the constraint solvers used to manage the basic operations on the global store correctly. In this paper, we describe all these components, their implementation and evaluation with a typical *tccp* example.

2 Conclusions and Future Work

We have presented an abstract machine for *tccp*, which defines the behaviour of *tccp* agents over a memory architecture called *store*. The abstract machine is composed of different modules which have been design to be as independent as possible. Most of the architecture components are unaware of the actual implementation of the memory or the particular implementation of the agent behaviour. We think that this approach facilitates and simplifies the development of tools for *tccp*. In addition, we have implemented a tool for the simulation of *tccp* following this abstract machine architecture. The tool has been implemented in *Java*, and uses other external libraries and frameworks to implement different elements. For example, we use ANTLR to generate the parsers, and PPL to implement the constraint solver for linear constraints.

We have evaluated the simulator with the photocopier example, running different number of abstract machine steps. We have presented the state of the abstract machine store after executing the example, and shown some performance measures obtained with profiling tools. We believe that the performance is acceptable, although we should improve the memory model to achieve more efficient implementations for constructing, for instance, a *tccp* model checker.

Although, the current tool at `http://morse.uma.es/tools/tccp` may be only used to simulate some available *tccp* codes, we plan to extend its capability by allowing users to simulate their own programs. In fact, the tool only lacks a frontend that manages syntax errors. In addition, due to the different implementation approaches followed by tool [9] and ours, it is difficult to compare performance of both tools but we plan to do it in the near future.

As future work, we wish to extend the abstract machine to Hy-*tccp* [1]. Hy-*tccp* is an extension of *tccp* for hybrid systems, which adds a notion of continuous time and new agents to describe the contin-

[1] The prototype tool can be found at `http://morse.uma.es/tools/tccp`.

uous dynamics of hybrid systems. Hy-*tccp* is independent from the kind of constraints over continuous variables. To implement a Hy-*tccp* simulator, we will assume that the hybrid systems are rectangular. Because of the independence amongst the different entities which compose implementation, the extension of the current abstract machine will involve adding the new agents of the Hy-*tccp* language and probably new abstract machine instructions. In addition, the parser should be extended to recognise the new agents. Finally, we will reuse PPL as the constraint solver for constraints over continuous variables.

References

[1] D. Adalid, M. Gallardo & L. Titolo (2015): *Modeling Hybrid Systems in the Concurrent Constraint Paradigm*. Electronic Proceedings in Theoretical Computer Science 173, doi:10.4204/EPTCS.173.1.

[2] H. Aït-Kaci (1991): *Warren's Abstract Machine: A Tutorial Reconstruction*. MIT Press, Cambridge, MA, USA.

[3] F. de Boer, M. Gabbrielli & M. Meo (2000): *A Timed Concurrent Constraint Language*. Information and Computation 161(1), pp. 45 – 83, doi:10.1006/inco.1999.2879.

[4] M. Gallardo, L. Lavado & L. Panizo (2017): *A Simulation Tool for tccp Programs*. In S. Schwarz & J. Voigtländer, editors: *Proceedings 29th and 30th Workshops on (Constraint) Logic Programming and 24th International Workshop on Functional and (Constraint) Logic Programming, and 24th International Workshop on Functional and (Constraint) Logic Programming, WLP 2015 / WLP 2016 / WFLP 2016, Dresden and Leipzig, Germany, 22nd September 2015 and 12-14th September 2016.*, EPTCS 234, pp. 120–134, doi:10.4204/EPTCS.234.9. Available at `http://dx.doi.org/10.4204/EPTCS.234.9`.

[5] T. Gautier & P. Le Guernic (1987): *SIGNAL: A declarative language for synchronous programming of real-time systems*. In G. Kahn, editor: *FPCA*, Lecture Notes in Computer Science 274, Springer, pp. 257–277, doi:10.1007/3-540-18317-5_15.

[6] S. Gregory, I.T. Foster, A.D. Burt & G.A. Ringwood (1989): *An abstract machine for the implementation of PARLOG on uniprocessors*. New Generation Computing 6(4), pp. 389–420, doi:10.1007/BF03037448.

[7] N. Halbwachs, P. Caspi, P. Raymond & D. Pilaud (1991): *The synchronous data flow programming language LUSTRE*. Proceedings of the IEEE 79(9), pp. 1305–1320, doi:10.1109/5.97300.

[8] E.Y. Kang, G. Perrouin & P.Y. Schobbens (2013): *Model-Based Verification of Energy-Aware Real-Time Automotive Systems*. In: Engineering of Complex Computer Systems (ICECCS), 2013 18th International Conference on, pp. 135–144, doi:10.1109/ICECCS.2013.27.

[9] A. Lescaylle & A. Villanueva (2009): *The tccp Interpreter*. Electronic Notes in Theoretical Computer Science 258(1), pp. 63 – 77, doi:10.1016/j.entcs.2009.12.005.

[10] J. Qian, J. Liu, X. Chen & J. Sun (2015): *Modeling and Verification of Zone Controller: The SCADE Experience in China's Railway Systems*. In: Complex Faults and Failures in Large Software Systems (COUFLESS), 2015 IEEE/ACM 1st International Workshop on, pp. 48–54, doi:10.1109/COUFLESS.2015.15.

[11] V.A. Saraswat & M. Rinard (1990): *Concurrent Constraint Programming*. In: Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '90, ACM, New York, NY, USA, pp. 232–245, doi:10.1145/96709.96733.

[12] T. Sjöland, E. Klintskog & S. Haridi (2001): *An interpreter for Timed Concurrent Constraints in Mozart (Extended Abstract)*.

[13] L.A. Tuan, M.C. Zheng & Q.T. Tho (2010): *Modeling and Verification of Safety Critical Systems: A Case Study on Pacemaker*. In: Secure Software Integration and Reliability Improvement (SSIRI), 2010 Fourth International Conference on, pp. 23–32, doi:10.1109/SSIRI.2010.28.

[14] K. Ueda & T. Chikayama (1990): *Design of the Kernel Language for the Parallel Inference Machine*. The Computer Journal 33(6), pp. 494–500, doi:10.1093/comjnl/33.6.494.