Universal Timed Concurrent Constraint Programming

Carlos Olarte[1,3], Catuscia Palamidessi[1], and Frank Valencia[2]

[1] INRIA Futurs, LIX, École Polytechnique, France.
[2] CNRS LIX, École Polytechnique, France.
[3] Department of Computer Science, Javeriana University Cali, Colombia.
{carlos.olarte, catuscia, frank.valencia}@lix.polytechnique.fr.

Abstract In this doctoral work we aim at developing a rich timed concurrent constraint (tcc) based language with strong ties to logic. The new calculus called *Universal Timed Concurrent Constraint* (utcc) increases the expressiveness of tcc languages allowing infinite behaviour and mobility. We introduce a constructor of the form (**abs** $x, c)P$ (*Abstraction* in P) that can be viewed as a dual operator of the hidden operator **local** x **in** P. i.e. the later can be viewed as an existential quantification on the variable x and the former as an universal quantification of x, executing $P[t/x]$ for all t s.t. the current store entails $c[t/x]$. As a compelling application, we applied this calculus to verify security protocols.

1 Introduction

Concurrent Constraint Programming (ccp) [3] is a well-established and mature model for concurrency with several reasoning techniques and strong ties to logic. ccp agents can alternatively be viewed as logic formulae, algebraic terms and computational processes. ccp is based on a monotonic shared-memory model and parametric in an information system. Processes interact by communicating through the shared store posting new constraints (**tell**(c) operator) or testing the structure of the store (**ask** c **then** P) for synchronisation purposes.

Timed Concurrent Constraint (tcc) [2] is a temporal extension of ccp aimed at specifying reactive systems. In tcc time is conceptually divided into discrete intervals and computation occurs in bursts of activity. When an stimulus (i.e. a constraint) is received from the environment, a tcc process is executed with that constraint as the initial store. When the resting point is reached, the environment can observe the store produced and a residual process is computed to be executed in the next time interval. As is shown in [2], tcc programs can be compiled into finite state automata.

Motivated in models for the analysis of security protocols where it is necessary to deal with the unbounded capabilities of the spy, in this doctoral work we are interested in increasing the expressiveness of tcc by adding two distinguished capabilities: (1) ability to express infinite behavior and (2) mobility. (1) will allow us to model complex systems such as those emerging e.g. in systemic biology and security and (2) will lead us to a name passing discipline in the tcc model. We have demonstrated that this new language is Turing complete.

2 An universal binder (Abstractions)

utcc is a derived language from tcc adding a new construct for process *abstraction*. This construct takes the form $(\mathbf{abs}\ x, c)\ P$ where intuitively $P[t/x]$ is executed for every possible term t s.t. the current store can entail the constraint $c[t/x]$. This operator is dual w.r.t. the hiding operator $(\mathbf{local}\ x\ , c)P$ where the former can be viewed as *forall* x *s.t.* $c(x)$ *do* P and the latter as *there exists* x *s.t.* $c(x)$ *and* P.

Formalising this new construct has challenging technical problems. In tcc, operational semantics requires that processes quiesce in a finite number of internal reductions to guarantee *instantaneous responses*[2]. Nevertheless, abstractions can easily generate infinite behaviour within a time unit. For example, consider the ability of composing messages posted in the network, i.e. given two messages m_1 and m_2, the spy can build a new compounded message $\{m_1, m_2\}$. An abstraction modelling this fact could be $(\mathbf{abs}\ x,\ out(x))(\mathbf{abs}\ y,\ out(y))out(\{x, y\})$ where *out* is an uninterpreted predicate in the constraint system. Given the output of the messages m_1 and m_2, this process generates a new one $(out(\{m_1, m_2\}))$ and with this, a new reduction can take place producing $out(\{m_1, \{m_1, m_2\}\})$ and so on. Thus the resting point will never be reached.

Inspired in works such as [1], we propose a symbolic semantics for utcc able to compute in a single symbolic step a possible infinite number of internal reductions in the operational semantics. The key point in this approach is to find a constraint representing the possible infinite number of constraints generated by reductions in the operational semantics.

We believe that utcc has much to offer to the concurrency theory community. In particular, to reason about security protocols. The underlying assumptions of utcc are reminiscent of those process calculi used for security. The protocols can be represented in a declarative way and reasoned about using the techniques utcc enjoys. Namely, operational, symbolic and denotational semantics. Furthermore, utcc allows for verification of reachability properties using a proof system based on Linear Temporal Logic.

References

1. M. Boreale. Symbolic trace analysis of cryptographic protocols. *Lecture Notes in Computer Science*, 2076, 2001.
2. V. Saraswat, R. Jagadeesan, and V. Gupta. Foundations of timed concurrent constraint programming. In Samson Abramsky, editor, *Proceedings of the 9th Annual IEEE Symp. on Logic in Computer Science, LICS*, 1994.
3. V.A. Saraswat, M. Rinard, and P. Panangadnen. Semantic foundation of Concurrent Constraint Programming. In *Proc. of 18th Annual ACM Symp. on Principles of Programming Languages*. ACM, 1991.