A Logical Encoding of Timed $\pi$-Calculus

Neda Saeedloei

Department of Computer Science, University of Minnesota Duluth, Duluth, USA
nsaeedlo@d.umn.edu

Abstract. We develop a logical encoding of the operational semantics of timed $\pi$-calculus: a real-time extension of Milner’s $\pi$-calculus. This executable encoding is based on Horn logical semantics of programming languages and directly leads to an implementation for timed $\pi$-calculus. This implementation can be used for modeling and verification of real-time systems and cyber-physical.

1 Introduction

In previous work [17], we extended the $\pi$-calculus [14] with real time by adding clocks and assigning time-stamps to actions. The resulting formalism, timed $\pi$-calculus, provides a simple and novel way to annotate transition rules of $\pi$-calculus with timing constraints. The timed $\pi$-calculus provides a framework for describing systems whose components interact with each other under time constraints. It contains an algebraic language for describing processes in terms of the communication actions they can perform. The timed $\pi$-calculus can model mobility, concurrency and message exchange between processes as well as infinite computation (through the infinite replication operator ‘!’), while taking into account the time constraints imposed on the actions. Therefore, it is suitable for modeling real-time systems and cyber-physical systems (CPS) [7,11] and support reasoning about their behavior related to time.

Our extension of $\pi$-calculus with time unlike most of other approaches [2,4,5,12], represents time faithfully as a continuous quantity; in other words, it does not discretize time. Discretizing means that time is represented through finite time intervals. As a result, infinitesimally small time intervals cannot be represented or reasoned about in these approaches. In practical real-time systems, e.g., a nuclear reactor, two or more events can occur within an infinitesimally small interval. Discretizing time can miss the modeling of such behavior which may be wholly contained within this infinitesimally small interval. Some other approaches for extending $\pi$-calculus with time e.g., the work of Chen [3] miss out the replication operator of the original $\pi$-calculus. Therefore, they are unable to model infinite processes. In our approach the infinite behavior of processes is modeled through the infinite replication operator ‘!’.

We also developed an operational semantics as well as a notion of timed bisimilarity for the timed $\pi$-calculus and we investigated the properties of timed bisimilarity; in particular, expansion theorem for real-time, concurrent, mobile processes [17].

© Springer International Publishing Switzerland 2014
DOI: 10.1007/978-3-319-14125-1_10
In this paper, we show how an executable operational semantics of timed π-calculus can be elegantly realized through coinductive constraint logic programming extended with coroutines. In our implementation of timed π-calculus concurrency is modeled by coroutines, and (rational) infinite computation in presence of constraints by coinductive constraint logic programming over reals (Co-CLP) [16]. The executable semantics faithfully captures real-time behaviors and allows us to prove behavioral and timing properties of a system modeled in timed π-calculus.

The work of Gupta et al. [21,22] showed how Horn logical semantics and partial evaluation can be used to generate provably correct code. From the Horn logical semantic description of the language $L$, one immediately obtains an interpreter of language $L$. In this paper, we apply the approach of [21,22] to our timed π-calculus to obtain an implementation of this language. First, we express the syntax of timed π-calculus in the Definite Clause Grammar (DCG) notation. This syntax specification trivially and naturally yields an executable parser for timed π-calculus. This parser can be used to parse timed π-calculus expressions and obtain their parse trees. Next, we express the semantic algebra and valuation functions of timed π-calculus in logic programming. The syntax and semantics specifications of timed π-calculus loaded into a coinductive constraint logic programming system directly leads to an interpreter for timed π-calculus. This interpreter can be executed and used for verifying properties of systems expressed as timed π-calculus processes. We illustrate our approach by applying it to the rail road crossing problem of Lynch and Heitmeyer [8].

Note that there is a past work on logic based implementation of the operational semantics of π-calculus, but not timed π-calculus [23]. However, this work is different from our work as it is unable to model infinite processes and infinite replication. In our implementation we are using coinductive logic programming, a more recently developed concept, which allows such modeling. Also our implementation is based on using Horn logic semantics.

2 Timed π-Calculus

Design decisions. Timed π-calculus [17] is an extension of the original π-calculus [14] with (local) clocks, clock operations and time-stamps. As in π-calculus, timed π-calculus processes use names (including clock names) to interact, and pass names to one another. We assume an infinite set $\mathcal{N}$ of names (channel names and names passing through channels), an infinite set $\mathcal{G}$ of clock names (disjoint from $\mathcal{N}$) and an infinite set $\Theta$ of variables representing time-stamps (disjoint from $\mathcal{N}$ and $\mathcal{G}$). When a process outputs a name through a channel, it also sends the time-stamp of the name and the clock that is used to generate the time-stamp. Thus, messages are represented by triples of the form $\langle m,t_m,c \rangle$, where $m$ is a name in $\mathcal{N}$, $t_m$ is the time-stamp on $m$, and $c$ is the clock that is used to generate $t_m$.

All the clocks are local clocks; however, their scope grows as they are sent among processes. Note that all the clocks advance at the same rate. At any