Theoretical Computer Science 611 (2016) 1–49

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Theory of interaction

Yuxi Fu

BASICS, Department of Computer Science, Shanghai Jiaotong University, 800 Dong Chuan Road, Shanghai 200240, China

ARTICLE INFO

Article history:
Received 10 April 2012
Received in revised form 7 March 2015
Accepted 25 July 2015
Available online 29 July 2015
Communicated by P.-A. Mellies

Keywords:
Process calculus
Bisimulation
Interaction
Computation

ABSTRACT

Theory of Interaction aims to provide a foundational framework for computation and interaction. It proposes four fundamental principles that characterize the common features of all models of computation and interaction. These principles suffice to support a model independent treatment of the two most important relationships in computer science, the equality between processes and the relative expressiveness between models. Based on the two relationships the theory of equality, the theory of expressiveness and the theory of completeness are developed.

1. Foundation

Modern computing is all about interaction [102]. This is however not to say that the traditional notion of computing is not about interaction. The difference is due to the angle of observation. In Church–Turing's models of computation, the focus is on the closed systems, systems that never interact with any other systems, and the internal actions of the systems. So the models of computation are concerned with interactions within. On the other hand, the interest of the distributed computing and the mobile computing is in the open systems. In addition to its internal interactions, an open system interacts with another open system. The complexity of an open system is often caused by the interleavings of its internal interactions with its external actions and the nondeterministic timings of these interleavings.

Models of computation were proposed and studied in 1930's. Some well known models are the Recursive Function Model, the Turing Machine Model, the λ-calculus, and the Random Access Machine [129,75,13,153]. By investigating the comparative power of these models, it was soon realized that all these models are equivalent in the sense that the partial functions definable in these models are all the same. The belief that all models of computation are equivalent in this sense has been referred to as Church–Turing Thesis. The original formulation of Church–Turing Thesis emphasizes on computability. It was revealed in subsequent studies, especially in the studies of algorithms [31] and computational complexity [113, 165,8], that the equivalence of the models of computation actually holds at the operational level. There is a translation from one model of computation to another that preserves and reflects computations. The translation is not only effective, it is actually efficient.

Milner [96] pioneered the study to formalize the notion of interaction between open systems (processes) by his work on CCS. At about the same time, Hoare [70] has also made landmark contribution to the theory in his work on the well known programming language CSP. Since then the theory of process calculus in particular and the concurrency theory in general have proliferated. In fact the development of concurrency theory has been so fast and the number of the proposed models has increased so dramatically that a call for a general theory of interaction has long been overdue. Looking back,

E-mail address: fu-yx@cs.sjtu.edu.cn.

http://dx.doi.org/10.1016/j.tcs.2015.07.043
0304-3975/

one cannot help remarking that the fundamental notions like interaction, composition, localization, and interface (channel) were all present in the very beginning in CCS as well as CSP. A crucial tool, bisimulation, was introduced to concurrency by Park [114] and Milner [99] in 1980's, which has greatly advanced the observation theory of processes ever since. An interesting account of the history of bisimulation in computer science can be found in [139].

An issue that has attracted more attention than ever recently is the relative expressiveness of process calculi. In view of the hundreds of process calculi [108], if not more, that have been proposed, results on expressiveness are rare. The issue is complicated by the lack of a consensus on the criteria for comparing the expressiveness. As Parrow [116] has pointed out, if we have i process calculi and j sets of criteria, we end up with $i \times j \times i$ positive or negative results on expressiveness. Worse still there may well be contradictory results since an expressiveness relationship from one calculus to another may be negative by one set of criteria and positive by another set of criteria. The truth is, as long as we are using two different sets of criteria, contradictory results will never be a surprise. This is definitely unacceptable for a theory that seeks to reveal the law of interaction. One source of the diversity is that too many process calculi have been manufactured with little industrial effort. The lack of constraints has led to a profusion of calculi that would fail any single set of criteria. If we are to look for a single set of sensible criteria applicable to some models, we will be at the same time ruling out a good few of others.

Another important issue is about what Abramsky [2] has called expressive completeness. Should we impose a minimal expressive power on all models of interaction? A positive answer would be a very foundational assumption that provides a fundamental constraint. If a process calculus is intended to be an extension of a computation model, the expressive completeness should imply that within the calculus one should be able to code up all the computable functions. So in a unified theory for both computation models and interaction models, the expressive completeness should subsume the so-called Turing completeness discussed in literature. But how should we formalize the expressive completeness? Let's take a look at how Turing completeness is formalized in the theory of process calculus. Again different criteria have been proposed in literature [90]. The proof of Turing completeness according to these criteria typically amounts to constructing a map $[\![\_]\!]$ from the set of the computing objects of a computation model $\mathbf{L}$ to a set of processes of a process calculus $\mathbf{M}$ in such a way that a computation of say L is simulated by an internal action sequence of $[\![L]\!]$ and vice versa. There is nothing wrong with most Turing completeness proofs in process calculus literature. But since these proofs are for interaction models, one should really see them from the perspective of interaction. Some of the equalities used in the proofs are not even trace equivalent. Consequently these Turing completeness results are not really useful for the interactive processes. Another conspicuous deficiency of most proofs of Turing completeness for process calculi is the lack of something like "value supply" or "result delivery". The input values are not picked up from environments properly, and the results cannot be delivered safely to any targeted receivers. Again this level of completeness cannot be promoted to interaction level.

One could give more examples demonstrating the lack of consensus, not to mention philosophies, in other parts of the theory of process calculus. But the point has been made. As much as it has benefited from the observational approach that tries to ignore all computations, the theory of process calculus has suffered from not paying enough attention to the fundamental role of the theory of computation. The theory of process calculus and the theory of computation have been two separate developments. A sensible thing to do is to carry out an integrated study that does not favor one over the other. It is expected that both theories should benefit from such an integration.

Before we formally develop the Theory of Interaction, we have to point out that the motivation for this work is mainly theoretical. We aim to lay down a framework that is as model independent as possible and at the same time allows one to prove significant general results. For that purpose we have to make some choices along the way. Firstly we will adopt the one-one synchronous communication as the basic interaction pattern. For two processes to interact they must be connected to the two ends of a channel. There are other communication patterns like one-many communication and CSP style synchronization that are not covered by our framework. Secondly we will focus exclusively on interleaving semantics. Some of our criteria do not apply in a noninterleaving semantics. Thirdly we assume that there are a couple of operators that are universal in the sense that these operators are present in all the models we will be considering. There are models that instead of having all the universal operators, have some kind of combined versions of these operators whose semantics is model specific. A study of an even more general framework that can cover models like some variants of CSP, Join Calculus, Distributed π and the Ambient Calculus (see for example [36,35,60,29]) is beyond the scope of this paper.

1.1. Theory of interaction

The prime motivation for Theory of Interaction is to bridge the gap between the computation theory and the interaction theory. By adopting the view that interaction is computation seen from within and that computation is interaction seen from without, Theory of Interaction eliminates the distinction between these two kinds of models from the outset. The benefit is that we may extend the results and methodologies well established in the computation theory to models that account for both computation and interaction.

The ultimate goal of Theory of Interaction is to provide a framework in which one may formalize the foundational assumptions, for example the Church–Turing Thesis, widely accepted in the major branches of computer science. Since postulates are formulated in terms of relations, one has to pin down the fundamental relationships in computer science before formalizing any postulates. These relationships must be about models and objects (processes, programs), there is nothing more basic than these two classes of entities, hence the following two relationships.