Information and Computation 239 (2014) 254–302

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Linear logical relations and observational equivalences for session-based concurrency

Jorge A. Pérez [a,*], Luís Caires [b], Frank Pfenning [c], Bernardo Toninho [b,c]

[a] Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, Netherlands
[b] CITI, NOVA LINCS and Departamento de Informática, FCT Universidade Nova de Lisboa, Portugal
[c] Computer Science Department, Carnegie Mellon University, United States

ARTICLE INFO

Article history:
Received 12 September 2013
Received in revised form 22 April 2014
Available online 11 August 2014

Keywords:
Session types
Linear logic
Process calculi
Strong normalization
Confluence
Logical relations
Observational equivalences

ABSTRACT

We investigate *strong normalization*, *confluence*, and *behavioral equality* in the realm of session-based concurrency. These interrelated issues underpin advanced correctness analysis in models of structured communications. The starting point for our study is an interpretation of linear logic propositions as session types for communicating processes, proposed in prior work. Strong normalization and confluence are established by developing a theory of *logical relations*. Defined upon a linear type structure, our logical relations remain remarkably similar to those for functional languages. We also introduce a natural notion of *observational equivalence* for session-typed processes. Strong normalization and confluence come in handy in the associated coinductive reasoning: as applications, we prove that all *proof conversions* induced by the logic interpretation actually express observational equivalences, and explain how *type isomorphisms* resulting from linear logic equivalences are realized by coercions between interface types of session-based concurrent systems.

1. Introduction

Modern computing systems rely heavily on the *communication* between distributed software artifacts. Hence, to a large extent, guaranteeing system correctness amounts to ensuring consistent dialogues between such artifacts. This is a challenging task, given the complex interaction patterns that communicating systems usually feature. *Session-based concurrency* provides a foundational approach to communication correctness: concurrent dialogues are structured into basic units called *sessions*; descriptions of the interaction patterns are then abstracted as *session types* [22,23,17], which are statically checked against specifications. These specifications are usually given in the π-calculus [31,42], and so we obtain *processes* interacting on so-called session channels which connect exactly two subsystems. The discipline of session types ensures protocols in which actions always occur in dual pairs: when one partner sends, the other receives; when one partner offers a selection, the other chooses; when a session terminates, no further interaction may occur. New sessions may be dynamically created by invoking shared servers. While *concurrency* arises in the simultaneous execution of sessions, *mobility* results from the exchange of session and server names.

The goal of this paper is to investigate *strong normalization*, *confluence*, and *typed behavioral equivalences* for session-typed, communicating processes. These interrelated issues underpin advanced correctness analysis in models of structured

* Corresponding author.
E-mail addresses: j.a.perez@rug.nl (J.A. Pérez), lcaires@fct.unl.pt (L. Caires), fp@cs.cmu.edu (F. Pfenning), btoninho@cs.cmu.edu (B. Toninho).

http://dx.doi.org/10.1016/j.ic.2014.08.001
0890-5401/

communications. Our study builds upon the interpretation of linear logic propositions as session types put forward by Caires and Pfenning in [11]. In a concurrent setting, such an interpretation defines a tight propositions-as-types/proofs-as-programs correspondence, in the style of the Curry–Howard isomorphism for the simply-typed λ-calculus [24]. In the interpretation, types (linear logic propositions) are assigned to names (communication channels) and describe their session protocol; typing rules correspond to linear sequent calculus proof rules and processes correspond to proof objects in logic derivations. Moreover, process reduction may be simulated by proof conversions and reductions, and vice versa. As a result, typed processes enjoy strong forms of *subject reduction* (type preservation) and *global progress* (deadlock-freedom). While the former states that well-typed processes always evolve to well-typed processes (a *safety* property), the latter says that well-typed processes will never get into a stuck state (a *liveness* property). These strong correctness properties make the framework in [11] a convenient basis for our study of strong normalization, confluence, and behavioral equivalences. Well-studied in sequential settings, these three interrelated issues constitute substantial challenges for theories of communicating processes, as we motivate next.

In typed functional calculi, *strong normalization* ensures that well-typed terms do not have infinite reduction sequences. Types rule out divergent computations; termination of reduction entails consistency of the corresponding logical systems. In the realm of communicating processes, reduction captures atomic synchronization; associated behavioral types exclude unintended structured interactions. As a result, strong normalization acquires an enhanced significance in a concurrent setting. In fact, even if subject reduction and progress are typically regarded as the key correctness guarantees for processes, requiring strongly normalizing behaviors is also most sensible: while from a global perspective systems are meant to run forever, at a local level we wish responsive participants which always react within a finite amount of time, and never engage into infinite internal behavior. For instance, in server-client applications it is critical for clients to be sure that running code provided by the server will not cause them to get stuck indefinitely (as in a denial-of-service attack, or just due to some bug).

Closely related to strong normalization, *confluence* is another appealing property. In a communication-based setting, confluence would strengthen correctness guarantees by ensuring *predictability* of structured interactions. This benefit may be more concretely seen by considering the principle of *typed process composition* derived from the logic interpretation. In [11], typing judgments specify both the session behavior that a process *offers* (or *implements*) and the set of (unrestricted and linear) behaviors that it *requires* to do so. For instance, the judgment

$$u{:}B\,;\, x_1{:}A_1, \ldots, x_n{:}A_n \vdash P :: z{:}C \tag{1}$$

specifies a process P which offers behavior C along name z by making use of an *unrestricted* behavior B (a replicated server, available on name u) and of *linear* behaviors $A_1, \ldots, A_n$ (offered on names $x_1, \ldots, x_n$). A process implementing one of these linear dependencies could be specified by the judgment

$$\cdot\,;\, \cdot \vdash S_1 :: x_1{:}A_1 \tag{2}$$

which says that process S_1 does not depend on any linear or unrestricted session behaviors to offer behavior A_1 along name x_1. We write '$\cdot$' to denote the empty set of dependencies. Given a typed interface such as (1), to satisfy each of the declared behavioral dependencies we need to first (i) compose the given process with another one which realizes the required behavior, and then (ii) restrict the name in which the behavior is required/offered, to avoid interferences. As a result, the interactions between the given process and the processes implementing its dependencies are unobservable. In the case of (1) and (2) above we would obtain the following typed composition:

$$u{:}B\,;\, x_2{:}A_2, \ldots, x_n{:}A_n \vdash (\boldsymbol{\nu} x_1)(S_1 \mid P) :: z{:}C \tag{3}$$

Hence, interactions on name x_1 become unobservable in the resulting composed process; its set of dependencies combines those of P (excepting $x_1{:}A_1$) and those of S_1 (in this case, the empty set). From (3) we could proceed similarly for all the behaviors declared in the left-hand side, thus obtaining a typed process without dependencies:

$$\cdot\,;\, \cdot \vdash (\boldsymbol{\nu}\widetilde{m})\big(!u(y).R \mid S_1 \mid \cdots \mid S_n \mid P\big) :: z{:}C \tag{4}$$

with $\widetilde{m} = u, x_1, \ldots, x_n$ and $\cdot\,;\, \cdot \vdash R :: y{:}B$. In the above process, all behavioral dependencies arise as internal reductions; the only visible behavior takes place on name z. Notice that processes $R, S_1, \ldots, S_n$ may well have internal behavior on their own. For processes such as the one in (4), the interplay of confluence with strong normalization would be significant, as it could crucially ensure that session behavior as declared by judgments in the right-hand side ($z{:}C$ in this case) will be always offered, independently from any arbitrary interleaving of internal reductions from different sources.

Now, in sharp contrast to the normalizing, confluent nature of computation in many typed functional calculi, process calculi are inherently non-terminating, non-confluent models of concurrent computation. Hence, unsurprisingly, ensuring strong normalization and confluence in calculi for concurrency is a hard problem: in (variants of) the π-calculus, proofs require heavy constraints on the language and/or its types, often relying on ad-hoc machineries (see [15] for a survey on termination in process calculi). As a first challenge, we wonder: building upon our linear type structure, directly obtained from the Curry–Howard correspondence in [11], can we establish useful forms of strong normalization and confluence for session-typed, communicating processes?