Theoretical Computer Science 585 (2015) 71–90

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Program equivalence in linear contexts

Yuxin Deng [a,1], Yu Zhang [b,c,*,2]

[a] Department of Computer Science and Engineering, Shanghai Jiao Tong University, China
[b] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[c] Zhuhai College, Jilin University, China

ARTICLE INFO

Article history:
Available online 6 March 2015

Keywords:
Linear PCF
Contextual equivalence
Trace equivalence
Non-determinism

ABSTRACT

Program equivalence in linear contexts, where programs are used or executed *exactly once*, is an important issue in programming languages. However, existing techniques like those based on bisimulations and logical relations only target at contextual equivalence in the usual (non-linear) functional languages, and fail in capturing non-trivial equivalent programs in linear contexts, particularly when non-determinism is present.

We propose the notion of *linear contextual equivalence* to formally characterize such program equivalence, as well as a novel and general approach to studying it in higher-order languages, based on labeled transition systems specifically designed for functional languages. We show that linear contextual equivalence indeed coincides with trace equivalence. We illustrate our technique in both deterministic (a linear version of PCF) and non-deterministic (linear PCF in Moggi's framework) functional languages.

1. Introduction

Contextual equivalence is an important concept in programming languages and can be used to formalize and reason about many interesting properties of computing systems. For functional languages, there are many techniques that can help to prove contextual equivalence. Among others, applicative bisimulations [1,17] and logical relations [29,32] are particularly successful.

On the other side, linear logic (and its term correspondence often known as linear λ-calculus) has seen significant applications in computer science ever since its birth, due to its native mechanism of describing restricted use of resources. For example, the linear λ-calculus provides the core of a functional programming language with an expressive type system, in which statements like "this resource will be used exactly once" can be formally expressed and checked. Such properties become useful when introducing imperative concepts into functional programming [15], structural complexity theory [16], or analyzing memory allocation [34]. Moreover, the linear λ-calculus, when equipped with dependent types, can serve as a representation language within a logical framework, a general meta-language for the formalization of deductive systems [7]. The study of linearity in concurrent languages such as the π-calculus allows for a fine-grained analysis of process behavior [20,36,4].

Introducing linearity also leads to novel observation over program equivalences. In particular, if we consider a special sort of contexts where candidate programs must be used linearly (we call these contexts *linear contexts*), program equivalence

* Corresponding author.
E-mail addresses: deng-yx@cs.sjtu.edu.cn (Y. Deng), yzhang@ios.ac.cn (Y. Zhang).

[1] Partially supported by the National Natural Science Foundation of China (61173033, 61261130589) and ANR 12IS02001 "PACE".
[2] Partially supported by the National Natural Science Foundation of China (61100063, 61161130530).

http://dx.doi.org/10.1016/j.tcs.2015.03.006
0304-3975/

with respect to these contexts should be a coarser relation than the usual notion of contextual equivalence, especially when non-determinism is present. For instance, take Moggi's language for non-determinism [23], where we have a primitive $\sqcap$ for non-deterministic choice (same as the internal choice in CSP [14]), and consider the following two functions:

$$\begin{aligned} f_1 &\stackrel{\text{def}}{=} \mathtt{val}(\lambda x.\mathtt{val}(0) \sqcap \mathtt{val}(1)) \\ f_2 &\stackrel{\text{def}}{=} \mathtt{val}(\lambda x.\mathtt{val}(0)) \sqcap \mathtt{val}(\lambda x.\mathtt{val}(1)). \end{aligned} \tag{1}$$

Existing techniques such as bisimulation or logical relations distinguish these two functions. In fact, it is easy to show that they are not equivalent in arbitrary contexts, by considering, e.g., the context

$$\mathtt{bind}\ f = [\_]\ \mathtt{in}\ \mathtt{bind}\ x = f(0)\ \mathtt{in}\ \mathtt{bind}\ y = f(0)\ \mathtt{in}\ \mathtt{val}(x = y).$$

The context makes a double evaluation of the function by applying it to concrete arguments (noticing that Moggi's language enforces a call-by-value evaluation of non-deterministic computations): with the first function f_1, the two evaluations of $f(0)$ can return different values since the non-deterministic choice is inside the function body; with the second function f_2, the non-deterministic choice is made before both evaluations of $f(0)$ and computation inside the function is deterministic, so the two evaluations always return the same value. But if we consider only linear contexts, where programs will be evaluated *exactly once*, then the two functions must be equivalent. However, no existing technique, at least to the best of our knowledge, can characterize such an equivalence relation with respect to linear contexts.

1.1. Potential application in computational cryptography

The motivation of the work first comes from the second author's work on building a logic (namely CSLR) for reasoning about *computational indistinguishability*, which is an essential concept in complexity-theoretic cryptography and helps to define many important security criteria [24,37,10]. The CSLR logic is based on a functional language which characterizes probabilistic polynomial-time computations by typing, where linearity plays an important role.

Let us consider an example. The *semantic security* is a fundamental property of encryption schemes and in CSLR we can define it as

$$\lambda\eta\,.\,\lambda m_0\,.\,\lambda m_1\,.\,\boldsymbol{Enc}(\eta, m_0, pk) \simeq_C \lambda\eta\,.\,\lambda m_0\,.\,\lambda m_1\,.\,\boldsymbol{Enc}(\eta, m_1, pk) \tag{2}$$

where ***Enc*** is the (probabilistic) encryption function, pk is a public key known to adversaries, and η is the security parameter. The equivalence relation $\simeq_C$ is what we call *computationally indistinguishability* in cryptography and the detailed definition can be found in [37, Definition 1]. Intuitively, it says that no feasible adversary can distinguish the related programs with non-negligible probabilities (w.r.t. the programs' security parameter). Formula (2) formally states the indistinguishability between two encryption oracles: an adversary can only do encryption (with the key pk) by calling the oracles, who will ask the adversary to supply a pair of messages at his/her choice and return the encryption of one of them — the left oracle always returns the cipher-text of the first message and the right one returns the cipher-text of the second. Computational indistinguishability between the two oracles says that no feasible adversary, when calling one of the two oracle, can answer which message is encrypted with a significant winning probability.

Computational indistinguishability is indeed a notion of observational equivalence, where adversaries must be computable in polynomial-time on probabilistic Turing machines. Complexity is well manipulated in the CSLR type system, where linearity particularly plays the essential role of controlling complexity in higher-order functions. The original definition indicates that the two oracles as defined in (2) can be called only for a polynomial number of times, however, Theorem 5.2.11 in [11] shows that the definition where an adversary only submits a single pair of messages to the oracle, is equivalent to the multi-messages version. It suggests that we can effectively replace the computational indistinguishability by the notion of linear contextual equivalence in (2) and consider an adversary who calls the oracle *only once*.

In fact, we believe that linear contextual equivalence can be used to define security criteria with *non-adaptive* adversaries. Non-adaptive adversaries send multiple messages to oracles but choose messages independently of the oracles' responses, so we can actually define the oracles as functions receiving a list of messages. Adversaries will also send their messages to the oracle *all at once* and in that case they call the oracle only *once*.

Although the language of the CSLR logic is probabilistic, a general proof technique of linear contextual equivalence is missing from the literature, particularly in a non-deterministic setting where there exist programs that are equivalent in linear contexts but not in general, as we described previously.

1.2. Related work

Program equivalence with respect to *non-linear* contexts has been widely investigated. Logical relations are one of the powerful tools for proving contextual equivalence in typed lambda-calculi, in both operational [26,27,6] and denotational settings [29,22,13]. They are defined by induction on types, hence are relatively easy to use. But it is known that completeness of (strict) logical relations are often hard to achieve, especially for higher-order types. It is even worse for monadic types, particularly when non-determinism is present [21].