Indexed linear logic and higher-order model checking

Charles Grellois
ENS Cachan and University Paris Diderot
grellois@pps.univ-paris-diderot.fr

Paul-André Melliès
CNRS and University Paris Diderot
mellies@pps.univ-paris-diderot.fr

In recent work, Kobayashi observed that the acceptance by an alternating tree automaton $\mathscr{A}$ of an infinite tree $\mathscr{T}$ generated by a higher-order recursion scheme $\mathscr{G}$ may be formulated as the typability of the recursion scheme $\mathscr{G}$ in an appropriate intersection type system associated to the automaton $\mathscr{A}$. The purpose of this article is to establish a clean connection between this line of work and Bucciarelli and Ehrhard's indexed linear logic. This is achieved in two steps. First, we recast Kobayashi's result in an equivalent infinitary intersection type system where intersection is not idempotent anymore. Then, we show that the resulting type system is a fragment of an infinitary version of Bucciarelli and Ehrhard's indexed linear logic. While this work is very preliminary and does not integrate key ingredients of higher-order model-checking like priorities, it reveals an interesting and promising connection between higher-order model checking and linear logic.

1 Introduction

Model-checking is a well-established technique in formal verification, based on the following model-theoretic procedure. In order to decide whether a given program P satisfies a property φ of interest, one interprets the program P into an appropriate model and translates the property φ into an equivalent automaton $\mathscr{A}$. The fact that the program P satisfies the property φ is then reduced to the existence of a successful run of the automaton $\mathscr{A}$ over the interpretation of the program P in the model, which is decidable. In the specific case of higher-order model checking, a higher-order program P is modelled as a *higher-order recursion scheme* (HORS) which generates the tree of all its possible behaviours. Recall that given a signature Σ and a set of variables $\mathscr{V}$, a higher-order recursion scheme $\mathscr{G} = \langle \Sigma, \mathscr{N}, \mathscr{R}, S \rangle$ consists of a set of simply-typed non-terminals $\mathscr{N}$, of an axiom $S \in \mathscr{N}$ of type o, and of a set of equations (or rewriting rules) of the form

$$F \quad = \quad \lambda x_1 \cdots \lambda x_n.\, t \qquad (\text{denoted } \mathscr{R}(F))$$

where t is a term of base type o and $F \in \mathscr{N}$ has simple type $\sigma_1 \to \cdots \to \sigma_n \to o$. One requires moreover that there is exactly one such equation per non-terminal, and that the simple types of F and of $\mathscr{R}(F)$ coincide. Every such recursion scheme $\mathscr{G}$ may be thus seen as a term of the simply typed λ-calculus with fixpoint operator Y. By definition, the *order* of a recursion scheme is the maximal order of its non-terminal's simple type. An example of an order-2 scheme over the signature $\Sigma = \{ a : 2 , b : 1 , c : 0 \}$ is

$$\begin{array}{lcl} S & = & F\ c \\ F & = & \lambda x.\, a\ x\ (F\ (b\ x)) \end{array} \tag{1}$$

The labelled and ranked tree generated by the recursion scheme $\mathscr{G}$ is called the *value tree* of the scheme. It is computed by application of these rules starting from the axiom. In our illustration, the value tree of the recursion scheme (1) depicted in Figure 1 is obtained as the limit of the rewriting sequence :

$$S \quad \longrightarrow \quad F\ c \quad \longrightarrow \quad a\ c\ (F\ (b\ c)) \quad \longrightarrow \quad a\ c\ (a\ (b\ c)\ F\ (b\ b\ c)) \quad \longrightarrow \quad \cdots$$

Jakob Rehof (Ed.): Intersection Types and Related Systems (ITRS)
EPTCS 177, 2015, pp. 43–52, doi:10.4204/EPTCS.177.4

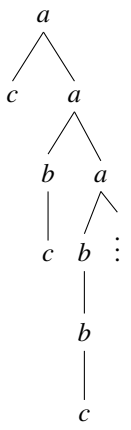

Figure 1: An order-2 tree.

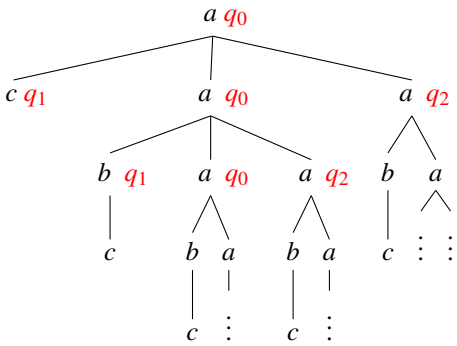

Figure 2: An alternating run-tree.

The decidability of monadic second order logic (MSO) over the value trees computed by higher-order recursion schemes was established for the first time by Ong [10] using game semantics. Other proofs of the same decidability result were then elaborated, either based on collapsible pushdown automata [7], on an intersection type system [9] or on Krivine machines [12]. All these proofs are based on the reduction of the decidability of MSO to the decidability of the modal μ-calculus, which is in turn equivalent to the existence of a winning run-tree of an alternating parity automaton $\mathscr{A}$ on the value tree of the higher-order recursion scheme $\mathscr{G}$. Recall that an *alternating* tree automaton proceeds in the same way as a usual top-down tree automaton, but with an additional ability: at each step of its exploration, the automaton can decide to explore a given subtree of the current node several times, or not at all. Since each of these explorations may be seen as occurring on a different copy of the same subtree, everything thus works as if the alternating automaton duplicates the subtree the number of times it explores it. By way of illustration, keeping the same signature $\Sigma = \{a:2, b:1, c:0\}$ as before, a typical transition will duplicate the rightmost child of a node labelled $a \in \Sigma$ and explore the first copy with state q_0 and the second copy with state q_2:

$$\delta(q_0, a) \;=\; (1, q_1) \wedge (2, q_0) \wedge (2, q_2) \tag{2}$$

In particular, when applied to the value-tree of the recursion scheme $\mathscr{G}$ in Figure 1, the transition induces a run-tree whose upper nodes are depicted in Figure 2. The starting point of this work is the observation of an apparent similarity between this ability of alternating automata to duplicate a tree in order to explore it several times and the duplication mechanisms associated to the exponential modality of linear logic. In order to clarify this tentative connection between linear logic and higher-order model checking, we start from the type-theoretic account of alternating parity automata by Kobayashi and Ong [9]. For simplicity, we prefer to restrict ourselves to Kobayashi's work [8] and do not consider priorities (or parity conditions) at this stage. By doing so, we restrict the expressivity of the logic to safety properties. A treatment of priorities would be possible however, along the lines of our recent observation [6] that priorities behave in just the same comonadic way as the exponential modality of linear logic.

Plan of the paper

We start by recalling in §2 the intersection type system originally considered by Kobayashi [8]. The first contribution of the paper is to establish in §3 a correspondence theorem between this type system

and a quantitative variant of Kobayashi's type system where intersection is not idempotent anymore. This preliminary steps leads us to establish in §4 that the quantitative intersection type system is a full fragment of an infinitary version of Bucciarelli and Ehrhard's indexed linear logic [1, 2].

2 Intersection types and alternating tree automata

The type-theoretic account of higher-order model-checking initiated by Kobayashi [8] is based on the idea that a transition like (2) may be reflected by giving to the symbol $a \in \Sigma$, in addition to its simple type $o \to o \to o$, the refined intersection type $q_1 \to (q_0 \wedge q_2) \to q_0$. In this approach, every such symbol $a \in \Sigma$ of the signature will generally have several such refined types, each of them derived from the transitions $\delta(q,a)$ associated to each state $q \in Q$ of the underlying alternating automaton $\mathscr{A}$. Note that the existence of an accepting alternating run-tree over the value tree of a recursion scheme $\mathscr{G}$ involves infinite objects, whose structure can be very complex — observe in particular that the order-2 tree of Figure 1 is not regular. In that respect, the type-theoretic account of the tree automaton $\mathscr{A}$ has one main benefit: the refined types defined on the symbols a, b, c of the signature Σ may be lifted to every simply-typed λ-term appearing in the recursion scheme $\mathscr{G}$. By way of illustration, consider again the recursion scheme (1) and assume that the alternating automaton $\mathscr{A}$ has the additional transitions

$$\delta(q_0, b) \;=\; (1, q_2) \qquad\qquad \delta(q_2, b) \;=\; (1, q_0) \wedge (1, q_1).$$

In this situation, the symbol b of simple type $o \to o$ is given the refined types $q_2 \to q_0$ and $(q_0 \wedge q_1) \to q_2$ and one can thus type the term $\mathscr{R}(F)$ in the following way:

$$\lambda x.\, a\; x\; (F\; (b\; x)) \quad : \quad (q_0 \wedge q_1 \wedge q_2) \to q_0$$

under the assumption that the non-terminal F of simple type $o \to o$ has the refined types $q_0 \to q_0$ and $q_2 \to q_2$. From this lifting property, Kobayashi [8] deduces a decision procedure for the existence of a run-tree of the alternating automaton $\mathscr{A}$ over the value-tree of the recursion scheme $\mathscr{G}$. Here, we consider the intersection type system of [8] as it is recently rephrased by Ong and Tsukada [11]. We thus define refined pre-types as follows:

$$\textit{Refined pre-types} \qquad \sigma, \tau \quad ::= \quad q \;\mid\; \tau \to \sigma \;\mid\; \bigwedge_{j \in J} \tau_j$$

Note that in this system refined types have to match the shape of simple types. To ensure this, denoting σ a refined type and κ a simple type[1], we introduce the proper refinement relation $\sigma :: \kappa$, defined by the following rules:

$$\frac{}{\vdash q :: o} \qquad\qquad \frac{\vdash \tau_j :: \kappa \quad (\text{for all } j \in J) \qquad \vdash \sigma :: \kappa'}{\vdash \bigwedge_{j \in J} \tau_j \to \sigma :: \kappa \to \kappa'}$$

A *refined type* is a refined pre-type which properly refines a simple type. A sequent is of the form

$$x_1 : \tau_1 :: \kappa_1 \,, \ldots, x_n : \tau_n :: \kappa_n \vdash M : \sigma :: \kappa$$

where the context

$$\Gamma \quad = \quad x_1 : \tau_1 :: \kappa_1 \,, \ldots, x_n : \tau_n :: \kappa_n$$

is a sequence of different variables, each of them typed by a refined type and by the simple type it refines. The rules of the system are given in Figure 3.

[1] In Kobayashi's original article [8], simple types were called *kinds*, and the word "type" was reserved to what we call here refined types.

$$\text{Axiom}\quad \frac{\vdash\ \tau_j :: \kappa \quad (\text{for all } j \in J)}{\Gamma, x : \bigwedge_{j\in J} \tau_j :: \kappa\ \vdash\ x : \tau_i :: \kappa} \qquad i \in J$$

$$\text{Application}\quad \frac{\Gamma \vdash M : (\bigwedge_{j\in J} \tau_j) \to \sigma :: \kappa \to \kappa' \qquad \Gamma \vdash N : \tau_j :: \kappa \quad (\text{for all } j \in J)}{\Gamma \vdash MN : \sigma :: \kappa'}$$

$$\text{Lambda}\quad \frac{\Gamma, x : \bigwedge_{j\in J} \tau_j :: \kappa\ \vdash\ M : \sigma :: \kappa'}{\Gamma \vdash \lambda x.M : (\bigwedge_{j\in J} \tau_j) \to \sigma :: \kappa \to \kappa'}$$

Figure 3: Kobayashi's intersection type system.

The decidability result requires that the set of refinement types σ which refine a given simple type κ remains finite. To that purpose, intersection is required to be *idempotent* in Kobayashi's type system. This idempotency property may be neatly formulated by requiring that intersections are stable under *surjective reindexing* $f : J \twoheadrightarrow I$ in the sense that

$$\bigwedge_{j\in J} \sigma_{f(j)} \quad = \quad \bigwedge_{i\in I} \sigma_i$$

for every family $\{\sigma_i \,|\, i \in I\}$ of refinement types indexed by a finite set I. Note in particular that the expected equation

$$\sigma \wedge \sigma \quad = \quad \sigma$$

follows from the consideration of the surjective reindexing $\{1,2\} \to \{1\}$. At this stage, we make the following observation:

Lemma 1. *If $\Gamma \vdash t : \sigma :: \kappa$ in Kobayashi's system and t η-expands to t', then $\Gamma \vdash t' : \sigma :: \kappa$.*

In other words, typability is preserved by η-expansion in Kobayashi's type system. For that reason, we will only consider $\beta\eta$-long normal form λ-terms in the sequel.

3 A quantitative variant of the original type system

Seen from the point of view of linear logic, Kobayashi's type system appears as a variant of natural deduction based on an additive translation of intuitionistic logic. In order to prepare the forthcoming connection with indexed linear logic, we turn it into a sequent calculus based this time on a multiplicative translation of intuitionistic logic. This leads us to the system of Figure 4, where the finite intersections are no longer required to be stable under surjective reindexing. Note that we do not even require any associativity or commutativity condition on the intersection: in particular, the intersections are not even stable under *bijective* reindexing.

We say that a variable x occurs *linearly* in a context Γ when the variable x has refined type q or $\bigwedge_{i\in I} \tau_i \to \sigma$ in the context Γ. In other words, the variable x is declared linear when the principal connective of its type is not an intersection. We say that the variable x occurs *linearly* in a λ-term t when there exists a unique occurence of the variable x in t.

Lemma 2. *If $\Gamma \vdash t : \sigma :: \kappa$ and x occurs linearly in Γ, then x occurs linearly in t.*

An important consequence of the lemma is that the variable x occurs linearly in the λ-term M considered in the Left $\to$ rule. From this follows that the Left $\to$ rule which transforms M into the λ-term $M[x := f\ N]$ introduces exactly one application node $f\ N$ in the λ-term M.

$$\text{Axiom} \qquad \frac{q \in Q}{x : q :: o \vdash x : q :: o}$$

$$\text{Left} \bigwedge \qquad \frac{\Gamma, x : \tau :: \kappa \vdash M : \sigma :: \kappa'}{\Gamma, x : \bigwedge_{j \in \{i\}} \tau_j :: \kappa \vdash M : \sigma :: \kappa'} \qquad i \in \mathbb{N},\ \tau_i = \tau$$

$$\text{Right} \bigwedge \qquad \frac{\Gamma_j \vdash M : \tau_j :: \kappa \quad (\text{for all } j \in J)}{\bigwedge_{j \in J} \Gamma_j \vdash M : \bigwedge_{j \in J} \tau_j :: \kappa}$$

$$\text{Left} \to \qquad \frac{\Gamma_1 \vdash N : \bigwedge_{j \in J} \tau_j :: \kappa \qquad \Gamma_2, x : \sigma :: \kappa' \vdash M : \alpha :: \kappa''}{\Gamma_1, \Gamma_2, f : \left(\bigwedge_{j \in J} \tau_j\right) \to \sigma :: \kappa \to \kappa' \vdash M[x := f\ N] : \alpha :: \kappa''} \qquad \sigma \text{ linear}$$

$$\text{Right} \to \qquad \frac{\Gamma, x : \bigwedge_{j \in J} \tau_j :: \kappa \vdash M : \sigma :: \kappa'}{\Gamma \vdash \lambda x.M : \left(\bigwedge_{j \in J} \tau_j\right) \to \sigma :: \kappa \to \kappa'}$$

$$\text{Weakening} \qquad \frac{\Gamma \vdash M : \sigma :: \kappa'}{\Gamma, x : \bigwedge_{j \in \emptyset} \tau_j :: \kappa \vdash M : \sigma :: \kappa'} \qquad (x \notin \Gamma)$$

$$\text{Contraction} \qquad \frac{\Gamma, y : \bigwedge_{i \in I} \tau_i :: \kappa, z : \bigwedge_{j \in J} \tau_j :: \kappa \vdash M : \sigma :: \kappa'}{\Gamma, x : \bigwedge_{k \in I \uplus J} \tau_k :: \kappa \vdash M[y, z := x] : \sigma :: \kappa'} \qquad (x \notin \Gamma)$$

Figure 4: The quantitative intersection type system.

Both intersection type systems formulated in Figure 3 and in Figure 4 are designed to simulate an alternating automaton $\mathscr{A}$ exploring the value-tree of a higher-order recursion scheme $\mathscr{A}$. One main difference comes from the fact that in Kobayashi's system the multiplicity of usage of a given state is not tracked, so that a function using its argument twice with refined type q_0 in order to answer a request q_1 may actually be typed $q_0 \wedge q_2 \to q_1$. This is impossible in the quantitative system, where the Weakening rule only introduces an intersection indexed by the empty family. In order to formalize a precise connection between the type systems, we thus need an appropriate notion of order over qualitative refined types (that is, the idempotent refined types of Kobayashi's system). This notion of order is precisely the one of the Scott lattice model of linear logic, see for instance [15, 5]:

- If $\sigma :: o$ and $\tau :: o$, then $\sigma \preccurlyeq \tau$ if and only if $\sigma = \tau$.
- Define $\bigwedge_{i \in I} \sigma_i \to \tau \preccurlyeq \bigwedge_{j \in J} \sigma'_j \to \tau'$ if and only if both types refine a same simple type $\kappa \to \kappa'$, $\tau \preccurlyeq \tau'$ and $\forall i \in I\ \ \exists j \in J \quad \sigma'_j \preccurlyeq \sigma_i$.

Given a quantitative type σ, define its *collapse* $|\sigma|$ as the qualitative type canonically obtained by assuming stability by surjective reindexing. This operation is extended to contexts in the standard way. We may now give a precise description of the connection between the two type systems. We will see in the next section that the quantitative type system is in fact designed to reflect the relational semantics of linear logic, this correspondence theorem may be seen as a type-theoretic transcription of Ehrhard's recent collapse theorem [5] between the relational semantics of linear logic and its Scott lattice model.

Theorem 1. *Every derivation tree of one system may be effectively translated in the other either by lifting qualitative types or by collapsing quantitative types:*

- *If* $\Gamma \vdash t : \sigma : \kappa$ *in the quantitative system, then* $|\Gamma| \vdash t : |\sigma| :: \kappa$ *in Kobayashi's system.*
- *If* $x_1 : \sigma_1 :: \kappa_1, \ldots, x_n : \sigma_n :: \kappa_n \vdash t : \tau : \kappa$ *in Kobayashi's system, there exists quantitative types* $\hat{\sigma}_i$ $(1 \leq i \leq n)$ *and* $\hat{\tau}$ *such that*

– $\forall i \in \{1,\dots,n\}\ \ \hat{\sigma}_i :: \kappa_i$ *and* $\hat{\tau} :: \kappa$,
– $\forall i \in \{1,\dots,n\}\ \ |\hat{\sigma}_i| \preccurlyeq \sigma_i$ *and* $|\hat{\tau}| \preccurlyeq \tau$,
– $x_1 : \hat{\sigma}_1 :: \kappa_1, \dots, x_n : \hat{\sigma}_n :: \kappa_n \vdash t : \hat{\tau} : \kappa$ *in the quantitative system.*

4 A linear interpretation of intersection types

In this section, we give an alternative formulation of the fragment of Bucciarelli and Ehrhard's indexed linear logic necessary to interpret general continuations, and thus higher-order recursion schemes. The restriction of Bucciarelli and Ehrhard's indexed linear logic [1, 2] to this specific fragment enables us to annotate every proof of the logic with a simply-typed λ-term. This leads us to a formulation of indexed linear logic in the style of de Carvalho [3]. We then explain how to translate Kobayashi's type system into the resulting fragment of indexed linear logic.

4.1 Indexed Linear Calculus

Every formula of indexed linear logic is indexed by a countable indexing set J. As already mentioned, we will focus on the fragment of the logic corresponding to general continuations, whose formulas are generated by the following grammar:

$$\begin{array}{lcll} \textit{Linear pre-formulas} & A,B & ::= & \bot_J \ \mid\ S \multimap A \\ \textit{Replicable pre-formulas} & S,T & ::= & !_u A \end{array}$$

where J is any countable set, and where $u : J \to K$ is any function between the two countable indexing sets J and K. Every linear or replicable formula of indexed linear logic is defined as a pre-formula obtained by a series of application of the rules below.

$$\frac{}{\vdash_J \bot_J} \qquad \frac{\vdash_J S \qquad \vdash_J A}{\vdash_J S \multimap A} \qquad \frac{\vdash_J A}{\vdash_K !_u A} \quad u : J \to K$$

Quite obviously, there exists for every linear formula A a unique countable indexing set J such that $\vdash_J A$. This specific countable set J is called the domain $dom(A)$ of the formula A. Now, suppose given a set $Q = \{q_1,\dots,q_n\}$ of elements, typically representing the states of an alternating automaton $\mathscr{A}$. The types of the logic are then defined by the following grammar:

$$\begin{array}{lcll} \textit{Linear pre-types} & \sigma,\tau & ::= & q \ \mid\ \vec{\varphi} \multimap \sigma \\ \textit{Compound pre-types} & \vec{\varphi},\vec{\psi} & ::= & [\sigma_j \mid j \in J] \end{array}$$

where J is any countable set of indices. A type is then defined as a pre-type which refines a specific formula of our fragment of indexed linear logic. The refinement relation is defined by structural induction, and may be formulated by the following derivation rules.

$$\frac{q_j \in Q \quad (\text{for all } j \in J)}{\vdash_{j\in J}\ q_j :: \bot_J} \qquad \frac{\vdash_{j\in J}\ \vec{\varphi}_j :: !_u A \qquad \vdash_{j\in J}\ \sigma_j :: B}{\vdash_{j\in J}\ \vec{\varphi}_j \multimap \sigma_j :: !_u A \multimap B}$$

$$\frac{\vdash_{j\in J}\ \sigma_j :: A}{\vdash_{k\in K}\ [\sigma_j \mid u(j) = k] :: !_u A} \quad u : J \to K$$

For conciseness, $\vdash_{i\in I}$ may be abbreviated $\vdash_I$. The idea behind this formulation is that a quantitative refinement type $\bigwedge_{k\in K}\ \sigma_k :: \kappa$ may be seen as an indexed type

$$\vdash_{k\in K}\ [\sigma_j \mid u(j) = k] :: !_u A \qquad u : K \to 1$$

where A is a linear formula refining the simple type κ. Indexing by $u : K \to 2$ would give two such intersection types in parallel. The type system of Indexed Linear Calculus (ILC) is described in Figure 5. In the Dereliction rule, the action u^* of a bijection $u : J \to K$ is defined by structural induction:

$$\text{Axiom} \quad \frac{q_j \in Q \quad (\text{for all } j \in J)}{x : q_j :: \bot_J \vdash_J x : q_j :: \bot_J}$$

$$\text{Right} \multimap \quad \frac{\Gamma, x : \phi_j :: !_u A \vdash_J M : \sigma_j :: B}{\Gamma \vdash_J \lambda x.M : \phi_j \multimap \sigma_j :: A \multimap B}$$

$$\text{Left} \multimap \quad \frac{\Gamma_1 \vdash_J M : \phi_j :: !_u A \qquad \Gamma_2, x : \sigma_j :: B \vdash_J N : \tau_j :: C}{\Gamma_1, \Gamma_2, f : \phi_j \multimap \sigma_j :: !_u A \multimap B \vdash_J N[x \leftarrow fM] : \tau_j :: C}$$

$$\text{Weakening} \quad \frac{\Gamma \vdash_J M : \sigma_j :: B}{\Gamma\,,\, x : [] :: !_{\mathbf{0}_J} A \vdash_J M : \sigma_j :: B} \qquad \mathbf{0}_J : \emptyset \to J$$

$$\text{Contraction} \quad \frac{\Gamma\,,\, y : \phi_j :: !_{u_1}(A|_{J_1})\,,\, z : \psi_j :: !_{u_2}(A|_{J_2}) \vdash_J M : \sigma_j :: B}{\Gamma\,,\, x : \phi_j \uplus \psi_j :: !_u A \vdash_J M[y,z \leftarrow x] : \sigma_j :: B} \qquad u = [u_1, u_2] : J_1 + J_2 \to J$$

$$\text{Dereliction} \quad \frac{u^*\Gamma\,,\, x : \sigma_j :: A \vdash_J M : \tau_{u(j)} :: u^*B}{\Gamma\,,\, x : \sigma_{u^{-1}(k)} :: !_u A \vdash_K M : \tau_k :: u^*B} \qquad u : J \to K \text{ bijective}$$

$$\text{Promotion} \quad \frac{\ldots,\ x_k : [\sigma_{i_k} \,|\, i_k \in I_k, u_k(i_k) = j] :: !_{u_k} A_k\,,\ \ldots \vdash_J M : \tau_j :: B}{\ldots,\ x_k : [\sigma_{i_k} \,|\, i_k \in I_k, v(u_k(i_k)) = l] :: !_{v \circ u_k} A_k\,,\ \ldots \vdash_L M : [\tau_j \,|\, v(j) = l] :: !_v B} \qquad v : J \to L$$

Figure 5: Indexed Linear Calculus.

- $u^*(\bot_K) \;=\; \bot_J$
- $u^*(S \multimap A) \;=\; u^*(S) \multimap u^*(A)$
- $u^*(!_v A) \;=\; !_{u^{-1} \circ v} A$

and in the Contraction rule, the domain restriction operation $A|_K$ where A is of formula of domain J is also defined by structural induction:

- $\bot_J|_K := \bot_K$
- $(S \multimap A)|_K := S|_K \multimap A|_K$
- $(!_u A)|_K = !_v(A|_{u^{-1}(K)})$ where $u : I \to J$ and $v : u^{-1}(K) \to K$ is equal to the function u restricted to the subset $u^{-1}(K) \subseteq I$.

Note that these operations were only defined on formulas. Their action on types is the expected one: for the domain restriction operation, a compound type $(\phi_j)_{j \in J}$ is restricted to $(\phi_j)_{j \in K}$, and for the other operation, the bijection naturally acts by reindexing over compound types.

4.2 Interpreting quantitative intersection types in ILC

Now translating the quantitative intersection types into indexed types is essentially immediate: given a quantitative type σ, define the corresponding indexed type $(\!|\, \sigma \,|\!)$ inductively as follows :

- $(\!|\, \bigwedge_{i \in I} \sigma_i \,|\!) \;=\; [(\!|\, \sigma_i \,|\!) \,|\, i \in I]$ (seen as a $\{\star\}$-indexed formula)
- $(\!|\, \sigma \to \tau \,|\!) \;=\; (\!|\, \sigma \,|\!) \multimap (\!|\, \tau \,|\!)$
- $(\!|\, q \,|\!) \;=\; q$

This operation can be inverted for $\{\star\}$-indexed formulas in the expected way. Given a $\{\star\}$-indexed type σ, the corresponding quantitative intersection type is denoted $\llparenthesis \sigma \rrparenthesis$.

Lifting simple types to formulas requires to index formulas properly. For example, $q_1 \wedge q_2 \to q_0$ refines the simple type $o \to o$. This lifts to the linear type $[q_1, q_2] \multimap q_0$ refining the formula $!_u \perp_2 \multimap \perp_1$ where u is the unique function $\{1,2\} \to \{1\}$. Given a formula of indexed linear logic A, we define inductively its corresponding simple type $\kappa(A)$ as follows:

- $\kappa(!_u A) = \kappa(A)$
- $\kappa(S \multimap A) = \kappa(S) \to \kappa(A)$
- $\kappa(\perp_J) = o$

Given a context $\Gamma = x_1 : \sigma_1 :: A_1, \dots, x_n : \sigma_n :: A_n$ of indexed linear calculus, define its associated quantitative intersection context $\{\!|\Gamma|\!\} = x_1 :: \llparenthesis \sigma_1 \rrparenthesis :: \kappa(A_1), \dots, x_n :: \llparenthesis \sigma_n \rrparenthesis :: \kappa(A_n)$.

Theorem 2.
- *If the sequent $\Gamma \vdash_I M : \sigma :: A$, where I is a singleton, is provable in the indexed linear calculus, then $\{\!|\Gamma|\!\} \vdash M : \llparenthesis \sigma \rrparenthesis :: \kappa(A)$ is provable in the quantitative intersection system.*
- *If the sequent $\Gamma \vdash M : \sigma :: \kappa$ is provable in the quantitative intersection system, there exists a context Γ' of the indexed linear calculus and a formula A of indexed linear logic such that $\{\!|\Gamma'|\!\} = \Gamma$ and that $\Gamma' \vdash_{\{\star\}} M : (\!| \sigma |\!) :: A$ is provable in the indexed linear calculus.*

Recall that indexation is a way to parallelize proofs with the same underlying tree, and only differing by their types labelling. This is the reason why the connection only makes sense for indexation families isomorphic to $\{\star\}$.

Remark now that the indexed linear calculus contains a lot of redundant information. Types may be computed from proof-trees, by picking the state information at the right axiom rule, and terms can be recovered from the rules of a proof-tree, since indexation ensures uniformity of terms when applying them – contrary to what occurs in the resource lambda-calculus. In fact, this indexed linear calculus captures precisely the fragment of the relational model corresponding to simply-typed λ-terms. Call *indexed tensorial logic* the system obtained by removing terms and types in the indexed linear calculus. Then derivation trees of this logic are in bijection with ILC derivation trees.

5 Related works

The starting point of this work is the observation by Kobayashi [8] that MSO model-checking over trees generated by higher-order recursion schemes can be performed by typing the scheme in an appropriate intersection type system.

Another inspiration was the recent development by Terui [15] of a semantic and type-theoretic approach based on linear logic, intersection types and automata theory in order to characterize the complexity of evaluation to the booleans in the simply-typed λ-calculus – relating on the qualitative model of Scott domains for linear logic. A quantitative account of intersection type in the relational model of linear logic was given by de Carvalho [3], also with the goal of studying complexity issues.

After Ong's seminal proof [10] of the decidability of higher-order model-checking, several other were given. One of them, by Kobayashi and Ong [9], extends Kobayashi's type system to capture all MSO. Salvati and Walukiewicz are currently developing a semantic approach to higher-order model checking, based on the interpretation of the Krivine environment machine in finite models of the λ-calculus with fixpoint operators, in order to obtain a semantic proof of this decidability result [14, 13].

Finally, Tsukada and Ong [11] introduced two-level game semantics to provide a model of Kobayashi's type system.

6 Conclusion and future work

The main purpose and contribution of the paper is to stress the tight and somewhat surprising connection between a series of recent advances in linear logic and the current type-theoretic approach to higher-order model-checking. In particular, the lucid reader will recognize that all the results stated in the present paper are essentially known to one community or to the other. However, besides the useful confrontation of two related lines of research, we believe that a tangible technical contribution of the paper is a careful proof of Theorem 2. The result is somewhat folklore and appears implicitly in the works by Bucciarelli and Ehrhard [1, 2] and by de Carvalho [3] but it was never proved (nor even formally stated) as far as the authors know. Another point: besides the connection between indexed linear logic and higher-order model checking, one main message of the paper conveyed in Theorem 1 is that the decidability results obtained in the field of higher-order model checking are to a large part regulated by the collapse theorem recently established by Ehrhard [5, 4]. We hope that the bridge with linear logic will clarify the constructions of higher-order model checking and reveal the deep and beautiful semantic ideas underlying it.

References

[1] Antonio Bucciarelli & Thomas Ehrhard (2000): *On Phase Semantics and Denotational Semantics in Multiplicative-Additive Linear Logic*. *Ann. Pure Appl. Logic* 102(3), pp. 247–282, doi:10.1016/S0168-0072(99)00040-8. Available at `http://www.pps.univ-paris-diderot.fr/~buccia/PUBLI/apal00.ps`.

[2] Antonio Bucciarelli & Thomas Ehrhard (2001): *On phase semantics and denotational semantics: the exponentials*. *Ann. Pure Appl. Logic* 109(3), pp. 205–241, doi:10.1016/S0168-0072(00)00056-7. Available at `http://www.pps.univ-paris-diderot.fr/~buccia/PUBLI/apal01.ps`.

[3] Daniel de Carvalho (2009): *Execution Time of lambda-Terms via Denotational Semantics and Intersection Types*. *CoRR* abs/0905.4251. Available at `http://arxiv.org/abs/0905.4251`.

[4] Thomas Ehrhard (2012): *Collapsing non-idempotent intersection types*. In Patrick Cégielski & Arnaud Durand, editors: *CSL*, *LIPIcs* 16, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 259–273. Available at `http://dx.doi.org/10.4230/LIPIcs.CSL.2012.259`.

[5] Thomas Ehrhard (2012): *The Scott model of linear logic is the extensional collapse of its relational model*. *Theor. Comput. Sci.* 424, pp. 20–45, doi:10.1016/j.tcs.2011.11.027. Available at `http://www.pps.univ-paris-diderot.fr/~ehrhard/pub/relscott.pdf`.

[6] Charles Grellois & Paul-André Melliès (2015): *Tensorial logic with colours and higher-order model checking*. submitted, `http://arxiv.org/abs/1501.04789`.

[7] Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong & Olivier Serre (2008): *Collapsible Pushdown Automata and Recursion Schemes*. In: *LICS*, IEEE Computer Society, pp. 452–461, doi:10.1109/LICS.2008.34. Available at `http://www.cs.ox.ac.uk/people/luke.ong/personal/publications/cpda-long.pdf`.

[8] Naoki Kobayashi (2009): *Types and higher-order recursion schemes for verification of higher-order programs*. In Zhong Shao & Benjamin C. Pierce, editors: *POPL*, ACM, pp. 416–428, doi:10.1145/1480881.1480933. Available at `http://www.kb.is.s.u-tokyo.ac.jp/~koba/papers/popl2009.pdf`.

[9] Naoki Kobayashi & C.-H. Luke Ong (2009): *A Type System Equivalent to the Modal Mu-Calculus Model Checking of Higher-Order Recursion Schemes*. In: *LICS*, IEEE Computer Society, pp. 179–188, doi:10.1109/LICS.2009.29. Available at `http://www.cs.ox.ac.uk/people/luke.ong/personal/publications/lics09-ko-long.pdf`.

[10] C.-H. Luke Ong (2006): *On Model-Checking Trees Generated by Higher-Order Recursion Schemes*. In: *LICS*, IEEE Computer Society, pp. 81–90, doi:10.1109/LICS.2006.38. Available at `http://www.cs.ox.ac.uk/people/luke.ong/personal/publications/ntrees.pdf`.

[11] C.-H. Luke Ong & Takeshi Tsukada (2012): *Two-Level Game Semantics, Intersection Types, and Recursion Schemes*. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts & Roger Wattenhofer, editors: *ICALP (2)*, *Lecture Notes in Computer Science* 7392, Springer, pp. 325–336, doi:10.1007/978-3-642-31585-5_31. Available at `http://www.cs.ox.ac.uk/people/luke.ong/personal/publications/icalp12.pdf`.

[12] Sylvain Salvati & Igor Walukiewicz (2012): *Recursive Schemes, Krivine Machines, and Collapsible Pushdown Automata*. In Alain Finkel, Jérôme Leroux & Igor Potapov, editors: *RP*, *Lecture Notes in Computer Science* 7550, Springer, pp. 6–20, doi:10.1007/978-3-642-33512-9_2.

[13] Sylvain Salvati & Igor Walukiewicz (2013): *Evaluation is MSOL-compatible*. In Anil Seth & Nisheeth K. Vishnoi, editors: *FSTTCS*, *LIPIcs* 24, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 103–114, doi:10.4230/LIPIcs.FSTTCS.2013.103. Available at `http://hal.inria.fr/docs/00/77/31/26/PDF/m.pdf`.

[14] Sylvain Salvati & Igor Walukiewicz (2013): *Using Models to Model-Check Recursive Schemes*. In Masahito Hasegawa, editor: *TLCA*, *Lecture Notes in Computer Science* 7941, Springer, pp. 189–204, doi:10.1007/978-3-642-38946-7_15. Available at `http://hal.archives-ouvertes.fr/docs/00/74/12/39/PDF/m.pdf`.

[15] Kazushige Terui (2012): *Semantic Evaluation, Intersection Types and Complexity of Simply Typed Lambda Calculus*. In Ashish Tiwari, editor: *RTA*, *LIPIcs* 15, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 323–338, doi:10.4230/LIPIcs.RTA.2012.323. Available at `http://www.kurims.kyoto-u.ac.jp/~terui/nbi8.pdf`.