Higher-order parity automata

Paul-André Melliès
CNRS and Université Paris Diderot

***Abstract*—We introduce a notion of higher-order parity automaton which extends to the infinitary simply-typed λ-calculus the traditional notion of parity tree automaton on infinitary ranked trees. Our main result is that the acceptance of an infinitary λ-term by a higher-order parity automaton $\mathcal{A}$ is decidable, whenever the infinitary λ-term is generated by a finite and simply-typed λY-term. The decidability theorem is established by combining ideas coming from denotational semantics and from infinitary rewriting theory.**

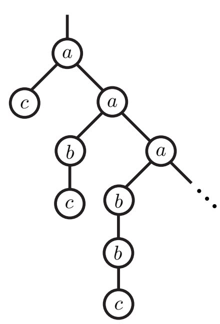
I. Introduction

A. *Higher-order model-checking on infinite trees*

Higher-order model-checking is understood today as the description of infinitary properties of ranked trees generated by higher-order recursion schemes. By way of illustration, consider the higher-order recursion scheme $\mathcal{G}$ on the signature $\Sigma = \{a : 2, b : 1, c : 0\}$

$$\mathcal{G} \;=\; \left\{ \begin{array}{lll} S & \mapsto & F\ a\ b\ c \\ F\ x\ y\ z & \mapsto & x\ z\ (F\ x\ y\ (y\ z)) \end{array} \right. \tag{1}$$

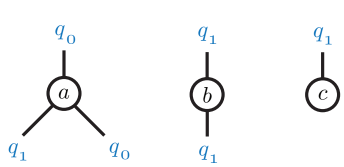
which generates the infinite tree

$\langle \mathcal{G} \rangle \quad =$  (2)

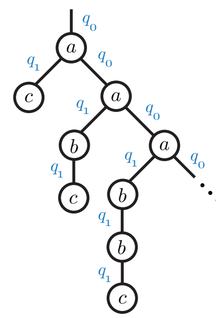
Imagine that one wants to check the following infinitary property $(*)$ on the higher-order recursion scheme $\mathcal{G}$: that the ranked tree $\langle \mathcal{G} \rangle$ it generates consists of an infinite path of letters a expanding on the right:

$$\langle \mathcal{G} \rangle = a(w_0, a(w_1, a(w_2, a(w_3, \ldots))))$$

where each subtree w_n is a finite word (= filiform tree) consisting of a finite sequence of letters b ended by a letter c. In order to answer that question using the tools of higher-order model-checking, one may for instance construct a parity tree automaton $\mathcal{A}$ with same signature Σ, with a set Q consisting of two states q_0 and q_1, with initial state q_0, with transition function δ represented as

 (3)

and with coinductive parity 2 on the state q_0 and inductive parity 1 on the state q_1. It is then not difficult to check that a higher-order recursion scheme $\mathcal{G}$ on the signature Σ satisfies the property $(*)$ if and only if the ranked tree $\langle \mathcal{G} \rangle$ it generates is accepted by the automaton $\mathcal{A}$. Typically, the higher-order recursion scheme $\mathcal{G}$ defined in (1) satisfies the property $(*)$ because there exists an accepting run-tree $\mathcal{R}$ of the parity tree automaton $\mathcal{A}$ on the ranked tree $\langle \mathcal{G} \rangle$, represented below

$\mathcal{R} \quad =$  (4)

One important observation of higher-order model-checking is that given any finite signature Σ, and

- any higher-order recursion scheme $\mathcal{G}$
- any alternating parity tree automaton $\mathcal{A}$

on that signature Σ, the problem $(**)$ whether the ranked tree $\langle \mathcal{G} \rangle$ generated by $\mathcal{G}$ is accepted by $\mathcal{A}$ is decidable. For instance, the question whether a given higher-order scheme $\mathcal{G}$ on the signature $\Sigma = \{a : 2, b : 1, c : 0\}$ generates a ranked tree $\langle \mathcal{G} \rangle$ satisfying the property $(*)$ reduces to the question $(**)$ instantiated with the parity tree automaton $\mathcal{A}$ on two states q_0, q_1 described in (3) ; from this, it follows that the property $(*)$ is decidable.

In the present paper, we will develop this idea and bring to light the notion of *higher-order parity automaton* which extends and adapts to higher-order model-checking the primary notion of *alternating tree automaton*. Our main task here will be to formulate the notion of higher-order automaton we have in mind, and to justify it by establishing a general decidability theorem, broader in scope than the traditional decidability theorem illustrated above.

B. *Higher-order arities and alphabets*

As we will see, our intended notion of higher-order automaton is designed to analyse infinitary properties of typically infinite simply-typed λ-terms. Recall that a simple type $A, B \in \mathit{Type}$ is a binary tree defined by the grammar below:

$$A, B \;:=\; \mathbb{o} \;\mid\; A \times B \;\mid\; A \Rightarrow B \;\mid\; \mathbf{1} \tag{5}$$

where $\oplus$ is a fixed ground type. A higher-order alphabet is then defined as a finite set Σ of letters together with a function

$$\mathbf{\Sigma} \ : \ \Sigma \to Type$$

which transports every letter $a \in \Sigma$ to its higher-order arity $\mathbf{\Sigma}(a) \in Type$, defined as a simple type of the λ-calculus. It is a good practice to write down such a higher-order alphabet $\mathbf{\Sigma}$ as the sequence of letters of the finite alphabet Σ, each of them attached to the specific simple type $A_k = \mathbf{\Sigma}(a_k)$ defining its arity:

$$a_1 : A_1, \ldots, a_n : A_n.$$

The letters $a_1, \ldots, a_n$ of the alphabet are all different, and the order in which they appear in the sequence does not matter. A higher-order alphabet in our sense is thus the same thing as what one traditionally calls a *context* of simply-typed variables $a_1, \ldots, a_n$ in the simply-typed λ-calculus. We choose this terminology derived from automata theory in order to prepare our definition of higher-order automaton next.

C. The Church encoding taken seriously

As a matter of fact, we will make a great use of the following dictionary between automata theory and the simply-typed λ-calculus:

higher-order alphabet	$\leftrightarrow$	simply-typed context
higher-order arity	$\leftrightarrow$	simple type
letter	$\leftrightarrow$	variable

The dictionary extends to letters $a \in \Sigma$ of arbitrary higher-order arity $\mathbf{\Sigma}(a) = A$ the familiar Church encoding of a finite ranked tree as a simply-typed λ-term. In this encoding, every letter a of arity $k \in \mathbb{N}$ in the original tree signature is translated into a letter $a \in \Sigma$ of higher-order arity

$$\mathbf{\Sigma}(a) \quad = \quad \underbrace{(\oplus \times \ldots \times \oplus)}_{k\text{-fold product}} \Rightarrow \oplus \qquad (6)$$

the arity of a k-ary function on the ground type $\oplus$. The Church encoding induces a one-to-one correspondence between

1. the ranked trees of a given signature Σ,
2. the simply-typed λ-terms in normal form of the corresponding higher-order alphabet $\mathbf{\Sigma}$.

By way of illustration, there is a one-to-one relationship between the finite trees on the signature $\Sigma = \{a : 2, b : 1, c : 0\}$ mentioned earlier in the introduction, and the simply-typed λ-terms M in normal form of higher-order alphabet $\mathbf{\Sigma}$ and type

$$a : (\oplus \times \oplus) \Rightarrow \oplus \, , \, b : \oplus \Rightarrow \oplus \, , \, c : \oplus \quad \vdash \quad M : \oplus. \qquad (7)$$

This correspondence between trees and λ-terms is also relevant to higher-order model-checking. The reason is that every higher-order recursion scheme $\mathcal{G}$ on a signature Σ may be translated into a λY-term M of higher-order alphabet $\mathbf{\Sigma}$ and of type $\oplus$, in the same way as we did for finite trees in (7). The infinitary Böhm tree $N = BT(M)$ generated by the λY-term M coincides then via the Church encoding with the infinitary ranked tree $\langle \mathcal{G} \rangle$ generated by $\mathcal{G}$. See §VI for a discussion on a definition of Böhm trees more suitable to higher-order model-checking. Typically, the λY-term M associated to the higher-order recursion scheme $\mathcal{G}$ in (1) is defined as

$$M \quad = \quad \Big(Y \big[\, \lambda F . \lambda x . \lambda y . \lambda z . \, x \, z \, (\, F \, x \, y \, (\, y \, z \,)\,) \big] \Big) \, a \, b \, c \qquad (8)$$

The infinite tree $\langle \mathcal{G} \rangle$ depicted in (2) coincides then with the infinitary Böhm tree $N = BT(M)$ of same higher-order alphabet $\mathbf{\Sigma}$ and same type $\oplus$

$$a : (\oplus \times \oplus) \Rightarrow \oplus \, , \, b : \oplus \Rightarrow \oplus \, , \, c : \oplus \quad \vdash \quad N : \oplus$$

generated by the simply-typed λY-term M.

D. Infinitary λ-terms and infinitary rewriting

At this point, we find convenient to take advantage of the foundational work on infinitary rewriting theory developed in [11], [4], and to regard the λY-term M as an infinitary simply-typed λ-term, obtained by applying the unfolding rule

$$Y \, P \ \simeq \ P \big(Y \, P \big)$$

as many times as necessary (typically a countable number of times) in order to make the fixpoint operator Y disappear. One main benefit of this infinitary point of view is that both M and the infinitary Böhm tree $N = BT(M)$ define infinitary simply-typed λ-terms. Moreover, the two infinitary λ-terms M and N are related by an infinite and strongly convergent sequence of β-rewriting steps in the sense of [11], [4], what we write:

$$f \quad : \quad M \twoheadrightarrow_{\beta} N \qquad (9)$$

In particular, the fact that M and N have the same higher-order alphabet $\mathbf{\Sigma}$ and the same type $\oplus$ may be seen as a consequence of the existence of this infinitary rewriting path f between them.

E. Higher-order automata

This formulation of the theory based on the Church encoding of infinitary trees into infinitary λ-terms reveals an intriguing limitation of higher-order model-checking: the fact that the λ-terms M and N have higher-order alphabet $\mathbf{\Sigma}$ restricted to letters of first-order arity — that is, of the specific form specified in (6). As a matter of fact, this limitation comes from the notion of tree automaton $\mathcal{A}$ itself, since it requires that the Böhm tree $N = BT(M)$ generated by M is [via the Church encoding] a ranked tree on a given signature Σ whose letters are thus of first-order arity.

This raises an interesting question: would it be possible to extend the notion of alternating parity tree automaton to an arbitrary higher-order alphabet, not limited any more to first-order letters? In short, the purpose of such a higher-order notion of automaton $\mathcal{A}$ would be to explore an infinitary simply-typed λ-term M

of arbitrary higher-order alphabet $\mathbf{\Sigma}$ and of arbitrary type A, and to tell us whether the λ-term M is accepted or rejected according to the inductive and coinductive conditions of the parities. Moreover, the automaton $\mathcal{A}$ should behave in the same way as an alternating parity tree automaton when all the letters of the alphabet $\mathbf{\Sigma}$ are of first-order arity.

We are ready at this stage to introduce the notion of higher-order parity automaton we have in mind. In its most basic acception, a higher-order automaton is defined as a tuple $\mathcal{A} = (Q, \mathbf{\Sigma}, A, \delta, q_0)$ where

- Q is a finite set of ground states,
- $\mathbf{\Sigma}$ is the higher-order alphabet of the automaton,
- A is the simple type of the automaton,
- δ is a transition function of the alphabet $\mathbf{\Sigma}$,
- q_0 is an initial state of type A.

This definition relies on the idea that every simple type A should come with its own set Q_A of higher-order states, defined by induction from the set Q of ground states:

$$Q_{\oplus} = Q \qquad\qquad Q_{A \Rightarrow B} = \mathcal{P}_{\mathrm{fin}}(Q_A) \times Q_B$$

where $\mathcal{P}_{\mathrm{fin}}(Q_A)$ denotes the set of finite subsets of Q_A. A higher-order state q of type $A \Rightarrow B$ is thus a pair of the form

$$q \quad = \quad (\, \{q_1, \ldots, q_n\}\, ,\, q'\,) \tag{10}$$

where $q' \in Q_B$ is a state of type B and each element $q_i \in Q_A$ of the finite set $\{q_1, \ldots, q_n\}$ is a state of type A. The idea of defining a higher-order state of type $A \Rightarrow B$ in this way comes from linear logic, and its linear decomposition of the intuitionistic implication

$$A \Rightarrow B \quad = \quad (\,!\,A\,) \multimap B. \tag{11}$$

into a linear implication $A \multimap B$ and $!\,A$ the exponential modality of the logic. The connection between linear logic and our notion of higher-order automaton is extremely strong. In particular, the set Q_A of higher-order states of type A coincides with the interpretation of the simple type A in a specific extensional semantics of linear logic, decomposing the Scott lattice semantics, see §III for a discussion. For that reason, we find convenient to use the notation

$$q \quad = \quad \{q_1, \ldots, q_n\} \multimap q'$$

for the higher-order state $q \in Q_{A \Rightarrow B}$ described in (10).

A higher-order automaton $\mathcal{A}$ of alphabet $\mathbf{\Sigma}$ and of type A is designed to explore an infinitary λ-term M of same alphabet $\mathbf{\Sigma}$ and of same type A. Its transition function δ associates to every letter $a_i \in \Sigma$ a finite set

$$\delta(a_i) \in \mathcal{P}_{\mathrm{fin}}(Q_{A_i})$$

of higher-order states of type $A_i = \mathbf{\Sigma}(a_i)$. Accordingly, its initial state $q_0 \in Q_A$ is a higher-order state of type A. Now, the distinctive feature of higher-order automata with respect to traditional tree automata is what happens when a higher-order automaton

$$\mathcal{A} = (Q, \mathbf{\Sigma}, A \Rightarrow B, \delta, q_0)$$

with initial state $q_0 = \{q_1, \ldots, q_n\} \multimap q$ starts exploring an infinitary λ-term $\mathbf{\Sigma} \vdash \lambda a.M \,:\, A \Rightarrow B$ of same alphabet $\mathbf{\Sigma}$ and of same type $A \Rightarrow B$, whose node λa at the root binds the letter a of arity A in the infinitary λ-term M. In that situation, the higher-order automaton $\mathcal{A}$ behaves in the following way: it starts by extending its alphabet $\mathbf{\Sigma}$ with the letter a of arity A, and its transition function δ with the assignment

$$\delta \;:\; a \;\mapsto\; \{q_1, \ldots, q_n\} \in \mathcal{P}_{\mathrm{fin}}(Q_A)$$

Now that the value of the transition function δ has been defined for the letter a, the automaton $\mathcal{A}$ may carry on and explore with initial state $q \in Q_B$ the infinitary λ-term $\mathbf{\Sigma}\,,a \,:\, A \vdash M \,:\, B$ of type B in the extended alphabet $\mathbf{\Sigma}, a \,:\, A$. By way of illustration, consider the infinitary λ-term

$$\vdash \;\; \lambda a.\lambda b.\lambda c.N \;\;:\;\; \big(((\oplus \times \oplus) \Rightarrow \oplus) \times (\oplus \Rightarrow \oplus) \times \oplus\big) \Rightarrow \oplus$$

derived from the Church encoding $N = BT(M)$ of the infinite tree $\langle \mathcal{G} \rangle$ depicted in (2) and generated by the λY-term M in (8) encoding the higher-order recursion scheme $\mathcal{G}$. We find convenient to write its type as

$$(\oplus \Rightarrow \oplus \Rightarrow \oplus) \;\Rightarrow\; (\oplus \Rightarrow \oplus) \;\Rightarrow\; \oplus \;\Rightarrow \oplus.$$

We depict in Figure 1 a run-tree of a higher-order automaton $\mathbf{\Sigma}$ with empty alphabet and thus empty transition function δ, starting from the initial state

$$\{\{q_1\} \multimap \{q_0\} \multimap q_0\} \multimap \{\{q_1\} \multimap q_1\} \multimap \{q_1\} \multimap q_0$$

of high-order state A, in the hierarchy of states generated by the set $Q = \{q_0, q_1\}$ of two ground states q_0 and q_1. Each time the initial state encounters a binder λa, λb or λc, it declares the new letter a, b or c in the alphabet, and extends its transition function δ accordingly. Once it has declared the three letters a, b or c and reached the root of the infinitary λ-term $N = \langle \mathcal{G} \rangle$, the higher-order automaton has $q_0 \in Q_{\oplus}$ as its current state, and the same transition function δ as the tree automaton described in (3), formulated here in a type-theoretic way:

$$\begin{array}{lcll} \delta(a) & = & \{\{q_1\} \multimap \{q_0\} \multimap q_0\} & \in \;\; \mathcal{P}_{\mathrm{fin}}(Q_{\oplus \Rightarrow \oplus \Rightarrow \oplus}) \\ \delta(b) & = & \{\{q_1\} \multimap q_1\} & \in \;\; \mathcal{P}_{\mathrm{fin}}(Q_{\oplus \Rightarrow \oplus}) \\ \delta(c) & = & \{q_1\} & \in \;\; \mathcal{P}_{\mathrm{fin}}(Q_{\oplus}). \end{array}$$

Note that when the higher-order alphabet $\mathbf{\Sigma}$ only contains letters of first-order arity, such a higher-order automaton $\widetilde{\mathcal{A}}$ is the same thing as an alternating tree automaton on the corresponding signature Σ. This correspondence can be seen as the automata-theoretic counterpart of the Church encoding of trees into λ-terms.

F. Higher-order parity automata

A higher-order parity automaton $\mathcal{A}$ on a finite set of parities or colors $\Omega = \{1, \ldots, k\}$ is defined as a tuple

$$\mathcal{A} = (Q, \mathbf{\Sigma}, A, \delta, q_0)$$

in just the same way as a higher-order automaton, except that the set Q_A of higher-order states is replaced in all

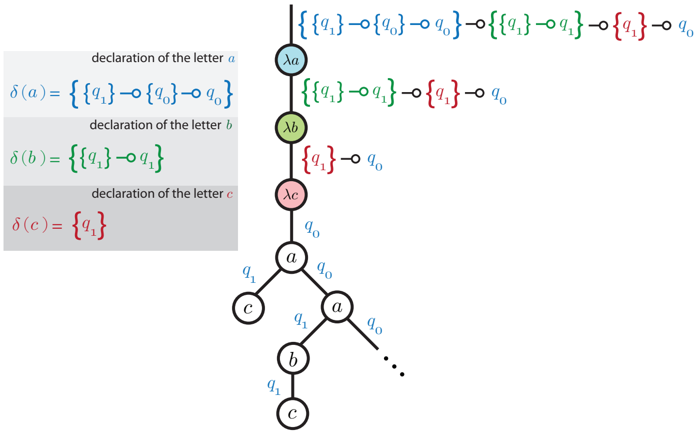


Fig. 1. Illustration of a run-tree of a higher-order automaton $\mathcal{A}$ with an empty alphabet, with type $(\circ \Rightarrow \circ \Rightarrow \circ) \Rightarrow (\circ \Rightarrow \circ) \Rightarrow \circ \Rightarrow \circ$ and with initial state $\{\{q_1\} \multimap \{q_0\} \multimap q_0\} \multimap \{\{q_1\} \multimap q_1\} \multimap \{q_1\} \multimap q_0$.

definitions by the set $\mathbb{Q}_A$ of *colored* higher-order states, defined by induction on the simple type A:

$$\mathbb{Q}_\circ = Q \qquad \mathbb{Q}_{A \Rightarrow B} = \mathcal{P}_{\mathrm{fin}}(\Omega \times \mathbb{Q}_A) \times \mathbb{Q}_B$$

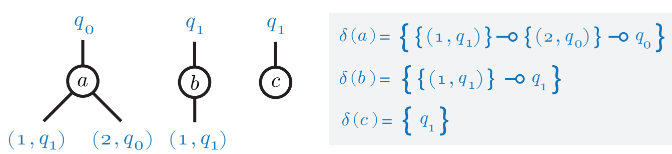
A colored state $q \in \mathbb{Q}_{A \Rightarrow B}$ is thus of the form

$$q \quad = \quad \{(c_1, q_1), \ldots, (c_n, q_n)\} \multimap q'$$

where $q' \in \mathbb{Q}_B$ is a colored state of type B and where $c_i \in \Omega$ is a color and $q_i \in \mathbb{Q}_A$ is a colored state of type A, for all $1 \leq i \leq n$. Accordingly, the transition function of a higher-order parity automaton $\mathcal{A}$ associates to every letter $a_i \in \Sigma$ a finite set

$$\delta(a_i) \in \mathcal{P}_{\mathrm{fin}}(\, \Omega \times \mathbb{Q}_{A_i}\,)$$

of pairs (c, q) consisting of a color $c \in \Omega$ and of a colored state $q \in \mathbb{Q}_{A_i}$ of type $A_i = \mathbf{\Sigma}(a_i)$. By way of illustration, the parity automaton $\mathcal{A}$ with transitions defined in (3) is encoded by the colored transitions below:



where the coinductive parity 2 on the state q_0 and the inductive parity 1 on the state q_1 appear now on the edges of the transitions, rather than on the states themselves, see [5], [6], [7], [8] for a discussion. Every run-tree $\mathcal{R}$ of the higher-order parity automaton $\mathcal{A}$ has its edges labelled by colors $c_i \in \Omega$ and one requires that it satisfies the familiar conditions that in every infinite branch $\mathbb{p}$ of the run-tree $\mathcal{R}$, the maximum color $c \in \Omega$ appearing infinitely often on the edges of $\mathbb{p}$ is even (that is, coinductive).

G. The decidability theorem

Once the notion of higher-order parity automaton $\mathcal{A}$ has been formulated as above, we justify it by establishing a general decidability result for its model-checking problem:

Theorem 1 (Decidability): Suppose given a higher-order parity automaton $\mathcal{A}$ and a finite λY-term M of same higher-order alphabet $\mathbf{\Sigma}$ and of same type A. Then, the question whether the infinitary simply-typed λ-term with boundary $N = BT(M)$ generated by M is accepted by $\mathcal{A}$ is decidable.

The decidability theorem is established in a clean and modular fashion, by combining ideas coming from denotational semantics and from infinitary rewriting theory.

a) Unfolding theorem: Suppose given a higher-order parity automaton $\mathcal{A} = (Q, \Omega, \mathbf{\Sigma}, A, \delta, q_0)$ and a λY-term M with same higher-order alphabet $\mathbf{\Sigma}$ and same type A. We start by establishing that

Theorem 2 (Unfolding): The question whether the infinitary simply-typed λ-term $[M]_\infty$ obtained by infinitary unfolding of M is accepted by the automaton $\mathcal{A}$ is decidable.

We will establish this result by translating the λY-calculus into the $\lambda Y_{\mu\nu}$-calculus, an inductive and coinductive refinement of the λY-calculus where each fixpoint operator Y in a term is treated as either inductive (Y_μ) or coinductive (Y_ν). Our translation of the λY-calculus into the $\lambda Y_{\mu\nu}$-calculus will follow very closely the semantic and comonadic recipe prescribed by Grellois and Melliès in [5], [6], [7]. In particular, the fixpoint operator Y will be interpreted as an alternation of inductive and coinductive fixpoint operators Y_μ and Y_ν, reflecting the parity structure of the higher-order automaton.

b) Invariance theorem: The second and most difficult part in our proof of decidability (Thm. 1) consists in establishing an invariance theorem for our higher-order notion of acceptance.

Theorem 3 (Invariance): Suppose that two simply-typed infinitary λ-terms M and N with higher-order alphabet $\mathbf{\Sigma}$ and arity A are related by an infinitary and strongly convergent β-rewriting path $M \twoheadrightarrow_\beta N$. In that case, a higher-order automaton $\mathcal{A} = (Q, \mathbf{\Sigma}, A, \delta, q_0)$ accepts M if and only if it accepts N.

The property relies on the notion of strongly convergent β-rewriting path introduced in [11] which plays today a central role in infinitary rewriting theory, see also [4]. We see the invariance theorem as the cornerstone of higher-order model-checking, in the same way as the invariance modulo $\beta\eta$-reduction of the interpretation of a simply-typed λ-terms in the finitary case. As it will appear, proving the invariance theorem is far from trivial, and it will be one of the main tasks of the paper.

Put together, the unfolding theorem and the invariance theorem imply the decidability theorem. Indeed, given a finite λY-term M and a higher-order parity automaton $\mathcal{A}$, the first theorem tells us that the acceptance of the infinite unfolding $[M]_\infty$ of M. Then, there exists an infinitary strongly convergent β-rewriting path

$$[M]_\infty \twoheadrightarrow_\beta N$$

to the Böhm tree $N = BT(M)$ generated by M. The invariance theorem (adapted to parity automata) ensures then that the acceptance of $N = BT(M)$ by $\mathcal{A}$ is equivalent to the acceptance of $[M]_\infty$ and is thus decidable.

Plan of the paper

After describing in §II the infinitary simply-typed λ-calculus, we develop in §III the notion of higher-order automaton. We then prove the "forward" and "backward" part of the invariance theorem in §IV and in §VI. The backward part is the more difficult to prove, and we develop in §V a residual theory of diffraction patterns in order to establish it rigorously. We then lift in §VII the invariance theorem from bare λ-terms to λ-terms with boundary. We establish our main theorem (Thm. 1) in §VIII by a translation of the original model-checking problem in $\lambda Y_{\mu\nu}$-calculus, following the ideas of [7].

II. Infinitary simply-typed λ-terms

We introduce now the infinitary simply-typed λ-calculus which will be used in the paper. The notion of simply-typed λ^∞-term is defined in two stages: we start by recalling the coinductive notion of untyped infinitary λ-term formulated in [13], and then refine it into a notion of simply-typed infinitary λ-term. Depending on the situation, the terms of the calculus are called simply-typed and infinitary λ-terms, or simply-typed λ^∞-terms.

A. Untyped infinitary λ-terms

We suppose from now on that Var denotes a countable infinite set of variables, which we find convenient to see as a nominal set, following [13]. The nominal set of untyped λ^∞-terms is then defined coinductively by the grammar

$$M, N \quad ::= \quad \lambda a.M \quad | \quad App(M, N) \quad | \quad a \in Var$$

where the constructor $\lambda a.M$ binds the variable a in the λ^∞-term M, in the nominal sense. One consequence of this nominal and coinductive definition is that the set $\mathbf{fv}(M) \subseteq Var$ of free variables of an untyped λ^∞-term M is always finite, see [13] for a discussion. On the other hand, an untyped λ^∞-term M in that sense may contain an infinite countable number of bounded variables, which is precisely what happens when one unfolds a λY-term M into an infinitary λ-term $[M]_\infty$, see an illustration in the Appendix, §F. Moreover, a free variable $a \in \mathbf{fv}(M)$ may appear at a countable (finite or infinite) number of occurrences of the λ^∞-term M. Hence, a binder $\lambda a.M$ may bind a countable (finite or infinite) set of occurrences of the free variable a in M. Note also that we prefer to use here the notation $App(M, N)$ for the application, instead of the more familiar notation MN used in the introduction.

B. Simply-typed infinitary λ-terms

A typing judgment

$$\mathbf{\Sigma} \vdash M : A$$

is defined as a triple consisting of a higher-order alphabet $\mathbf{\Sigma}$, of an untyped λ^∞-term M and of a simple type A, such that all the free variables of M are letters in the alphabet $\mathbf{\Sigma}$. A typing derivation Υ is a syntactic tree defined coinductively using the three typing rules below:

$$\text{Variable} \quad \frac{}{\mathbf{\Sigma}, a : A \vdash a : A}$$

$$\text{Abstraction} \quad \frac{\mathbf{\Sigma}, a : A \vdash M : B}{\mathbf{\Sigma} \vdash \lambda a.M : A \Rightarrow B}$$

$$\text{Application} \quad \frac{\mathbf{\Sigma} \vdash M : A \Rightarrow B \qquad \mathbf{\Sigma} \vdash N : A}{\mathbf{\Sigma} \vdash App(M, N) : B}$$

The conclusion of a typing derivation Υ is defined as the typing judgment $\mathbf{\Sigma} \vdash M : A$ labelling the root of Υ. A simply-typed λ^∞-term of type A in the higher-order alphabet $\mathbf{\Sigma}$ is defined as a pair (M, Υ_M) consisting of an untyped λ^∞-term M and of a typing derivation Υ_M with conclusion $\mathbf{\Sigma} \vdash M : A$. Note that the typing derivation Υ_M assigns a unique type to every node of the infinitary λ-term M. As it is customary in the λ-calculus, we often identify the simply-typed λ^∞-term (M, Υ_M) with the underlying untyped λ^∞-term M, and keep the derivation tree Υ_M implicit.

C. Infinitary rewriting

In order to establish our decidability theorems for higher-order automata, we will need to study very closely the infinitary rewriting paths $f : M \twoheadrightarrow_\beta N$ which transform an infinitary λ-term M, typically the unfolding a simply-typed λY-term, into another infinitary λ-term N. A typical example is provided by the rewriting process which transforms a higher-order recursion scheme M seen as an infinitary λ-term into the infinitary tree N it generates. In order to define such a notion of infinitary rewriting, we follow [11] and thus start by introducing the notion of occurrence. An occurrence is a finite word constructed on the grammar

$$o \quad ::= \quad \varepsilon \quad | \quad \mathsf{body} \cdot o \quad | \quad \mathsf{fun} \cdot o \quad | \quad \mathsf{arg} \cdot o.$$

The set $\mathsf{occ}(M)$ of occurrences of an simply-typed and infinitary λ-term M is defined by coinduction

$$\begin{array}{lcl}
\mathsf{occ}(a) & = & \{\,\varepsilon\,\} \\
\mathsf{occ}(\lambda a.\, M) & = & \{\,\varepsilon\,\} \uplus \{\,\mathsf{body} \cdot o \,|\, o \in \mathsf{occ}(M)\,\} \\
\mathsf{occ}(App(M, N)) & = & \{\,\varepsilon\,\} \uplus \{\,\mathsf{fun} \cdot o \,|\, o \in \mathsf{occ}(M)\,\} \\
 & & \qquad \uplus \{\,\mathsf{arg} \cdot o \,|\, o \in \mathsf{occ}(N)\,\}
\end{array}$$

The purpose of an occurrence is to designate a specific position in an infinitary λ-term M. Given two occurrences $o, o' \in \mathsf{occ}(M)$, we write $o \preceq_M o'$ and say that the occurrence o nests the occurrence o' when o is a prefix of o'. Every occurrence $o \in \mathsf{occ}(M)$ induces a context $\mathbf{C}_{(M,o)}[-]$ defined as the infinitary λ-term M with a unique hole at the occurrence o ; and an infinitary λ-term $M_{|o}$ defined as the subterm at occurrence o of the infinitary λ-term M. We write $\mathbf{type}(M, o)$ for the type of the subterm $M_{|o}$ of M. One recovers the infinitary λ-term M by plugging the infinitary λ-term $M_{|o}$ inside the unique hole of the context $\mathbf{C}_{(M,o)}[-]$ in the following way:

$$M = \mathbf{C}_{(M,o)}[M_{|o}]$$

A β-redex R is defined as a triple (M, o, N) consisting of a simply-typed infinitary λ-term M and of an occurrence $o \in \mathsf{occ}(M)$ such that M restricted to the occurrence o is a β-reduction pattern

$$M_{|o} = App(\lambda a.\, P, Q)$$

with the infinitary λ-term $N = \mathbf{C}_{(M,o)}[P\,[\,a := Q\,]]$ obtained by plugging the infinitary λ-term $P\,[\,a := Q\,]$ inside the unique hole of the context $\mathbf{C}_{(M,o)}[-]$. See [11], [4], [13] for the definitions of context and of substitution in the infinitary λ-calculus. An important property of the simply-typed λ^{∞}-calculus is the following

Proposition 1 (Subject Reduction): Suppose that a simply-typed λ^{∞}-term M is defined by the typing derivation Υ_M of the typing judgment $\mathbf{\Sigma} \vdash M : A$ and that (M, o, N) is a β-redex. Then, there exists a canonical typing derivation Υ_N of the typing judgment $\mathbf{\Sigma} \vdash N : A$ which turn N into a simply-typed infinitary λ-term, with the same alphabet $\mathbf{\Sigma}$ and the same type A as the original λ-term M.

From this, one defines a graph $\mathcal{G}(\mathbf{\Sigma}, A)$ whose vertices are the infinitary simply-typed λ-terms M, N of alphabet $\mathbf{\Sigma}$ and of type A, and whose edges $M \to N$ are the β-redexes (M, o, N). The graph $\mathcal{G}(\mathbf{\Sigma}, A)$ is called the β-rewriting graph associated to $\mathbf{\Sigma}$ and A. The depth of a β-redex (M, o, N) is the length of its occurrence o. This notion of depth counts every tag body, fun and node in the occurrence, and thus coincides with the depth associated to the triple $abc = 111$ and noted D^{111} in Def. 6 of [11]. The diameter $||R||$ of a β-redex $R = (M, o, N)$ is defined as the fraction

$$||R|| = \frac{1}{2^n}$$

where n is the depth of the β-redex. By extension, the diameter of a finite path

$$f \;=\; M_0 \xrightarrow{R_0} M_1 \xrightarrow{R_1} \cdots \longrightarrow M_{p-1} \xrightarrow{R_{p-1}} M_p$$

in the rewriting graph $\mathcal{G}(\mathbf{\Sigma}, A)$ is defined as the maximum $||f|| = \max\{||R_i||, 0 \leq i \leq p-1\}$ of the diameters of its edges. Accordingly, the diameter of an empty path is zero. Now, consider an infinite path

$$f \;=\; M_0 \overset{R_0}{\longrightarrow} M_1 \overset{R_1}{\longrightarrow} \cdots \longrightarrow M_{n-1} \overset{R_{p-1}}{\longrightarrow} M_p \overset{R_p}{\longrightarrow} \cdots$$

in the rewriting graph $\mathcal{G}(\mathbf{\Sigma}, A)$. Every pair of natural numbers $p, q \in \mathbb{N}$ with $p \leq q$ induces a finite rewriting path, obtained by restriction:

$$f_{|p,q} \;=\; M_p \xrightarrow{R_p} M_{p+1} \xrightarrow{R_{p+1}} \cdots \longrightarrow M_{q-1} \xrightarrow{R_{q-1}} M_q$$

An infinite path in the graph $\mathcal{G}(\mathbf{\Sigma}, A)$ is strongly convergent when for every natural number $n \in \mathbb{N}$, there exists a natural number $\mathsf{N}(n) \in \mathbb{N}$ such that

$$\forall p, q \in \mathbb{N}, \quad \mathsf{N}(n) < p \leq q \quad \Rightarrow \quad ||f_{|p,q}|| < \frac{1}{2^n}.$$

Our notion of strong convergence coincides with the notion of strong convergence associated to the measure of depth $abc = 111$ formulated in [11], see also [4]. The definition of strong convergence enables us to establish an infinitary version of the Subject Reduction property:

Proposition 2 (Infinitary Subject Reduction): Suppose that the simply-typed λ^{∞}-term M is defined by the typing derivation Υ_M of the typing judgment $\mathbf{\Sigma} \vdash M : A$ and that $f : M \twoheadrightarrow_{\beta} N$ is a strongly convergent β-rewriting path. Then, there exists a canonical typing derivation Υ_N of the typing judgment $\mathbf{\Sigma} \vdash N : A$ which turn N into a simply-typed infinitary λ-term, with the same alphabet $\mathbf{\Sigma}$ and the same type A as the original λ-term M.

III. Higher-order automata

A. Higher-order states

Recall that a preorder is a transitive and reflexive binary relation and that a preordered set $(X, \sqsubseteq)$ is a set X equipped with a preorder $\sqsubseteq$. From now on, we suppose given a finite preordered set of ground states $(Q, \sqsubseteq_Q)$. Every simply type A induces a preordered set $(Q_A, \sqsubseteq_A)$ of higher-order states of type A, defined by structural induction on the type A. The preordered set $(Q_{\mathbb{o}}, \sqsubseteq_{\mathbb{o}})$ of ground states is defined as the set Q equipped with the preorder $\sqsubseteq_Q$:

$$\forall q, q' \in Q_{\mathbb{o}}, \qquad\qquad q \sqsubseteq_{\mathbb{o}} q' \iff q \sqsubseteq_Q q'.$$

The preordered set $(Q_{A\Rightarrow B}, \sqsubseteq_{A\Rightarrow B})$ is defined in two stages, using the linear decomposition formula (11) of the intuitionistic arrow. Given a simple type A, the preordered set $(Q_{!A}, \sqsubseteq_{!A})$ is defined as the set of finite subsets of higher-order states of A

$$Q_{!A} \quad = \quad \mathcal{P}_{\mathrm{fin}}(Q_A)$$

equipped with the following preorder:

$$\{a_1, \ldots, a_m\} \quad \sqsubseteq_{!A} \quad \{b_1, \ldots, b_n\}$$

if and only if there exists a functor $f : [m] \to [n]$ such that $a_i \sqsubseteq_A b_{f(i)}$ where $[m] = \{1, \ldots, m\}$ denotes the finite set of natural numbers with m elements. The preordered set $(Q_{A\Rightarrow B}, \sqsubseteq_{A\Rightarrow B})$ of higher-order states of type $A \Rightarrow B$ is then defined as the product:

$$Q_{A\Rightarrow B} = Q_{!A} \times Q_B \qquad \sqsubseteq_{A\Rightarrow B} \,=\, \sqsubseteq_{!A}^{op} \times \sqsubseteq_B .$$

where $\sqsubseteq_{!A}^{op}$ denotes the preorder obtained by reversing the orientation of the preorder $\sqsubseteq_{!A}$.

B. Higher-order automata

A higher-order automaton $\mathcal{A}$ with type B and with higher-order alphabet

$$\mathbf{\Sigma} : \Sigma \longrightarrow Type$$

is defined as a tuple $(Q, \mathbf{\Sigma}, B, \delta, q_0)$ where Q is a set of states ; where the transition function δ associates to every letter $a \in \Sigma$ a finite set

$$\delta(a) \in \mathcal{P}_{\text{fin}}(Q_A)$$

of higher-order states of type $A = \mathbf{\Sigma}(a)$ in the hierarchy of higher-order states generated by the set of ground states $Q_{\mathtt{o}} = Q$; where B is a type ; and where $q_0 \in Q_B$ is a higher-order state of type B called the *initial state* of the automaton. Note that the transition function δ may be equivalently seen as a vector of finite sets of higher-order states:

$$\delta \in \prod_{a \in \Sigma} \mathcal{P}_{\text{fin}}(Q_{\mathbf{\Sigma}(a)})$$

indexed by letters of the finite alphabet Σ of the automaton $\mathcal{A}$. This remark enables us to define a preorder relation $\sqsubseteq_{\mathbf{\Sigma}}$ on transition functions δ, δ' of the same alphabet $\mathbf{\Sigma}$, using the preorder relation $\sqsubseteq_{!A}$ on finite sets of higher-order states of type A, in the following way: we write $\delta \sqsubseteq_{\mathbf{\Sigma}} \delta'$ when $\forall a \in \Sigma, \delta(a) \sqsubseteq_{!\mathbf{\Sigma}(a)} \delta'(a)$. This leads us to the following preorder relation between higher-order automata:

Definition 1 (Preorder on automata): Given two higher-order automata $\mathcal{A} = (Q, \mathbf{\Sigma}, A, \delta, q)$ and $\mathcal{A}' = (Q, \mathbf{\Sigma}, A, \delta', q')$ with same alphabet $\mathbf{\Sigma}$ and same type A, one writes $\mathcal{A}' \sqsubseteq_{\mathbf{\Sigma},A} \mathcal{A}$ precisely when $\delta \sqsubseteq_{\mathbf{\Sigma}} \delta'$ and $q' \sqsubseteq_A q$.

C. Run-trees

We find convenient to define in a type-theoretic fashion the set of run-trees $\langle\, \mathbf{\Sigma} \vdash M : A \,|\, \delta, q \,\rangle$ of an higher-order automaton $\mathcal{A} = (Q, \mathbf{\Sigma}, A, \delta, q)$ against a simply-typed infinitary λ-term $\mathbf{\Sigma} \vdash M : A$ with same alphabet $\mathbf{\Sigma}$ and same type A. The reader more inclined towards an automata-theoretic style will find an equivalent formulation in the Appendix, §B.

Definition 2 (Run-trees): A run-tree $\mathcal{R}$ of the automaton $\mathcal{A}$ against the infinitary λ-term M is defined as a possibly infinite derivation tree of the judgment

$$\langle\, \mathbf{\Sigma} \vdash M : A \,|\, \delta, q \,\rangle$$

in the deduction system generated by the rules below:

$$\text{Var}\ \frac{q \sqsubseteq_A q' \qquad q' \in \delta(a)}{\langle\, \mathbf{\Sigma}, a : A \vdash a : A \,|\, \delta, q \,\rangle}$$

$$\text{Abs}\ \frac{\langle\, \mathbf{\Sigma}\,,\, a : A \vdash M : B \ |\ \delta + a \mapsto \{q_1, \ldots, q_n\}\,,\, q \,\rangle}{\langle\, \mathbf{\Sigma} \vdash \lambda a.M : A \Rightarrow B \ |\ \delta\,,\, \{q_1, \ldots, q_n\} \multimap q \,\rangle}$$

$$\text{App}\ \frac{\langle\, \mathbf{\Sigma} \vdash M : A \Rightarrow B \,|\, \delta, u \multimap q \,\rangle \qquad \langle\!\langle\, \mathbf{\Sigma} \vdash N : A \,|\, \delta, u \,\rangle\!\rangle}{\langle\, \mathbf{\Sigma} \vdash App(M, N) : B \,|\, \delta, q \,\rangle}$$

$$\text{Bag}\ \frac{\langle\, \mathbf{\Sigma} \vdash M : A \,|\, \delta, q_1 \,\rangle \quad \ldots \quad \langle\, \mathbf{\Sigma} \vdash M : A \,|\, \delta, q_n \,\rangle}{\langle\!\langle\, \mathbf{\Sigma} \vdash M : A \,|\, \delta, \{q_1, \ldots, q_n\} \,\rangle\!\rangle}$$

In the rule *App*, $u = \{q_1, \ldots, q_n\}$ denotes a finite set of higher-order states $q_1, \ldots, q_n$ of type A. The *Bag* rule then reflects the alternating nature of our higher-order automata. In particular, we use the notation

$$\langle\!\langle\, \mathbf{\Sigma} \vdash M : A \,|\, \delta, \{q_1, \ldots, q_n\} \,\rangle\!\rangle \tag{12}$$

to denote the set of n-tuples of run-trees:

$$\prod_{1 \leq i \leq n} \langle\, \mathbf{\Sigma} \vdash M : A \,|\, \delta, q_i \,\rangle$$

Note that such a n-tuple of run-trees is the same thing as an infinitary derivation tree with conclusion (12) in the deduction system just defined. We will use in §VI (proof of Thm. 5) an infinite-branching variant of run-tree, where the *Bag* is rule is replaced by the ∞-*Bag* rule:

$$\infty\text{-Bag}\ \frac{\forall i \in I,\ \langle\, \mathbf{\Sigma} \vdash M : A \,|\, \delta, q_i \,\rangle}{\langle\!\langle\, \mathbf{\Sigma} \vdash M : A \,|\, \delta, u \,\rangle\!\rangle}$$

with a countable (finite or infinite) family of premises indexed by $i \in I$, and with $u \in \mathcal{P}_{\text{fin}}(Q_A)$ defined as the finite set $u = \{q \in Q_A \,|\, \exists i \in I, q = q_i\}$. Note that the existence of a run-tree is equivalent to the existence of an infinite-branching run-tree for a higher-order automaton.

D. Acceptance

This leads us to the following definition of acceptance for a higher-order automaton.

Definition 3: A simply-typed and infinitary λ-term M of alphabet $\mathbf{\Sigma}$ and of type A is accepted by a higher-order automaton $\mathcal{A} = (Q, \mathbf{\Sigma}, A, \delta, q)$ precisely when the set of run-trees $\langle\, \mathbf{\Sigma} \vdash M : A \,|\, \delta, q \,\rangle$ is non-empty.

The next statement shows that acceptance is closed under the preorder relation between higher-order automata formulated at the end of §III-B (Def. 1).

Proposition 3: Suppose given two higher-order automata $\mathcal{A}$ and $\mathcal{A}'$ with same alphabet $\mathbf{\Sigma}$ and same type A, such that $\mathcal{A}' = (Q, \mathbf{\Sigma}, A, \delta', q') \sqsubseteq_{\mathbf{\Sigma},A} \mathcal{A} = (Q, \mathbf{\Sigma}, A, \delta, q)$. In that case, every simply-typed and infinitary λ-term of alphabet $\mathbf{\Sigma}$ and of type A accepted by $\mathcal{A}$ is also accepted by $\mathcal{A}'$.

Proof. Technically speaking, an easy proof by coinduction on the four rules *Abs*, *App*, *Var* and *Bag* of the deduction system enables one to replace every run-tree $\mathcal{R}$ in $\langle\, \mathbf{\Sigma} \vdash M : A \,|\, \delta, q \,\rangle$ by a run-tree $\mathcal{R}'$ in $\langle\, \mathbf{\Sigma} \vdash M : A \,|\, \delta', q' \,\rangle$ defined by increasing the transition function and by decreasing the initial state at each node of the run-tree $\mathcal{R}$. More conceptually, the preorder reflects the fact that a run-tree does not have to use all the transitions offered by the transition function of the higher-order automaton.

IV. Forward preservation theorem

We prove the first and easier direction of the invariance theorem (Thm. 3) by establishing that

Theorem 4 (Forward Preservation): Suppose that two simply-typed infinitary λ-terms M and N with alphabet $\mathbf{\Sigma}$ and type A are related by a strongly convergent β-rewriting path $f : M \twoheadrightarrow_\beta N$. In that case, every higher-order automaton $\mathcal{A}$ with alphabet $\mathbf{\Sigma}$ and type A which accepts M also accepts N.

Proof. The proof is based on an easy adaptation to the deduction system in §III-C of the Infinitary Subject Reduction theorem (Prop. 2) established for the original deduction system of the simply-typed λ-calculus in §II-B. The key observation is that one can substitute run-trees in variable occurrences of run-trees, in the same way as one substitutes infinitary simply-typed λ-terms in variable occurrences of infinitary simply-typed λ-terms. In this way, it is possible to transform the original run-tree $\mathcal{R}$ of the automaton $\mathcal{A}$ on M into a run-tree $\mathcal{R}_{p+1}$ of the same automaton $\mathcal{A}$ on the infinitary λ-term M_{p+1} obtained after computing the prefix

$$f_{|0,p} \quad = \quad M_0 \xrightarrow{R_0} M_1 \xrightarrow{R_1} \cdots \longrightarrow M_{p-1} \xrightarrow{R_{p-1}} M_p$$

of length p of the infinitary β-rewriting path f. The fact that f is a strongly convergent infinitary β-rewriting path ensures that the family of run-trees $(\mathcal{R}_p)_{p\in\mathbb{N}}$ converges towards a run-tree $\mathcal{R}_\infty$ of the simply-typed infinitary λ-term N. This establishes the theorem.

V. A DIFFRACTION THEORY

In this section, we introduce the notion of *diffraction pattern* which extends the familiar notion of *occurrence* of a simply-typed λ^∞-term. We then develop a residual theory for diffraction patterns which refines the residual theory for occurrences. This "diffraction theory" will be the main tool to prove in §VI the most difficult direction: backward preservation, of our invariance theorem (Thm. 3). We start by defining the notion in §V-A and §V-B and then illustrate it in §V-C ; although the notion of diffraction pattern is extremely natural, it is also new, and we thus advise the reader to read §V-A, §V-B and §V-C simultaneously, and to look at the diffraction of an occurrence of M along a strongly convergent rewriting path $f : M \twoheadrightarrow_\beta N$ illustrated in the Appendix, §F.

A. Diffraction patterns

We define the notion of a diffraction pattern of type A in a simply-typed λ^∞-term M, by structural induction on the type A. We recall that the set of occurrences of M is denoted by $\mathsf{occ}(M)$ and that every occurrence $o \in \mathsf{occ}(M)$ has a simple type in M, noted $\mathbf{type}(M, o)$.

Definition 4 (Diffraction pattern of ground type): A diffraction pattern D of type $\oplus$ in M is an occurrence $o \in \mathsf{occ}(M)$ of ground type $\oplus$ in M ; the head occurrence of the diffraction pattern $D = o$ is defined as $\mathsf{hdocc}(D) = o$.

Definition 5 (Diffraction pattern of arrow type): A diffraction pattern D of type $A \Rightarrow B$ in M is either

- an occurrence $D = o$ of type $A \Rightarrow B$ in M ; the head occurrence of the diffraction pattern D is defined in that case as $\mathsf{hdocc}(D) = o$,
- a pair $D = (\{D_{A,i} \,|\, i \in I\}, D_B)$ also noted

$$D \quad = \quad \{D_{A,i} \,|\, i \in I\} \multimap D_B$$

consisting of a family $\{D_{A,i} \,|\, i \in I\}$ of diffraction patterns $D_{A,i}$ of type A in M, indexed by a countable (finite or infinite) set I, together with a diffraction pattern D_B of type B in M ; one requires moreover that the head occurrence of the diffraction pattern D_B nests in M the head occurrences of the diffraction patterns $D_{A,i}$, as follows:

$$\forall i \in I, \qquad \mathsf{hdocc}(D_B) \preceq_M \mathsf{hdocc}(D_{A,i})$$

the head occurrence of the diffraction pattern D is then defined as the head occurrence $\mathsf{hdocc}(D) = \mathsf{hdocc}(D_B)$ of the diffraction pattern D_B in M.

The definition of a diffraction pattern of ground type $\oplus$ and of arrow type $A \Rightarrow B$ ensures that every occurrence $o \in \mathsf{occ}(M)$ of the simply-typed λ^∞-term M may be understood as a specific kind of diffraction pattern $D = o$ with the same type $\mathbf{type}(M, o)$ as the occurrence o in M. We are thus allowed to see diffraction patterns as a generalisation of occurrences, and to write

$$\mathsf{occ}(M) \subseteq \mathsf{diff}(M)$$

where $\mathsf{diff}_A(M)$ denotes the set of diffraction patterns of type A in M, and denotes the set of diffraction patterns of arbitrary type $A \in Type$ in M.

B. The shape of a diffraction pattern

What makes the notion of diffraction pattern D so interesting for higher-order model-checking is that every such $D \in \mathsf{diff}(M)$ comes together with a rooted tree $\mathsf{shape}(D)$ called the shape of D. The rooted tree $\mathsf{shape}(D)$ is defined by induction on the diffraction pattern D:

- when the diffraction pattern D of ground type $\oplus$ or of arrow type $A \Rightarrow B$ is equal to an occurrence $o \in \mathsf{occ}(M)$ of the same type, the rooted tree $\mathsf{shape}(D)$ is equal to the trivial rooted tree with one root labelled $*$ and no other node,
- when the diffraction pattern D of type $A \Rightarrow B$ is equal to an arrow $D = \{D_{A,i} \,|\, i \in I\} \multimap D_B$, the rooted tree $\mathsf{shape}(D)$ is defined as the rooted tree $\mathsf{shape}(D_B)$ with root $*_B$, extended with the rooted trees $\mathsf{shape}(D_{A,i})$ with their roots $*_{A,i}$ connected as children to the root $*_B$ of $\mathsf{shape}(D_B)$.

Note that every node of $\mathsf{shape}(D)$ is a diffraction pattern of M. This enables us to define the function

$$\mathsf{embed}_D \quad : \quad (\mathsf{shape}(D), \preceq_D) \quad \longrightarrow \quad (\mathsf{occ}(M), \preceq_M)$$

which maps every node $*_E \in \mathsf{shape}(D)$ to the head occurrence $\mathsf{hdocc}(E) \in \mathsf{occ}(M)$ of the underlying diffraction pattern E. Note that embed_D defines a monotone function

embed_D from the set of nodes of $\mathsf{shape}(D)$ equipped with its tree nesting order $\preceq_D$, to the set of occurrences $\mathsf{occ}(M)$ of the simply-typed λ^∞-term M, equipped with the nesting order $\preceq_M$.

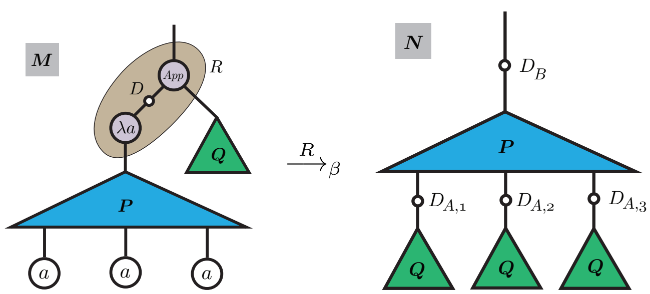
C. Residuals of diffraction patterns

Suppose given a β-redex $R = (M, o, N)$ with occurrence $o_R = o$ of type $A \Rightarrow B$ of the form

$$M = \mathbf{C}_{(M,o)}[(\lambda a.P)Q] \quad \xrightarrow{R}_\beta \quad \mathbf{C}_{(M,o)}[P\,[\,a := Q\,]] = N$$

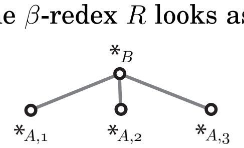
where $\mathbf{C}_{(M,o)}[-]$ is the context with unique hole at occurrence o defined in §II-C, and where the λ-term P has type A and the λ-term Q has type B.

We are interested now in what happens to the occurrence o_R of the β-redex R of type $A \Rightarrow B$ after β-reduction of R. In the traditional theory of residuals, the occurrence o_R of type $A \Rightarrow B$ has no residual along the β-reduction of R. The intuition is that the occurrence o has been "consumed" during the process of β-reduction, and thus disappears. The intuition underlying our diffraction theory is quite different: our idea is that the occurrence o_R has not really disappeared, but rather that it has been "diffracted" by the β-redex R into a diffraction pattern $E = \{D_{A,i} \mid i \in I\} \multimap D_B$ defined with the following components: the occurrence $D_B = o_B$ of P of type B in N, and an occurrence $D_{A,i} = o_{A,i}$ in N for each occurrence of the free variable a in M, and thus, for each occurrence of a copy Q in N. The situation can be depicted as follows:



where we depict the occurrence $D = o_R$ as a white circle in the λ^∞-term M, and similarly for the occurrence $D_B = o_B$ and each occurrence $D_{A,i} = o_{A,i}$ in the λ-term N, for $i \in I$. Note that the shape of the diffraction pattern E in N residual of D along the β-redex R looks as follows:

$\mathsf{shape}(E) \quad =$



and that the monotone function embed_D transports every node of $\mathsf{shape}(E)$ to the following occurrences: $*_B \mapsto o_B$ and $*_{A,i} \mapsto o_{A,i}$. Note that the function embed_D is indeed monotone, since $o_B \preceq_N o_{A,i}$ for all $i \in I$.

Given a β-redex $R = (M, o, N)$ as above, the definition of a residual $E \in \mathsf{diff}(N)$ of a diffraction pattern $D \in \mathsf{diff}(M)$ along a β-redex $M \to_\beta N$ is done by induction on the type of D. As a matter of fact, the definition is not difficult to reconstruct starting from the key example just described. It is given in the Appendix, §E. We write $D[R]E$ when $E \in \mathsf{diff}(N)$ is a residual of $D \in \mathsf{diff}(M)$ along a β-redex $R : M \to_\beta N$. By extension, we write $D[f]E$ when $E \in \mathsf{diff}(N)$ is a residual of $D \in \mathsf{diff}(M)$ along a finite β-rewriting path from M to N, in the expected sense, see the Appendix, §D. An important observation is that

Proposition 4: Every strongly convergent infinitary β-rewriting path $M \twoheadrightarrow_\beta N$ induces a residual relation

$$[f] \quad \subseteq \quad \mathsf{diff}(M) \times \mathsf{diff}(N).$$

This residual relation will play a central role in the proof of the backward preservation theorem (Thm. 5) next section.

VI. Backward preservation theorem

We are ready now to establish the second (and more difficult) direction of our invariance theorem:

Theorem 5 (Backward preservation): Suppose that two simply-typed and infinitary λ-terms M and N with higher-order alphabet $\mathbf{\Sigma}$ and arity A are related by a strongly convergent β-rewriting path $M \twoheadrightarrow_\beta N$. In that case, a higher-order automaton $\mathcal{A} = (Q, \mathbf{\Sigma}, A, \delta, q)$ which accepts N also accepts M.

Proof. In order to establish the theorem, we need to deduce the existence of a run-tree $\mathcal{R}_M$ of the automaton $\mathcal{A} = (Q, \mathbf{\Sigma}, A, \delta, q)$ on the λ^∞-term M from the existence of a run-tree $\mathcal{R}_N$ of $\mathcal{A}$ on the λ^∞-term N, using the fact that M and N are related by a strongly convergent β-rewriting path $f : M \twoheadrightarrow_\beta N$. The diffraction theory formulated in §V plays a key role in that reconstruction. First of all, it follows from Prop. 4 that every occurrence $o \in \mathsf{occ}(M)$ has a countable set $o[f] \subseteq \mathsf{diff}(N)$ of diffraction patterns E as residuals along the strongly convergent path f. Every such diffraction pattern $E \in o[f]$ residual of $D = o$ along f induces a set $E[\mathcal{R}_N]$ of diffraction patterns F in the run-tree $\mathcal{R}_N$. The key observation is that every such diffraction pattern $F \in \mathsf{diff}(\mathcal{R}_N)$ comes equipped with a higher-order state $q_F \in Q_A$, where $A = \mathbf{type}(M, o)$ denotes the type of the occurrence o in M. The higher-order state q_F is defined by collecting the states appearing on the nodes of the rooted tree $\mathsf{shape}(F)$ embedded in the run-tree $\mathcal{R}_N$. If we write $[f \cdot \mathcal{R}_N]$ for the composite relation $[f] \circ [\mathcal{R}_N]$, we associate in this way to every occurrence $o \in \mathsf{occ}(M)$ of type $A = \mathbf{type}(M, o)$ in M a countable set $\mathcal{D}_o = o[f \cdot \mathcal{R}_N]$ of diffraction patterns F in $\mathcal{R}_N$, each of them labelled by a higher-order state $q_F \in Q_A$. A careful inspection establishes that the elements of the $\mathcal{D}_o$'s combine together to define an infinite-branching run-tree $\mathcal{R}_M$ of the automaton $\mathcal{A}$ on the λ^∞-term M, in the sense of §III-C. The construction of $\mathcal{R}_M$ from $\mathcal{D}$ is a variant of the Grothendieck construction. The equivalence in §III-C between run-trees and infinite-branching run-trees concludes the proof.

The invariance theorem (Thm. 3) follows immediately, as a consequence of the forward and backward preservation theorems (Thm. 4 and 5). This invariance theorem enables us to establish a primary decidability theorem (Thm. 6) for λY-terms and higher-order automata. The theorem relies on a refined notion of Böhm tree, better adapted to higher-order model-checking, and parametrized by the set of ground states Q.

Definition 6 (Böhm tree): A Böhm tree N is defined by the coinductive grammar

$$N ::= \quad \lambda a_1 \ldots \lambda a_m . a N_1 \ldots N_n \mid \perp_{\mathbf{\Sigma} \vdash u : A}$$

where $\perp_{\mathbf{\Sigma} \vdash u:A}$ is a constant of the calculus indexed by a higher-order alphabet $\mathbf{\Sigma}$, a type A, and a finite subset u of higher-order states of type $\mathbf{\Sigma} \Rightarrow A$.

Using the infinitary Church-Rosser property in [11], one establishes that given a finite set Q of ground states, every simply-typed λ^{∞}-term M has a unique Böhm tree $BT(M)$. The usual (inductive) notion of Böhm tree corresponds to the case where $u = \emptyset$ for every constant $\perp_{\mathbf{\Sigma} \vdash u:A}$; an λ^{∞}-term M is called productive when its Böhm tree $BT(M)$ does not contain any constant of the form $\perp_{\mathbf{\Sigma} \vdash u:A}$.

Theorem 6 (Decidability): Suppose given a higher-order automaton $\mathcal{A}$ and a simply-typed λY-term M of same higher-order alphabet $\mathbf{\Sigma}$ and of same type A. Then, the question whether the infinitary simply-typed λ-term $N = BT(M)$ generated by M is accepted by $\mathcal{A}$ is decidable.

VII. Lambda-terms with boundary

The invariance theorem (Thm. 3) has been established in §VI for a very primitive notion of infinitary λ-term, of a purely coinductive nature. In this section, we extend the theorem to an inductive and coinductive notion of infinitary λ-term, called λ-term with boundary.

Definition 7 (Infinite Path): An infinite path $\mathbb{p}$ in an infinitary λ-term M is an infinite sequence $\mathbb{p} = (o_n)_{n \in \mathbb{N}}$ of occurrences of M such that o_0 is the empty occurrence ε, the occurrence $o_{n+1} \in \mathsf{occ}(M)$ immediately extends the occurence $o_n \in \mathsf{occ}(M)$ for all $n \in \mathbb{N}$, in the sense that one of the three following cases occurs: $o_{n+1} = o_n \cdot \mathsf{body}$, $o_{n+1} = o_n \cdot \mathsf{fun}$ or $o_{n+1} = o_n \cdot \mathsf{arg}$. We write $\infty\text{-}\mathsf{path}(M)$ for the set of infinite paths of M. Given $\mathbb{p} = (o_n)_{n \in \mathbb{N}}$ in M, we write $\mathbb{p}_{|n}$ for the occurrence $o_n \in \mathsf{occ}(M)$.

Definition 8 (Boundary): A boundary Þ of a simply-typed infinitary λ-term M is defined as a set $\text{Þ} \subseteq \infty\text{-}\mathsf{path}(M)$ of infinite paths of M. A simply-typed infinitary λ-term with boundary is a triple

$$(M, \Upsilon_M, \text{Þ}_M)$$

consisting of a simply-typed infinitary λ-term $(M, \Upsilon(M))$ together with a boundary Þ_M of the infinitary λ-term M.

The intuition is that an infinite path $\mathbb{p}$ should be considered as part of the λ-term M precisely when $\mathbb{p}$ is an element of the boundary Þ_M, and not otherwise. The notion of boundary is inspired by the definition of a topological game whose winning condition is described by a subset Ω (e.g. borelian or projective) of an underlying topological space, see [9] for details. It is worth mentioning that every such infinite game may be encoded as a boundary Þ on the Church encoding $\mathsf{BinTree}$ of the infinite binary tree of signature $\mathbf{\Sigma} = \{a : 2\}$.

The purpose of λ-terms with boundary is to provide an interpretation of terms of the $\lambda Y_{\mu\nu}$-calculus, defined as the simply-typed λ-calculus extended with

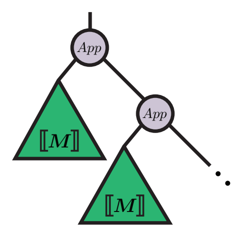
– an inductive fixpoint operator $\quad Y_\mu : (A \Rightarrow A) \Rightarrow A$
– a coinductive fixpoint operator $\quad Y_\nu : (A \Rightarrow A) \Rightarrow A$

Proposition 5: Every simply-typed $\lambda Y_{\mu\nu}$-term M defines an infinitary simply-typed λ-term with boundary

$$[M]_\infty = ([M]_\infty, \Upsilon_{[M]_\infty}, \text{Þ}_{[M]_\infty})$$

by structural induction on the $\lambda Y_{\mu\nu}$-term M.

The construction is easy to describe. The infinitary simply-typed λ-term $[M]_\infty$ is obtained by infinite unfolding of M, using the equations $Y_\mu \, P \simeq P \, (Y_\mu \, P)$ and $Y_\nu \, P \simeq P \, (Y_\nu \, P)$. The typing derivation $\Upsilon_{[M]_\infty}$ is defined similarly. The boundary $\text{Þ}_{[M]_\infty}$ is also defined by structural induction on M. The two important cases by induction are the definition of the boundary of a $\lambda Y_{\mu\nu}$-term of the form $Y_\mu M$ or $Y_\nu M$. The $\lambda Y_{\mu\nu}$-terms $Y_\mu M$ and $Y_\nu M$ are unfolded as the same infinitary λ-term $[Y_\mu M]_\infty = [Y_\nu M]_\infty$ depicted as



(13)

where $[M]_\infty$ denotes the infinitary unfolding of M. The boundary of $[Y_\mu M]_\infty$ consists of all the infinite paths $\mathbb{p}$ of the λ-term (13) which start by exploring a finite number of application nodes depicted in (13), and then enter in one copy of $[M]_\infty$ and implement in it an infinite path $\mathbb{q} \in \text{Þ}_{[M]_\infty}$ of the boundary of $[M]_\infty$. The boundary of $[Y_\nu M]_\infty$ consists of the boundary of $[Y_\mu M]_\infty$ extended with the infinite path which explores an infinite number of application nodes depicted in (13) without ever entering into any copy of $[M]_\infty$. Since the three other induction cases Var, *Abs* and *App* are straighforward, this defines for every simply-typed $\lambda Y_{\mu\nu}$-term M the boundary $\text{Þ}_{[M]_\infty}$ of the infinitary λ-term $[M]_\infty$ associated to M. Although it is clear at this stage, it is worth mentioning that the inductive or coinductive nature of the fixpoint operators Y_μ and Y_ν is not detected by the infinitary λ-term $[M]_\infty$ itself, but by its boundary $\text{Þ}_{[M]_\infty}$.

Now, suppose given a strongly convergent β-rewriting path $f : M \twoheadrightarrow_\beta N$. By definition of strong convergence, there exists for every $n \in \mathbb{N}$ a natural number $\mathsf{N}(n)$ such that the path $f_{|p,q}$ is of diameter $||f_{|p,q}||$ strictly smaller than $\varepsilon = 1/2^n$ for every $p, q \in \mathbb{N}$ such that $\mathsf{N}(n) < p \leq q$. This leads us to the following definition

Definition 9: An infinite path $\mathbb{p} \in \infty\text{-}\mathsf{path}(M)$ is called *necessary* to an infinite path $\mathbb{q} \in \infty\text{-}\mathsf{path}(N)$ along $f : M \twoheadrightarrow_\beta N$ when for every $m \in \mathbb{N}$, there exists $n \in \mathbb{N}$ such that the occurrence $\mathbb{p}_{|m}$ is necessary (in the traditional sense) to the computation of the occurrence $\mathbb{q}_{|n}$ along the finite β-rewriting path $f_{|0,\mathsf{N}(n)}$ prefix of length $\mathsf{N}(n)$ of f.

From this follows the important observation that

Proposition 6: Every strongly convergent β-rewriting path $f : M \twoheadrightarrow_\beta N$ starting from a simply-typed λ^∞-term M with boundary $\mathrm{Þ}_M$ induces a boundary $\mathrm{Þ}_N$ on the simply-typed λ^∞-term N.

The boundary $\mathrm{Þ}_N$ is defined as the set of all the infinite paths $\mathbb{q} \in \infty\text{-}\mathsf{path}(N)$ such that every infinite path $\mathbb{p} \in \infty\text{-}\mathsf{path}(M)$ necessary to $\mathbb{q}$ along $f : M \twoheadrightarrow_\beta N$ is an element $\mathbb{p} \in \mathrm{Þ}_M$ of the original boundary $\mathrm{Þ}_M$. This construction applies in particular to the strongly convergent β-rewriting path $f : M \twoheadrightarrow_\beta N$ which computes the Böhm tree $N = BT(M)$ generated by the unfolding $[M]_\infty$ of a simply-typed $\lambda Y_{\mu\nu}$-term M. This defines the boundary of the simply-typed λ^∞-term $N = BT(M)$ generated by a simply-typed $\lambda Y_{\mu\nu}$-term M. An essentially straightforward adaption of the techniques developed in §IV, §V, §VI to the $\lambda Y_{\mu\nu}$-calculus and to the λ^∞-terms with boundary enables us to refine Thm. 6 and to establish the following decidability theorem:

Theorem 7 (Decidability): Suppose given a higher-order automaton $\mathcal{A}$ and a simply-typed $\lambda Y_{\mu\nu}$-term M of same higher-order alphabet $\mathbf{\Sigma}$ and of same type A. Then, the question whether the infinitary simply-typed λ-term with boundary $N = BT(M)$ generated by M is accepted by $\mathcal{A}$ is decidable.

VIII. Higher-order parity automata

In this section, we deduce the decidability theorem for λY-terms and higher-order parity automata from the decidability theorem (Thm. 7) just established in §VII for $\lambda Y_{\mu\nu}$-terms and higher-order automata. To that purpose, we follow a translation $[\![-]\!]$ from the simply-typed λ-calculus into itself, formulated for the first time in [7]. Suppose given a set of parities $\Omega = \{1, \ldots, k\}$, and define the $\square$-modality as follows:

$$\square A \quad = \quad \underbrace{(A \times \ldots \times A)}_{k\text{-fold product}} \tag{14}$$

The translation $[\![-]\!]$ transports every typing judgment $\mathbf{\Sigma} \vdash M : A$ into a typing judgment $\square [\![\mathbf{\Sigma}]\!] \vdash [\![M]\!] : [\![A]\!]$ where the translation $[\![-]\!]$ on simple types is defined as

$$\begin{array}{rclcrcl} [\![\mathbf{1}]\!] & = & \mathbf{1} & \quad & [\![A \times B]\!] & = & [\![A]\!] \times [\![B]\!] \\ [\![\mathrm{o}]\!] & = & \mathrm{o} & & [\![A \Rightarrow B]\!] & = & (\square [\![A]\!]) \Rightarrow [\![B]\!] \end{array}$$

and the translation $[\![-]\!]$ on λ-terms applies the recipe of the translation of a simply-typed λ-term in a model of linear logic where $\times$ denotes *at the same time* the tensor product $\otimes$ and the cartesian product $\&$ of the free cartesian-closed category generated by o, and the exponential modality ! is defined as the modality $\square$. Note that in this purely syntactic translation, the comonadic structure of $\square$ is provided by two maps

$$\delta_A : \square A \longrightarrow \square\square A \qquad\qquad \varepsilon_A : \square A \longrightarrow A$$

defined as the simply-typed λ-terms δ_A and ε_A below:

$$\begin{array}{lcc} (a_1, \ldots, a_k) : \square A & \vdash & \Big(\, (a_{\mathsf{max}(i,j)})_{1 \leq i \leq k} \, \Big)_{1 \leq j \leq k} : \square\square A \\ (a_1, \ldots, a_k) : \square A & \vdash & a_1 : A \end{array}$$

Note that the max operator appearing in the definition of δ_A coincides with the max operator used in the computation of the parity of a branch in the higher-order automaton ; while the definition of ε_A as the projection on the variable a_1 reflects the fact that the parity 1 is the neutral element of max in the set of parities Ω, see [5], [6], [7] for details. Then, we take advantage of the elementary fact that every simply-typed λY-term M is $\beta\eta$-equivalent to a λY-term of the form $Y(\lambda F.P)$, where P is a simply-typed λ-term, not containing any fixpoint operator. The translation of such a λ-term P of alphabet and type

$$F : A \ , \ \mathbf{\Sigma} \quad \vdash \quad P : A$$

defines a simply-typed λ-term $[\![P]\!]$ of alphabet and type

$$F_1 : A \, , \, F_2 : A \, , \, \ldots \, , \, F_k : A \ , \ \square [\![\mathbf{\Sigma}]\!] \quad \vdash \quad [\![P]\!] : [\![A]\!]$$

where we use implicitly the definition of the modality $\square$ in (14). We may suppose without loss of generality that the highest parity k is odd, and construct the $\lambda Y_{\mu\nu}$-term:

$$N \quad = \quad \underbrace{Y_\mu Y_\nu Y_\mu Y_\nu \ldots Y_\nu Y_\mu}_{k \text{ fixpoints}} (\, [\![P]\!] \,)$$

We have just explained how to translate a λY-term M into a $\lambda Y_{\mu\nu}$-term N whose structure reflects the inductive and coinductive nature of parities in Ω. Let us turn on the other side of higher-order automata, and make the following key observation:

Proposition 7: A higher-order parity automaton $\mathcal{A}$ with higher-order alphabet $\mathbf{\Sigma}$ and type A is the same thing as a higher-order automaton $[\![\mathcal{A}]\!]$ with higher-order alphabet $[\![\mathbf{\Sigma}]\!]$ and type $[\![A]\!]$.

Moreover, the acceptance of the simply-typed λY-term M by a higher-order parity automaton $\mathcal{A}$ is equivalent to the acceptance of the $\lambda Y_{\mu\nu}$-term N defined above by the higher-order automaton $[\![\mathcal{A}]\!]$. The decidability theorem (Thm. 1) follows immediately from this, and the decidability theorem established in §VII.

IX. Related works

The model-checking problem discussed in the introduction was originally established by Knapik, Niwinski and Urzyczyn [16] for safe higher-order recursion schemes. The safety condition on recursion schemes was then relaxed in subsequent works, using a large variety of approaches and techniques: game semantics [19], intersection types [18], collapsible pushdown automata [15] or Krivine environment machines [21].

A second generation of proofs then emerged, based on a tight connection with denotational semantics, and with earlier ideas by Aehlig [1] and Salvati [20] on the relationship between language recognizability and finite models of the simply-typed λ-calculus. Two research groups developed simultaneously this denotational reconstruction of higher-order model-checking, with a series of mutual influences and crossed inspirations: Salvati and Walukiewicz [22], [23] and Grellois and Melliès [5], [6], [7], [8]. Looking in retrospect, the discovery of a connection between higher-order model-checking and linear logic played a decisive role in the success of the denotational approach, see [23], [7] for a discussion. This connection with linear logic is not accidental, and relies on a series of foundational works by Ehrhard [3] and Terui [24] on the Scott semantics of linear logic. The work presented here combines this denotational approach with fundamental ideas in infinitary rewriting theory [11], [4], most notably strong convergence, diffraction patterns and infinitary λ-terms with boundary.

The notion of higher-order parity automaton was presented for the first time during a talk at IHP in June 2014, more than two years and a half ago. The reason for delaying the publication is that we wanted a clean and transparent proof of decidability. Although we focus here on the "local" model-checking problem, we believe that the automata-theoretic approach to the "global" model-checking selection problem elaborated by Haddad [14] can be adapted to higher-order parity automata, as was done in [5] for traditional (first-order) automata. Finally, a recent translation [2] between higher-order model-checking and higher-order modal fixpoint logic [25] seems to share a number of primary ingredients with [5], [6], [7] and with our proof of decidability. A comparison would deserve further investigations.

X. Conclusion

We have introduced a notion of higher-order parity automaton designed to express inductive and coinductive properties of simply-typed infinitary λ-terms. We have then justified our notion of automaton by establishing a general decidability theorem. The theorem extends the scope of traditional higher-order model-checking from alphabets with letters of first-order arities, to alphabets with letters of arbitrary higher-order arities. One main technical contribution of the paper is to articulate a simple and conceptually rigorous proof of decidability, combining ideas coming from denotational semantics, linear logic and infinitary rewriting. In particular, the notion of diffraction pattern introduced in the paper is interesting for its own sake, and would deserve further study.

Acknowledgments.

Many ideas developed in the present paper emerged in my work with Charles Grellois on linear logic and higher-order model-checking, and I thus want to thank him warmly here. I am also grateful to Arnaud Carayol, Thomas Colcombet, Etienne Lozes, Sylvain Salvati, Olivier Serre and Igor Walukiewicz for discussions and feedbacks on this work.

References

[1] Klaus Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007.
[2] Florian Bruse, Naoki Kobayashi, Étienne Lozes. On the Relationship between Higher-Order Recursion Schemes and Higher-Order Fixpoint Logic. Proceedings of POPL 2017.
[3] Thomas Ehrhard. The Scott model of linear logic is the extensional collapse of its relational model. *Theor. Comput. Sci.*, 424:20–45, 2012.
[4] Jörg Endrullis, Dimitri Hendriks, Jan Willem Klop. *Highlights in Infinitary Rewriting and Lambda Calculus.* Theoretical Computer Science, Volume 464, 2012, Pages 48-71.
[5] Charles Grellois and Paul-André Melliès. An infinitary model of linear logic. Proceedings of FOSSACS 2015.
[6] Charles Grellois and Paul-André Melliès. Relational semantics of linear logic and higher-order model checking. Proceedings of CSL 2015.
[7] Charles Grellois and Paul-André Melliès. Finitary semantics of linear logic and higher-order model-checking. Proceedings of MFCS 2015.
[8] Charles Grellois. Semantics of linear logic and higher-order model-checking. PhD Thesis, Université Paris Diderot, April 2016.
[9] Alexander S. Kechris. Classical Descriptive Set Theory. Graduate Texts in Mathematics, Springer Verlag, 1995.
[10] Richard Kennaway and Fer-Jan de Vries. *Infinitary Lambda Calculus.* Theoretical Computer Science, 175(1):93Ű125, 1997.
[11] Richard Kennaway, Jan Willem Klop, Ronan Sleep, and Fer-Jan de Vries. *Infinitary Lambda Calculus.* Theoretical Computer Science, 175(1):93Ű125, 1997.
[12] Richard Kennaway, Vincent van Oostrom, Fer-Jan de Vries. Meaningless terms in rewriting, The Journal of Functional and Logic Programming 1 (1999).
[13] Alexander Kurz, Daniela Petrisan, Paula Severi, Fer-Jan de Vries. An alpha-corecursion principle for the infinitary lambda calculus, in: Proc. 11th Int. Workshop on Coalgebraic Methods in Computer Science, Springer, 2012.
[14] Axel Haddad. *Shape-preserving transformations of higher-order recursion schemes*. PhD thesis, Université Paris Diderot, 2013.
[15] Matthew Hague, Andrzej S. Murawski, Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, 2008.
[16] Teodor Knapik, Damian Niwinski, Pawel Urzyczyn. Higher-order pushdown trees are easy. Proceedings of FoSSaCS. LNCS, vol. 2303 (2002)
[17] Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In Zhong Shao and Benjamin C. Pierce, editors, *POPL*, 2009.
[18] Naoki Kobayashi and Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *LICS*, 2009.
[19] Luke Ong. On model-checking trees generated by higher-order recursion schemes. Proceedings of *LICS*, pages 81–90, 2006.
[20] Sylvain Salvati. Recognizability in the Simply Typed Lambda-Calculus Logic. Proceedings of Language, Information and Computation, 16th International Workshop, WoLLIC 2009.
[21] Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. In Proceedings of ICALP 2011, LNCS vol. 6756, pages 162–173. Springer, 2011.
[22] Sylvain Salvati and Igor Walukiewicz. Using models to model-check recursive schemes. In *TLCA*, LNCS vol. 7941. Springer 2013.
[23] Sylvain Salvati and Igor Walukiewicz. A model for behavioural properties of higher-order programs. In Proceedings of CSL 2015, volume 41 of LIPIcs, pages 229–243.
[24] Kazushige Terui. Semantic evaluation, intersection types and complexity of simply typed lambda calculus. In *RTA*, volume 15 of *LIPIcs*, 2012.
[25] M. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In Proceedings of CONCUR 2004, volume 3170 of Lecture Notes in Computer Science, pages 512-528. Springer, 2004.

APPENDIX

A. A glimpse on the expressive power of higher-order parity automata

Although we will not study here their expressive power we would like to illustrate with a very elementary example how the notion of higher-order parity automaton extends the traditional boundaries of higher-order model-checking. Consider the higher-order alphabet $\mathbf{\Sigma}$ consisting of a unique letter cons of higher-order arity

$$\mathsf{cons} : \omega \Rightarrow ⦶ \Rightarrow ⦶$$

where $\omega = (⦶ \Rightarrow ⦶) \Rightarrow ⦶ \Rightarrow ⦶$ denotes the simple type of natural numbers used in the Church encoding. An infinitary λ-term in normal form of type $⦶$ in the higher-order alphabet $\mathbf{\Sigma}$ is the same thing as an infinite stream

$$[\, n_0, n_1, n_2, \ldots] \quad = \quad \mathsf{cons}(n_0, \mathsf{cons}(n_1, \mathsf{cons}(n_2, \ldots)))$$

of completed natural numbers $n_k \in \mathbb{N} \uplus \{\infty\}$. Every higher-order parity automaton $\mathcal{A}$ of alphabet $\mathbf{\Sigma}$ and of type $⦶$ thus describes a specific infinitary property of these streams. When instantiated at the higher-order alphabet $\mathbf{\Sigma}$, the main result of the paper (Thm. 1) establishes that for every finite λY-term M of type

$$\mathsf{cons} : \omega \Rightarrow ⦶ \Rightarrow ⦶ \quad \vdash \quad M : ⦶$$

the question whether M generates an infinite stream $N = BT(M)$ of completed natural numbers accepted by the automaton $\mathcal{A}$ is decidable.

B. Run-trees

One main novelty of our notion of higher-order automaton with respect to the traditional notion of *alternating tree automaton* is that the alphabet $\mathbf{\Sigma}$ and transition function δ of the automaton may be altered (and in fact extended) in the course of the exploration of the simply-typed λ^{∞}-term. This typically happens when a higher-order automaton with alphabet $\mathbf{\Sigma}$, transition function δ and initial state $q_0 = \{q_1, \ldots, q_n\} \multimap q$ reaches a simply-typed λ^{∞}-term of the form

$$\mathbf{\Sigma} \vdash \lambda a.M : A \Rightarrow B.$$

Recall that the purpose of the abstraction node $\lambda a.M$ is to declare a new variable in the context, using the Abstraction rule:

$$\text{Abstraction} \quad \frac{\mathbf{\Sigma}, a : A \vdash M : B}{\mathbf{\Sigma} \vdash \lambda a.M : A \Rightarrow B}$$

The higher-order automaton should thus behave accordingly and declare a new letter a in its alphabet in order to adapt to the situation. This extra letter a should come together with an extension of the transition function δ with

$$\delta(a) \quad = \quad \{q_1, \ldots, q_n\} \quad \in \quad \mathcal{P}_{\text{fin}}(Q_A).$$

This ability to extend the alphabet in the course of exploration of the simply-typed λ^{∞}-term makes the notion of *run-tree* more sophisticated but also more expressive than in the usual notion of run-tree in an alternating tree automaton. We find convenient to give a coinductive definition of the set of run-trees of an automaton $\mathcal{A} = (Q, \mathbf{\Sigma}, A, \delta, q_0)$ against a simply-typed λ^{∞}-term $\mathbf{\Sigma} \vdash M : A$ of context $\mathbf{\Sigma}$ and of type A. By definition, the set of run-trees is only defined when:

- the context $\mathbf{\Sigma}$ of the simply-typed λ^{∞}-tree M coincides with the alphabet $\mathbf{\Sigma}$ of the automaton $\mathcal{A}$,
- the type A of the simply-typed λ^{∞}-tree M coincides with the type A of the automaton $\mathcal{A}$.

For that reason, we generally find convenient to write

$$\langle\, \mathbf{\Sigma} \vdash M : A \,|\, \delta, q \,\rangle$$

for the set of run-trees of the higher-order automaton $\mathcal{A} = (Q, \mathbf{\Sigma}, A, \delta, q)$ against the infinitary simply-typed λ-term $\mathbf{\Sigma} \vdash M : A$. This notation makes sense because the preordered set of ground states $(Q, \sqsubseteq_Q)$ remains fixed in the course of the definition of the run-tree.

1) Abstraction: Suppose given a simply-typed λ^{∞}-term

$$\mathbf{\Sigma} \vdash \lambda a.M : A \Rightarrow B$$

and an automaton $\mathcal{A} = (Q, \mathbf{\Sigma}, \delta, q_0)$ of same type $A \Rightarrow B$. By definition, the initial state q_0 of the automaton $\mathcal{A}$ is a higher-order state of type $A \Rightarrow B$, and it is thus of the form

$$q_0 \;=\; \{q_1, \ldots, q_n\} \multimap q$$

where the q_i's are higher-order states of type A and q is a higher-order state of type B. The set of run-trees

$$\langle\, \mathbf{\Sigma} \vdash \lambda a.M : A \Rightarrow B \,|\, \delta, \{q_1, \ldots, q_n\} \multimap q \,\rangle \qquad (15)$$

is defined coinductively as

$$(\mathbf{Lam}, q_0) \cdot \langle\ \mathbf{\Sigma}, a : A \vdash M \ |\ \delta + a \mapsto \{q_1, \ldots, q_n\}\ ,\ q\ \rangle$$

The intuition behind this definition is that a run-tree exploring $\lambda a.M$ with initial state $\{q_1, \ldots, q_n\} \multimap q$ starts by declaring the letter a with transition function $\delta(a) = \{q_1, \ldots, q_n\}$, and then explores the simply-typed λ^{∞}-term M with initial state q. Note that every state q_i is of type A and that the state q is of type B. This ensures that our coinductive definition of (15) is valid. Finally, the notation $(\mathbf{Lam}, q) \cdot Z$ stands for the set $\{(\mathbf{Lam}, q)\} \times Z$ where $\mathbf{Lam}$ is a tag, q is a state and Z is a set ; this addition to the bare construction is only here to label the run-trees in a clear and non-equivocal way.

2) Application: Suppose given a simply-typed λ^{∞}-term

$$\mathbf{\Sigma} \;\vdash\; \mathit{App}(M, N) \;:\; B$$

whose underlying typing derivation $\Upsilon_{\mathit{App}(M,N)}$ has last rule

$$\text{Application} \quad \frac{\mathbf{\Sigma} \vdash M : A \Rightarrow B \qquad \mathbf{\Sigma} \vdash N : A}{\mathbf{\Sigma} \vdash \mathit{App}(M, N) : B}$$

Suppose that $\mathcal{A} = (Q, \mathbf{\Sigma}, \delta, q)$ is an automaton of same alphabet $\mathbf{\Sigma}$ and of same type B. The set of run-trees

$$\langle\ \mathbf{\Sigma} \;\vdash\; \mathit{App}(M, N) : B \ |\ \delta\ ,\ q\ \rangle \qquad (16)$$

is defined coinductively as the set

$$\coprod_{\substack{u \in \mathcal{P}_{\mathrm{fin}}(Q_A) \\ u = \{q_1, \ldots, q_n\}}} (\mathbf{App}, u \multimap q) \cdot \langle\, \mathbf{\Sigma} \vdash M \,|\, \delta, u \multimap q \,\rangle \times \prod_{i=1}^{i=n} \langle\, \mathbf{\Sigma} \vdash N \,|\, \delta, q_i \,\rangle$$

The intuition behind this definition is that a run-tree exploring $App(M, N)$ with initial state q is provided by a run-tree exploring the simply-typed λ^∞-term M with initial state $u \multimap q$ for some finite set $u = \{q_1, \ldots, q_n\}$ of states $q_i \in Q_A$, together with a vector of n run-trees exploring the simply-typed λ^∞-term N, each run-tree starting with the initial state $q_i \in Q_A$, for $1 \leq i \leq n$. Note the state $u \multimap q$ has the same type $A \Rightarrow B$ as the function M, and that each state $q_i \in Q_A$ has the same type A as the argument N. This ensures that our coinductive definition of (16) is valid. The notation $(\mathbf{App}, q) \cdot Z$ stands for the set $\{(\mathbf{App}, q)\} \times Z$ where **App** is a tag, q is a state and Z is a set.

3) Variable: Suppose given a simply-typed λ^∞-term

$$\mathbf{\Sigma}\,,\, a : A \,\vdash\, a \,:\, A$$

with underlying typing derivation Υ_a is defined by a *Variable* rule. Suppose given an automaton $\mathcal{A} = (Q, \mathbf{\Sigma}, \delta, q)$ of the same alphabet and of same type A. In that case, the set of run-trees of the automaton is defined as

$$\langle\, \mathbf{\Sigma}\,,\, a : A \,\vdash\, a : A \,|\, \delta + a \mapsto \{q_1, \ldots, q_n\}, q \,\rangle = (\mathbf{Var}, q)$$

when $q \in Q_A$ satisfies

$$q \sqsubseteq_A q_i$$

for an element $q_i \in Q_A$ of the finite set $\{q_1, \ldots, q_n\}$, and

$$\langle\, \mathbf{\Sigma}\,,\, a : A \,\vdash\, a : A \,|\, \delta + a \mapsto \{q_1, \ldots, q_n\}, q \,\rangle = \emptyset$$

otherwise, when q is not an element of $\{q_1, \ldots, q_n\}$. Here, the notation $(\mathbf{Var}, q)$ stands for the singleton set $\{(\mathbf{Var}, q)\}$ where **Var** is a tag and q is a state.

This concludes our coinductive definition of the set of run-trees of an automaton $\mathcal{A} = (Q, \mathbf{\Sigma}, A, \delta, q)$ against an infinitary and simply-typed λ-term $\mathbf{\Sigma} \vdash M : A$ of same higher-order alphabet $\mathbf{\Sigma}$ and of same higher-order arity A.

C. Run-trees as an intersection type system

The two figures (Fig. 3 and Fig. 2) illustrate why it is important to equip the deduction system of §III-C (Def. 2) with a subtyping relation $q \sqsubseteq_A q'$ in the *Var* rule

$$\text{Var} \quad \frac{q \sqsubseteq_A q' \qquad q' \in \delta(a)}{\langle\, \mathbf{\Sigma}, a : A \vdash a : A \,|\, \delta, q \,\rangle}$$

By way of illustration, this subtyping relation $q \sqsubseteq_A q'$ appearing in the *Var* rule ensures that the automaton $\mathcal{A}$ with empty alphabet, with type

$$(A \Rightarrow B) \Rightarrow A \Rightarrow B$$

and with initial state

$$\{\{\,\} \multimap q_B\} \multimap \{q_A\} \multimap q_B$$

accepts the simply-typed λ-term

$$\vdash \lambda f.f : (A \Rightarrow B) \Rightarrow A \Rightarrow B$$

in the same way as it accepts without the need of any explicit subtyping relation its η-expansion, the simply-typed λ-term

$$\vdash \lambda f.\lambda a.App(f, a)$$

The subtyping relation $q \sqsubseteq_A q'$ appearing in the rule *Var* is thus necessary to have a typing system invariant under η-conversion, even when the preorder $\sqsubseteq = \sqsubseteq_\circ$ on the set Q of ground states is defined as the identity relation:

$$\forall q, q' \in Q, \qquad q \sqsubseteq q' \quad \Longleftrightarrow \quad q = q'.$$

This formulation of the intersection type system for infinitary run-trees (based on subtyping $\sqsubseteq$) comes from a fundamental connection between higher-order model-checking and linear logic, discovered and developed in [5], [6], [7]. This formulation which improves the intersection type systems originally formulated without subtyping in [17], [18] has been recently adopted in [2].

D. Diffraction patterns in run-trees

In our proof of the backward preservation theorem (Thm. 5) in §VI, we use the notion of diffraction pattern

$$D \in \mathsf{diff}(\mathcal{R})$$

living in a run-tree $\mathcal{R}$ of an automaton $\mathcal{A}$ on a simply-typed λ^∞-term M. The notion of diffraction pattern in a run-tree

$$\mathcal{R} \in \langle\, \mathbf{\Sigma} \vdash M : B \,|\, \delta, q \,\rangle \tag{17}$$

is defined in just the same way as the notion of diffraction pattern in a simply-typed λ^∞-term

$$\mathbf{\Sigma} \vdash M : B$$

using a structural induction on the type A of the diffraction pattern D. In particular, every diffraction pattern $D \in \mathsf{diff}(\mathcal{R})$ comes equipped with a monotone function

$$\mathsf{embed}_D \quad : \quad (\mathsf{shape}(D), \preceq_D) \quad \longrightarrow \quad (\mathsf{occ}(\mathcal{R}), \preceq_\mathcal{R})$$

which transports every node of the rooted tree $\mathsf{shape}(D)$ to its occurrence in the run-tree $\mathcal{R}$. A remarkable fact is that every run-tree $\mathcal{R}$ in (17) induces a residual relation

$$[\,\mathcal{R}\,] \quad \subseteq \quad \mathsf{diff}(M) \times \mathsf{diff}(\mathcal{R})$$

which relates a diffraction pattern $D \in \mathsf{diff}(M)$ to its residuals in $\mathsf{diff}(\mathcal{R})$, defined in the expected way. Note that every residual $E \in \mathsf{diff}(\mathcal{R})$ of the diffraction pattern $D \in \mathsf{diff}(M)$ has the same underlying rooted tree

$$\mathsf{shape}(E) = \mathsf{shape}(D).$$

Moveover, the embedding of a node in $\mathsf{shape}(E)$ is an occurrence of $\mathcal{R}$ above the embedding of the corresponding node in $\mathsf{shape}(D)$.

$$\dfrac{\dfrac{\{q_A\} \multimap q_B \sqsubseteq_{A\Rightarrow B} \{\,\} \multimap q_B \qquad \{\,\}\multimap q_B \in \delta(f)}{\langle\, f : A\Rightarrow B \vdash f : A \Rightarrow B \mid f \mapsto \{\{\,\}\multimap q_B\}, \{q_A\}\multimap q_B\,\rangle}\ \mathrm{Var}}{\langle \vdash \lambda f.f : (A\Rightarrow B)\Rightarrow A\Rightarrow B \mid \{\{\,\}\multimap q_B\}\multimap\{q_A\}\multimap q_B\,\rangle}\ \mathrm{Abs}$$

Fig. 2. Run-tree exploring the typing judgement $\vdash \lambda f.f : (A \Rightarrow B) \Rightarrow A \Rightarrow B$

$$\dfrac{\dfrac{\dfrac{\dfrac{\{\,\}\multimap q_B \in \delta(f)}{\langle\, f : A\Rightarrow B, a : A \vdash f : A\Rightarrow B \mid f\mapsto\{\{\,\}\multimap q_B\}, a\mapsto\{q_A\}, \{\,\}\multimap q_B\,\rangle}\ \mathrm{Var} \qquad \text{empty set of derivations}}{\langle\, f : A\Rightarrow B, a : A \vdash App(f,a) : B \mid f\mapsto\{\{\,\}\multimap q_B\}, a\mapsto\{q_A\}, q_B\,\rangle}\ \mathrm{App}}{\langle\, f : A\Rightarrow B \vdash \lambda a.App(f,a) : A\Rightarrow B \mid f\mapsto\{\{\,\}\multimap q_B\}, \{q_A\}\multimap q_B\,\rangle}\ \mathrm{Abs}}{\langle \vdash \lambda f.\lambda a.App(f,a) : (A\Rightarrow B)\Rightarrow A\Rightarrow B \mid \{\{\,\}\multimap q_B\}\multimap\{q_A\}\multimap q_B\,\rangle}\ \mathrm{Abs}$$

Fig. 3. Run-tree exploring the η-expansion of the typing judgement $\vdash \lambda f.f : (A \Rightarrow B) \Rightarrow A \Rightarrow B$

E. Residuals of diffraction patterns

Suppose given a β-redex $R = (M, o, N)$ between simply-typed λ^{∞}-terms M and N. The residual relation between diffraction patterns

$$[R] \subseteq \mathsf{diff}(M) \times \mathsf{diff}(N)$$

along the β-redex R is defined by induction on the type of the diffraction pattern $D \in \mathsf{diff}(M)$.

Definition 10 (residual of ground type): The residual of a diffraction pattern $D = o_D$ of type $\mathbb{o}$ is any residual E of the occurrence o_D along R.

Definition 11 (residual of arrow type, occurrence case): A residual $E \in \mathsf{diff}(N)$ of a diffraction pattern $D \in \mathsf{diff}(M)$ defined as an occurrence $D = o_D$ of type $A \Rightarrow B$ in M is either defined as

- an occurrence residual $o_E \in \mathsf{occ}(N)$ of the occurrence $o_D \in \mathsf{occ}(M)$ along the β-redex R,
- when the occurrence o of the β-redex coincides with the occurrence $D = o_D$ of the diffraction pattern, then D has a unique residual E defined as $E = \{D_{A,i} \mid i \in I\} \multimap D_B$ where $D_B = o_B$ is the unique occurrence of type B in N such that $o = o_B \cdot \mathsf{body}$, where I denotes the countable (finite or infinite) set of occurrences of the free variable a in M, and where each $D_{A,i}$ denotes the occurrence of the root of one copy of the λ-term Q in N.

Note that the second case of Def. 11 corresponds to the illustration given at the beginning of the section.

Definition 12 (residual of arrow type, arrow case): The residuals of a diffraction pattern D of type $A \Rightarrow B$ defined as a pair $D = \{D_{A,i} \mid i \in I\} \multimap D_B$ along a β-redex R are defined as

- the copies E of the diffraction pattern D when the head occurrence $\mathsf{hdocc}(D)$ is duplicated by the β-redex R,
- otherwise, the unique residual $E = \{E_{A,i} \mid j \in J\} \multimap E_B$ where E_B is the unique residual of D_B along R, and where $\{E_{A,i} \mid j \in J\}$ denotes the set of residuals of the diffraction patterns $D_{A,i}$ along R, for $i \in I$.

The residual relation

$$[f] \subseteq \mathsf{diff}(M) \times \mathsf{diff}(N)$$

associated to a finite β-rewriting path

$$f \;=\; M \xrightarrow{R_0} M_1 \xrightarrow{R_1} \cdots \longrightarrow M_{p-1} \xrightarrow{R_{p-1}} N$$

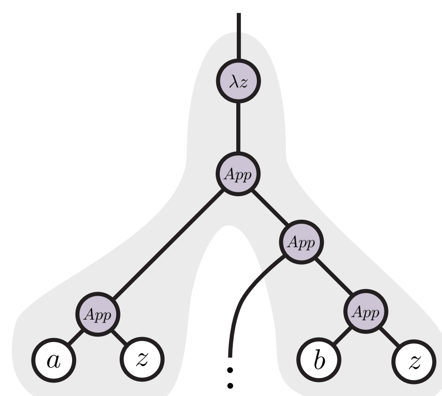
is simply defined by composing the residual relations together $[f] = [R_0] \circ \cdots \circ [R_{p-1}]$

F. An illustration of diffraction patterns

The encoding of the higher-order recursion scheme $\mathcal{G}$ in (8) can be simplified, and replaced by the simply-typed λY-term below

$$M \;=\; \Big(Y\big[\,\lambda F.\lambda z.\, a\, z\,(\,F\,(\,b\,z\,)\,)\big]\Big)\, c \qquad (18)$$

where the functional F is of type $\mathbb{o} \Rightarrow \mathbb{o}$. We use that λY-term M to illustrate the notion of diffraction pattern developed in §V. The λY-term may be unfolded into a simply-typed λ-term $[M]_{\infty}$ obtained by plugging the context

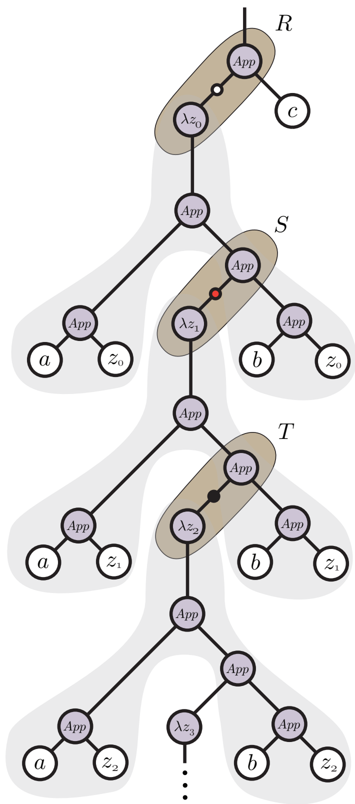


into itself, coinductively. Note that the letters a, b are treated in our diagrams as variables of first-order arities:

$$a : \mathbb{o} \Rightarrow \mathbb{o} \Rightarrow \mathbb{o} \qquad\qquad b : \mathbb{o} \Rightarrow \mathbb{o}$$

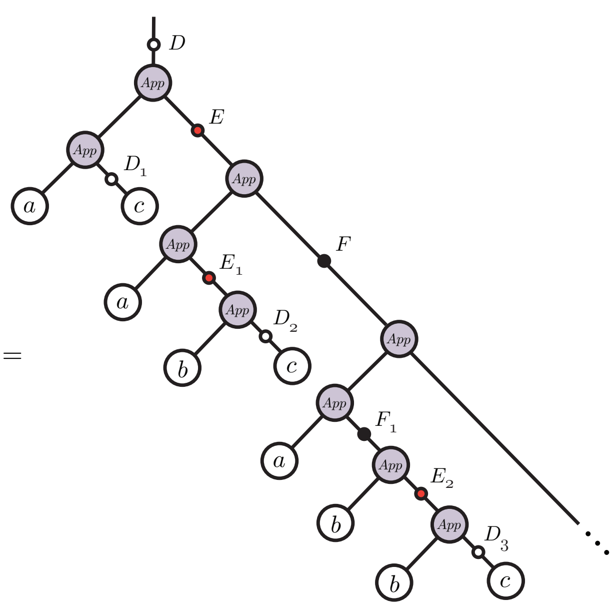
which behave as functions, and thus do have any child in the λ-term. One obtains in that way the simply-typed infinitary λ-term $[M]_{\infty}$ below. In this infinitary λ-term $[M]_{\infty}$, we encircle the three β-redexes R, S and T within a brown area:

$[M]_\infty \quad =$

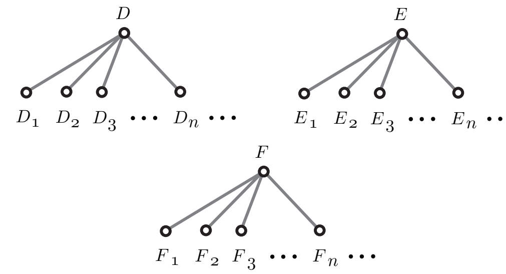


For notational convenience, we identify from now on $[M]_\infty$ with the original λY-term M. We also indicate in the picture the occurrences o_R, o_S and o_T of the three β-redexes $R = (M, o_R, N_R)$, $S = (M, o_S, N_S)$ and $T = (M, o_T, N_T)$ with a white circle, a red circle and a black circle, respectively. The Böhm tree $N = BT(M)$ generated by M coincides with the Church encoding of the infinite tree $\langle \mathcal{G} \rangle$ described in the introduction, and depicted in (2). The diffraction patterns D, E and F residual in the infinitary λ-term N of the occurrences o_R, o_S and o_T in M are represented in the picture using white circles, red circles and black circles, respectively.

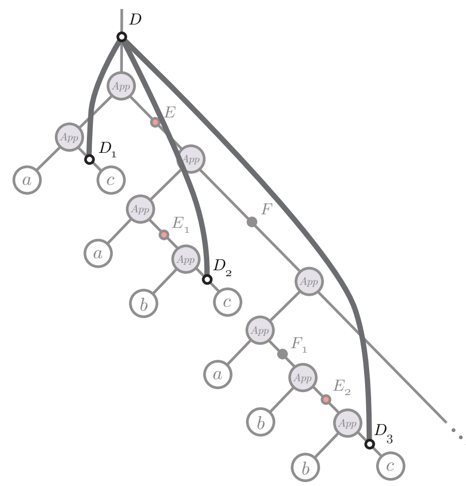
$N \quad =$



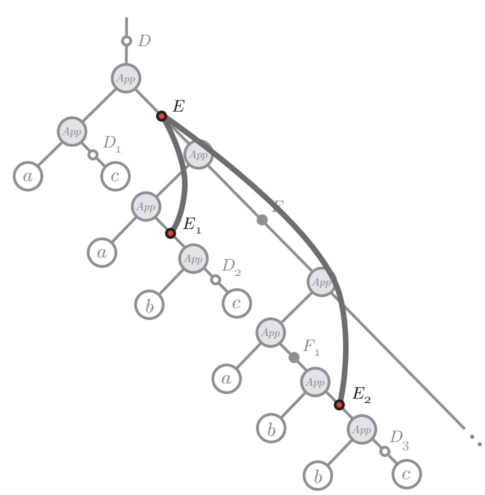
Note that the three diffraction patterns D, E and F have the same infinitary shape



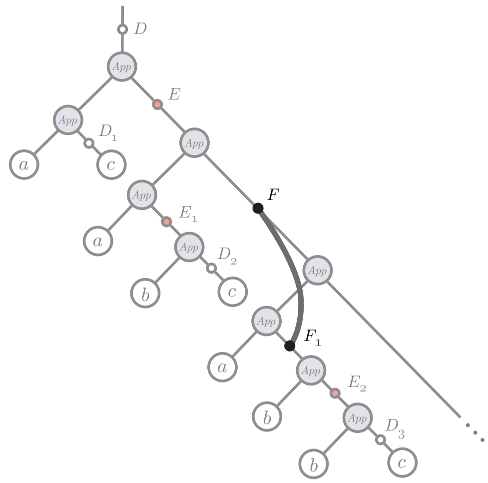
Typically, the diffraction pattern $D \in \mathsf{diff}(N)$ residual in $N = BT(M)$ of the occurrence $o_R \in \mathsf{diff}(M)$ is depicted by attaching every node x of the rooted tree $\mathsf{shape}(D)$ to its occurrence $\mathsf{embed}_D(x) \in \mathsf{occ}(N)$ in the infinitary λ-term N, as follows:



In the same way, the diffraction pattern E residual in $N = BT(M)$ of the occurrence o_S of the β-redex S in M is represented as follows:



while the diffraction pattern F residual in $N = BT(M)$ of the occurrence o_T of the β-redex T in M is represented as follows:



Note that the diffraction patterns D, E and F indicate very clearly which part each β-redex R, S and T took in the construction of the infinitary λ-term $N = BT(M)$. In particular, we have shown in §VI that this information is sufficient to reconstruct a run-tree over M from a run-tree over N, for every higher-order automaton $\mathcal{A}$ of same higher-order alphabet and type as M and N.

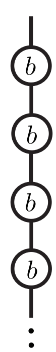
G. Another illustration of diffraction patterns

We illustrate the fact that an occurrence o of M may be very well be erased by an infinitary strongly convergent β-rewriting path $f : M \twoheadrightarrow_\beta N$ although none of the finite prefixes $f_{|0,n}$ of f erases it, see [8] for a discussion. To that purpose, consider the higher-order recursion scheme $\mathcal{G}$ below:

$$\mathcal{G} \;=\; \begin{cases} S & \mapsto \quad F\ b\ c \\ F\ y\ z & \mapsto \quad y\ (F\ z) \end{cases} \tag{19}$$

The higher-order recursion scheme $\mathcal{G}$ generates the following infinite word (filiform tree)

$\langle \mathcal{G} \rangle \quad =$

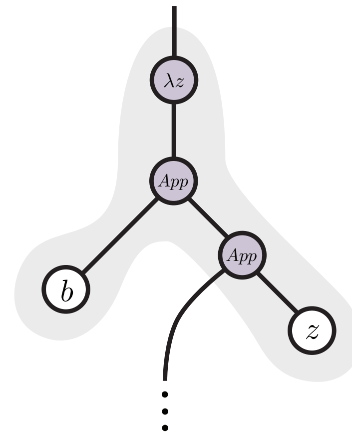


(20)

The higher-order recursion scheme (19) may be encoded as the following λY-term

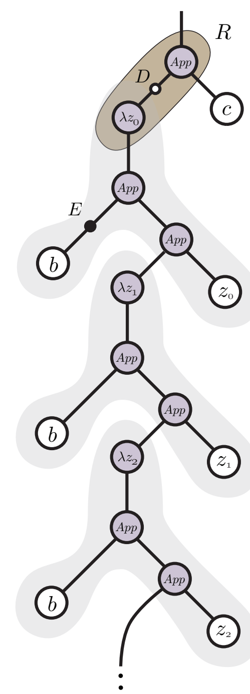
$$M \;=\; \Big(Y \big[\lambda F. \lambda z.\, b\,(\,F\,z\,) \big] \Big)\, c \tag{21}$$

where the functional F is of type $o \Rightarrow o$. The λY-term M may be then unfolded into a simply-typed infinitary λ-term $[M]_\infty$ obtained by plugging the context below into itself, coinductively:



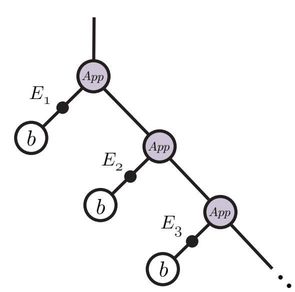
The resulting simply-typed infinitary λ-term $[M]_\infty$ is depicted as follows, where we depict the occurrence $D = o_R$ of the β-redex R with a white circle, and the occurrence E with a black circle:

$[M]_\infty \quad =$



The $[M]_\infty$ rewrites then into the Böhm tree $N = BT(M)$ by a strongly convergent β-rewriting path $f : M \twoheadrightarrow_\beta N$. The generated λ-term N is the Church encoding of the infinitary filiform tree (20) generated by the higher-order recursion scheme $\mathcal{G}$:

$N \quad =$



Note that the occurrence $D = o_R$ in M has no residual in the generated Böhm tree $N = BT(M)$. This reflects the fact that the letter c does not appear in the infinite word or filiform tree (20). On the other hand, the occurrence E in M has an infinite countable set of residual occurrences in N, depicted as the occurrences $E_1, \ldots, E_n, \ldots$ in the graphical description of N above.