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 infinite ranked trees. Our main result is that the acceptance of an infinitary λ-term by a higher-order parity automaton $A$ 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 $\Sigma = \{a : 2, b : 1, c : 0\}$

$$\mathcal{G} = \begin{cases} S & \mapsto F a b c \\ F x y z & \mapsto x z (F x y (y z)) \end{cases}$$

which generates the infinite tree

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

![Tree Diagram](4)

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 $\Sigma$ satisfies the property $(\ast)$ if and only if the ranked tree $\langle \mathcal{G} \rangle$ it generates is accepted by the automaton $A$. Typically, the higher-order recursion scheme $\mathcal{G}$ defined in (1) satisfies the property $(\ast)$ because there exists an accepting run-tree $R$ of the parity tree automaton $A$ on the ranked tree $\langle \mathcal{G} \rangle$, represented below

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

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

on that signature $\Sigma$, the problem $(\ast \ast)$ whether the ranked tree $\langle \mathcal{G} \rangle$ generated by $\mathcal{G}$ is accepted by $A$ 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 $(\ast)$ reduces to the question $(\ast \ast)$ instantiated with the parity tree automaton $A$ on two states $q_0, q_1$ described in (3); from this, it follows that the property $(\ast)$ 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 $A, B \in \text{Type}$ is a binary tree defined by the grammar below:

$$A, B := \mathcal{O} \mid A \times B \mid A \Rightarrow B \mid 1$$
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 \(\lambda\)-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 familiar Church encoding with the infinitary ranked 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 \(\Sigma(a) = A\) the familiar Church encoding of a finite ranked tree as a simply-typed \(\lambda\)-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

\[
\begin{align*}
\Sigma(a) &= (\circ \times \ldots \times \circ) \Rightarrow \circ \\
& \text{k-fold product}
\end{align*}
\]

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

1. the ranked trees of a given signature \(\Sigma\),
2. the simply-typed \(\lambda\)-terms in normal form of the corresponding higher-order alphabet \(\Sigma\).

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 \(\lambda\)-terms \(M\) in normal form of higher-order alphabet \(\Sigma\) and type

\[
a : (\circ \times \circ) \Rightarrow \circ, \quad b : \circ \Rightarrow \circ, \quad c : \circ \vdash \quad M : \circ. \quad (7)
\]

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

\[
M = \left( Y [\lambda F.\lambda x.\lambda y.\lambda z. x z ( F x y ( y z ))] \right) a b c \quad (8)
\]

The infinite tree \(\langle G \rangle\) depicted in (2) coincides then with the infinitary Böhm tree \(N = BT(M)\) of same higher-order alphabet \(\Sigma\) and same type \(\circ\) generated by the simply-typed \(\lambda Y\)-term \(M\).

D. Infinitary \(\lambda\)-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 \(\lambda Y\)-term \(M\) as an infinitary simply-typed \(\lambda\)-term, obtained by applying the unfolding rule

\[
YP \simeq P (YP)
\]

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 \(\lambda\)-terms. Moreover, the two infinitary \(\lambda\)-terms \(M\) and \(N\) are related by an infinite and strongly convergent sequence of \(\beta\)-rewriting steps in the sense of [11], [4], what we write:

\[
f : M \rightarrow_\beta N
\]

In particular, the fact that \(M\) and \(N\) have the same higher-order alphabet \(\Sigma\) and the same type \(\circ\) 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 \(\lambda\)-terms reveals an intriguing limitation of higher-order model-checking: the fact that the \(\lambda\)-terms \(M\) and \(N\) have higher-order alphabet \(\Sigma\) 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 \(A\) 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 \(\Sigma\) 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 \(A\) would be to explore an infinitary simply-typed \(\lambda\)-term \(M\).
of arbitrary higher-order alphabet $\Sigma$ and of arbitrary type $A$, and to tell us whether the $\lambda$-term $M$ is accepted or rejected according to the inductive and coinductive conditions of the parities. Moreover, the automaton $A$ should behave in the same way as an alternating parity tree automaton when all the letters of the alphabet $\Sigma$ 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 $A = (Q, \Sigma, A, \delta, q_0)$ where

- $Q$ is a finite set of ground states,
- $\Sigma$ is the higher-order alphabet of the automaton,
- $A$ is the simple type of the automaton,
- $\delta$ is a transition function of the alphabet $\Sigma$,
- $q_0$ 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:

$$Q_A = Q \implies Q_{A \Rightarrow B} = \mathcal{P}_{\text{fin}}(Q_A) \times Q_B$$

where $\mathcal{P}_{\text{fin}}(Q_A)$ 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

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

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

$$A \Rightarrow B = (!A) \multimap B.$$  

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

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

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

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

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

of higher-order states of type $A_i = \Sigma(a_i)$. 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

$$A = (Q, \Sigma, A \Rightarrow B, \delta, q_0)$$

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

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

Now that the value of the transition function $\delta$ has been defined for the letter $a$, the automaton $A$ may carry on and explore with initial state $q \in Q_B$ the infinitary $\lambda$-term $\Sigma$ and each element $q_i \in Q_A$ of type $A$. In that situation, the higher-order automaton $A$ behaves in the following way: it starts by extending its alphabet $\Sigma$ with the letter $a$ of arity $A$, and its transition function $\delta$ with the assignment

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

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

### F. Higher-order parity automata

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

$$A = (Q, \Sigma, A, \delta, q_0)$$

in just the same way as a higher-order automaton, except that the set $Q_A$ of higher-order states is replaced in all
definitions by the set $Q_A$ of colored higher-order states, defined by induction on the simple type $A$:

$$Q_\circ = Q \quad Q_A \mapsto B = \mathcal{P}_\text{fin}(\Omega \times Q_A) \times Q_B$$

A colored state $q \in Q_A \mapsto B$ is thus of the form

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

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

$$\delta(a_i) \in \mathcal{P}_\text{fin}(\Omega \times Q_A)$$

of pairs $(c, q)$ consisting of a color $c \in \Omega$ and of a colored state $q \in Q_A$, of type $A_i = \Sigma(a_i)$. By way of illustration, the parity automaton $A$ 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 $A$ has its edges labelled by colors $c_i \in \Omega$ and one requires that it satisfies the familiar conditions that in every infinite branch $p$ of the run-tree $\mathcal{R}$, the maximum color $c \in \Omega$ appearing infinitely often on the edges of $p$ is even (that is, coinductive).

G. The decidability theorem

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

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

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

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

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

\textit{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.

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

The property relies on the notion of strongly convergent $\beta$-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 $\lambda$-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 $\lambda Y$-term $M$ and a higher-order parity automaton $A$, the first theorem tells us that the acceptance of the infinite unfolding $[M]_\infty$ of $M$. Then, there exists an infinitary strongly convergent $\beta$-rewriting path $[M]_\infty \Rightarrow_\beta N$. 

![Fig. 1. Illustration of a run-tree of a higher-order automaton $A$ with an empty alphabet, with type $(\sigma \Rightarrow \sigma \Rightarrow \sigma) \Rightarrow (\sigma \Rightarrow \sigma) \Rightarrow \sigma$, and with initial state $\{(q_1) \Rightarrow (q_0) \Rightarrow (q_0) \Rightarrow (q_1) \Rightarrow (q_0) \Rightarrow q_0\}$.](image_url)
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 \( A \) is equivalent to the acceptance of \([M]_\infty\) and is thus
decidable.

**Plan of the paper**

After describing in [11] the infinitary simply-typed \( \lambda \)-
calculus, we develop in [11] the notion of higher-order automaton. We then
prove the “forward” and “backward” part of the invariance theorem in [14] and in [15]. The
backward part is the more difficult to prove, and we develop in [15] a residual theory of
diffraction patterns in order to establish it rigorously. We then lift in [17] the
invariance theorem from bare \( \lambda \)-terms to \( \lambda \)-terms with boundary. We establish our main theorem (Thm. 1)
in [17] by a translation of the original model-checking problem in \( \lambda \mathrm{Y} \mu \)-calculus, following the ideas of [7].

II. INFINTARY SIMPLY-TYPED \( \lambda \)-TERMS

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

**A. Untyped infinitary \( \lambda \)-terms**

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

\[
M, N ::\ = \ \lambda a. M \mid \text{App}(M, N) \mid a \in \operatorname{Var}
\]

where the constructor \( \lambda a. M \) binds the variable \( a \) in the
\( \lambda^\infty \)-term \( M \), in the nominal sense. One consequence
of this nominal and coinductive definition is that the set \( \text{fv}(M) \subseteq \operatorname{Var} \) of free variables of an untyped \( \lambda^\infty \)-term \( M \) is always finite, see [13] for a discussion. On the other hand, an untyped \( \lambda^\infty \)-term \( M \) in that sense
may contain an infinite countable number of bounded variables, which is precisely what happens when one
unfolds a \( \lambda \mathrm{Y} \)-term \( M \) into an infinitary \( \lambda \)-term \( [M]_\infty \),
see an illustration in the Appendix, [13]. Moreover, a free variable \( a \in \text{fv}(M) \) may appear at a countable (finite or infinite) number of occurrences of the \( \lambda^\infty \)-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 \( \text{App}(M, N) \) for the
application, instead of the more familiar notation \( MN \) used in the introduction.

**B. Simply-typed infinitary \( \lambda \)-terms**

A typing judgment

\[
\Sigma \vdash M : A
\]

is defined as a triple consisting of a higher-order alphabet \( \Sigma \), of an untyped \( \lambda^\infty \)-term \( M \) and of a simple type \( A \), such that all the free variables of \( M \) are letters in the
alphabet \( \Sigma \). A typing derivation \( \Upsilon \) is a syntactic tree
defined coinductively using the three typing rules below:

- **Variable**
  \[
  \Sigma, a : A \vdash a : A
  \]

- **Abstraction**
  \[
  \Sigma, a : A \vdash M : B \Rightarrow \Sigma \vdash \lambda a. M : A \Rightarrow B
  \]

- **Application**
  \[
  \Sigma \vdash M : A \Rightarrow B \quad \Sigma \vdash N : A \\
  \Sigma \vdash \text{App}(M, N) : B
  \]

The conclusion of a typing derivation \( \Upsilon \) is defined as the
typing judgment \( \Sigma \vdash M : A \) labelling the root of \( \Upsilon \).
A simply-typed \( \lambda^\infty \)-term of type \( A \) in the higher-order alphabet \( \Sigma \) is defined as a pair \((M, \Upsilon_M)\) consisting of an
untyped \( \lambda^\infty \)-term \( M \) and of a typing derivation \( \Upsilon_M \) with

**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 \( f : M \Rightarrow^* N \) which transform an
infinitary \( \lambda \)-term \( M \), typically the unfolding a
simply-typed \( \lambda \mathrm{Y} \)-term, into another infinitary \( \lambda \)-term \( N \).
A typical example is provided by the rewriting process
which transforms a higher-order recursion scheme \( M \)
seen as an infinitary \( \lambda \)-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

\[
o := \varepsilon \mid \text{body} \cdot o \mid \text{fun} \cdot o \mid \text{arg} \cdot o.
\]

The set \( \text{occ}(M) \) of occurrences of a simply-typed and
infinitary \( \lambda \)-term \( M \) is defined by coinduction

\[
\begin{align*}
\text{occ}(a) &= \{ \varepsilon \} \\
\text{occ}(\lambda a. M) &= \{ \varepsilon \} \uplus \{ \text{body} \cdot o \mid o \in \text{occ}(M) \} \\
\text{occ}(\text{App}(M, N)) &= \{ \varepsilon \} \uplus \{ \text{fun} \cdot o \mid o \in \text{occ}(M) \} \\
&\quad \uplus \{ \text{arg} \cdot o \mid o \in \text{occ}(N) \}
\end{align*}
\]

The purpose of an occurrence is to designate a specific position in an infinitary \( \lambda \)-term \( M \). Given two occurrences \( o, o' \in \text{occ}(M) \), 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 \( o \in \text{occ}(M) \) induces a
case \( C_{(M,o)}[-] \) defined as the infinitary \( \lambda \)-term with
a unique hole at the occurrence \( o \); and an infinitary \( \lambda \)-term \( M_\omega \) defined as the subterm at occurrence \( o \) of the infinitary \( \lambda \)-term \( M \). We write type\(( M_\omega, o)\) for the type of the subterm \( M_\omega \) of \( M \). One recovers the infinitary \( \lambda \)-term \( M \) by plugging the infinitary \( \lambda \)-term \( M_\omega \) inside the unique hole of the context \( \mathcal{C}_{(M_\omega,o)}[\cdot] \) in the following way:

\[
M = \mathcal{C}_{(M_\omega,o)}[M_\omega]
\]

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

\[
M_\omega = \text{App}(\lambda a. P, Q)
\]

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

**Proposition 1 (Subject Reduction):** Suppose that a simply-typed \( \lambda^\omega \)-term \( M \) is defined by the typing derivation \( \Upsilon_M \) of the typing judgment \( \Sigma \vdash M : A \) and that \(( M, o, N)\) is a \( \beta \)-redex. Then, there exists a canonical typing derivation \( \Upsilon_N \) of the typing judgment \( \Sigma \vdash N : A \) which turn \( N \) into a simply-typed infinitary \( \lambda \)-term, with the same alphabet \( \Sigma \) and the same type \( A \) as the original \( \lambda \)-term \( M \).

From this, one defines a graph \( G(\Sigma, A) \) whose vertices are the infinitary simply-typed \( \lambda \)-terms \( M, N \) of alphabet \( \Sigma \) and of type \( A \), and whose edges \( M \rightarrow N \) are the \( \beta \)-rewriting 

\( (M, o, N) \) is the \( \beta \)-rewriting path. Then, there exists a canonical typing derivation \( \Upsilon_N \) of the typing judgment \( \Sigma \vdash N : A \) which turn \( N \) into a simply-typed infinitary \( \lambda \)-term, with the same alphabet \( \Sigma \) and the same type \( A \) as the original \( \lambda \)-term \( M \).

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

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

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 \( \lambda^\omega \)-term \( M \) is defined by the typing derivation \( \Upsilon_M \) of the typing judgment \( \Sigma \vdash M : A \) and that \( f : M \rightarrow N \) is a strongly convergent \( \beta \)-rewriting path. Then, there exists a canonical typing derivation \( \Upsilon_N \) of the typing judgment \( \Sigma \vdash N : A \) which turn \( N \) into a simply-typed infinitary \( \lambda \)-term, with the same alphabet \( \Sigma \) and the same type \( A \) as the original \( \lambda \)-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 \((Q_A, \sqsubseteq_{\downarrow A})\) of higher-order states of type \( A \), defined by structural induction on the type \( A \). The preordered set \((Q_\downarrow, \sqsubseteq_\downarrow)\) of ground states is defined as the set \( Q \) equipped with the preorder \( \sqsubseteq_Q \):

\[
\forall q, q' \in Q_\downarrow, \quad q \sqsubseteq_Q q' \iff q \sqsubseteq_Q q'.
\]

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 \((Q_{\downarrow A}, \sqsubseteq_{\downarrow A})\) is defined as the set of finite subsets of higher-order states of \( A \)

\[
Q_{\downarrow A} = \forall_{\overline{n}}(Q_A)
\]

equipped with the following preorder:

\[
\{a_1, \ldots, a_m\} \sqsubseteq_{\downarrow A} \{b_1, \ldots, b_n\}
\]

if and only if there exists a functor \( f : [m] \rightarrow [n] \) such that \( a_i \sqsubseteq_{\downarrow A} b_{f(i)} \) 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:

\[
Q_{A\Rightarrow B} = Q_{\downarrow A} \times Q_B \quad \sqsubseteq_{A\Rightarrow B} = \sqsubseteq_{\downarrow A} \times \sqsubseteq_B.
\]

where \( \sqsubseteq_{\downarrow A} \) denotes the preorder obtained by reversing the orientation of the preorder \( \sqsubseteq_{\downarrow A} \).
B. Higher-order automata

A higher-order automaton \( A \) with type \( B \) and with higher-order alphabet

\[
\Sigma : \Sigma \rightarrow Type
\]

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

\[
\delta(a) \in \mathcal{P}_{fin}(Q_A)
\]

of higher-order states of type \( A = \Sigma(a) \) in the hierarchy of higher-order states generated by the set of ground states \( Q_o = Q \); 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 \( \delta \) may be equivalently seen as a vector of finite sets of higher-order states:

\[
\delta \in \prod_{a \in \Sigma} \mathcal{P}_{fin}(Q_{\Sigma(a)})
\]

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

**Definition 1 (Preorder on automata):** Given two higher-order automata \( A = (Q, \Sigma, A, \delta, q) \) and \( A' = (Q, \Sigma, A, \delta', q') \) with same alphabet \( \Sigma \) and same type \( A \), one writes \( A' \sqsubseteq \Sigma A \ A \) precisely when \( \delta \sqsubseteq \Sigma \delta' \) and \( q' \sqsubseteq A \ q \).

C. Run-trees

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

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

\[
(\Sigma \vdash M : A | \delta, q)
\]

in the deduction system generated by the rules below:

- **Var** \( \vdash q \sqsubseteq A q' \quad q' \in \delta(a) \)

- **Abs**

  \[
  \langle \Sigma, a : A \vdash a : A | \delta, q \rangle \\
  \langle \Sigma, \lambda a. M : A \Rightarrow B | \delta, \{q_1, \ldots, q_n\} \Rightarrow q \rangle
  \]

- **App**

  \[
  \langle \Sigma \vdash M : A \Rightarrow B | \delta, u \Rightarrow q \rangle \\
  \langle \Sigma \vdash A(\text{App}(M, N)) : B | \delta, q \rangle
  \]

- **Bag**

  \[
  \langle \Sigma \vdash M : A | \delta, q_1 \rangle \ldots \langle \Sigma \vdash M : A | \delta, q_n \rangle
  \]

  \[
  \langle \Sigma \vdash M : A | \delta, \{q_1, \ldots, q_n\} \rangle
  \]

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

\[
\langle \Sigma \vdash M : A | \delta, \{q_1, \ldots, q_n\} \rangle
\]

to denote the set of \( n \)-tuples of run-trees:

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

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 \( \text{Bag} \) is rule is replaced by the \( \text{\inf-Bag} \) rule:

\[
\forall i \in I. \langle \Sigma \vdash M : A | \delta, q_i \rangle
\]

D. Acceptance

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

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

The next section shows that acceptance is closed under the preorder relation between higher-order automata formulated at the end of §III-B (Def. 4).

**Proposition 3:** Suppose given two higher-order automata \( A \) and \( A' \) with same alphabet \( \Sigma \) and same type \( A \), such that \( A' = (Q, \Sigma, A, \delta', q') \sqsubseteq \Sigma A A \). Then \( A \) is accepted by \( A' \).

**Proof.** Technically speaking, an easy proof by coinduction on the four rules \( \text{Abs}, \text{App}, \text{Var} \) and \( \text{Bag} \) of the deduction system enables one to replace every run-tree \( R \) in \( \langle \Sigma \vdash M : A | \delta, q \rangle \) by a run-tree \( R' \) in \( \langle \Sigma \vdash M : A | \delta', q' \rangle \) defined by increasing the transition function and by decreasing the initial state at each node of the run-tree \( R \). 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.
Theorem 4 (Forward Preservation): Suppose that two simply-typed infinitary λ-terms \( M \) and \( N \) with alphabet \( \Sigma \) and type \( A \) are related by a strongly convergent \( \beta \)-rewriting path \( f : M \rightarrow B \). In that case, every higher-order automaton \( A \) with alphabet \( \Sigma \) and type \( A \) which accepts \( M \) also accepts \( N \).

Proof. The proof is based on an easy adaptation to the deduction system in \([\text{III}C]\) of the Infinitary Subject Reduction theorem (Prop. 2) established for the original deduction system of the simply-typed \( \lambda \)-calculus in \([\text{II}B]\). 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 \( \lambda \)-terms in variable occurrences of infinitary simply-typed \( \lambda \)-terms. In this way, it is possible to transform the original run-tree \( R \) of the automaton \( A \) on \( M \) into a run-tree \( R_{p+1} \) of the same automaton \( A \) on the infinitary \( \lambda \)-term \( M_{p+1} \) obtained after computing the prefix

\[
f_{0,p} = M_0 \xrightarrow{R_0} M_1 \xrightarrow{R_1} \cdots \xrightarrow{R_{p-1}} M_p
\]

of length \( p \) of the infinitary \( \beta \)-rewriting path \( f \). The fact that \( f \) is a strongly convergent infinitary \( \beta \)-rewriting path ensures that the family of run-trees \( \{R_p\}_{p \in \mathbb{N}} \) converges towards a run-tree \( R_\infty \) of the simply-typed infinitary \( \lambda \)-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 \( \lambda_\infty \)-term. We then develop a residual theory for diffraction patterns which restricts the residual theory for occurrences. This “diffraction theory” will be the main tool to prove in \([\text{VII}]\) the most difficult direction: backward preservation, of our invariance theorem (Thm. 3). We start by defining the notion in \([\text{VA}] \) and \([\text{VB}] \) and then illustrate it in \([\text{VC}] \); although the notion of diffraction pattern is extremely natural, it is also new, and we thus advise the reader to read \([\text{VA}] \), \([\text{VB}] \) and \([\text{VC}] \) simultaneously, and to look at the diffraction of an occurrence of \( M \) along a strongly convergent rewriting path \( f : M \rightarrow B \) illustrated in the Appendix, \([\text{DF}] \).

A. Diffraction patterns

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

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

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 \( \text{hdocc}(D) = o \).
- a pair \( D = \{D_{A,i} \mid i \in I\} \) also noted

\[
D = \{D_{A,i} \mid i \in I\} \rightarrow D_B
\]

consisting of a family \( \{D_{A,i} \mid i \in I\} \) 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:

\[
\forall i \in I, \quad \text{hdocc}(D_B) \preceq_M \text{hdocc}(D_{A,i})
\]

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

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

\[
\text{occ}(M) \subseteq \text{diff}(M)
\]

where \( \text{diff}(M) \) denotes the set of diffraction patterns of type \( A \) in \( M \), and denotes the set of diffraction patterns of arbitrary type \( A \in \text{Type} \) 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 \( D \in \text{diff}(M) \) comes together with a rooted tree \( \text{shape}(D) \) called the shape of \( D \). The rooted tree \( \text{shape}(D) \) is defined by induction on the diffraction pattern \( D \):

- when the diffraction pattern \( D \) of ground type \( o \) or of arrow type \( A \Rightarrow B \) is equal to an occurrence \( o \in \text{occ}(M) \) of the same type, the rooted tree \( \text{shape}(D) \) 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 \( D = \{D_{A,i} \mid i \in I\} \rightarrow D_B \), the rooted tree \( \text{shape}(D) \) is defined as the rooted tree \( \text{shape}(D_B) \) with root \( *_B \), extended with the rooted trees \( \text{shape}(D_{A,i}) \) with their roots \( *_{A,i} \) connected as children to the root \( *_B \) of \( \text{shape}(D_B) \).

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

\[
\text{embed}_D : (\text{shape}(D), \preceq_E) \rightarrow (\text{occ}(M), \preceq_M)
\]

which maps every node \( *_E \in \text{shape}(D) \) to the head occurrence \( \text{hdocc}(E) \in \text{occ}(M) \) of the underlying diffraction pattern \( E \). Note that \( \text{embed}_D \) defines a monotone function.
embed\textsubscript{\textit{D}} from the set of nodes of shape(D) equipped with 
its tree nesting order \preceq\textsubscript{\textit{D}}, to the set of occurrences \textit{occ}(\textit{M}) of the simply-typed \textit{\lambda}\textsuperscript{\textit{\infty}}-term \textit{M}, equipped with 
the nesting order \preceq\textsubscript{\textit{M}}.

C. Residuals of diffraction patterns

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

\[
M = C_{(M,o)}[(\lambda a.P)Q] \xrightarrow{R} C_{(M,o)}[P[a := Q]] = N
\]

where \( C_{(M,o)}[-] \) is the context with unique hole at occurrence \( o \) defined in \S\textsuperscript{II-C} and where the \( \lambda\)-term \( P \) has type \( A \) and the \( \lambda\)-term \( Q \) has type \( B \).

We are interested now in what happens to the occurrence \( o_R \) of the \( \beta\)-redex \( R \) of type \( A \rightarrow B \) after \( \beta\)-reduction of \( R \). In the traditional theory of residuals, the occurrence \( o_R \) of type \( A \rightarrow B \) has no residual along the \( \beta\)-reduction of \( R \). The intuition is that the occurrence \( o \) has been “consumed” during the process of \( \beta\)-reduction, and thus disappears. The intuition underlying our diffraction theory is quite different: our idea is that the occurrence \( o \) has been “diffracted” by the \( \beta\)-redex \( R \) into a diffraction pattern \( E = \{D_{A,i} | i \in I\} \rightarrow D_B \) defined with the following components: the occurrence \( D_B = o_B \) 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:

\[
M \xrightarrow{R} \beta N
\]

where we depict the occurrence \( D = o_R \) as a white circle in the \( \lambda\textsuperscript{\textit{\infty}}\)-term \( M \), and similarly for the occurrence \( D_B = o_B \) and each occurrence \( D_{A,i} = o_{A,i} \) in the \( \lambda\)-term \( N \), for \( i \in I \). Note that the shape of the diffraction pattern \( E \) in \( N \) residual of \( D \) along the \( \beta\)-redex \( R \) looks as follows:

\[
\text{shape}(E) = \ast_B \ast_{A,1} \ast_{A,2} \ast_{A,3}
\]

and that the monotone function embed\textsubscript{\textit{D}} transmits every node of shape\textsubscript{\textit{E}} to the following occurrences: \( \ast_B \rightarrow o_B \) and \( \ast_{A,i} \rightarrow o_{A,i} \). Note that the function embed\textsubscript{\textit{D}} is indeed monotone, since \( o_B \preceq_N o_{A,i} \) for all \( i \in I \).

Given a \( \beta\)-redex \( R = (M,o,N) \) as above, the definition of a residual \( E \in \text{diff}(N) \) of a diffraction pattern \( D \in \text{diff}(M) \) along a \( \beta\)-redex \( M \rightarrow \beta N \) 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, \S\textsuperscript{D} We write 

\[
D[R]E \text{ when } E \in \text{diff}(N) \text{ is a residual of } D \in \text{diff}(M) \text{ along a } \beta\text{-redex } R : M \rightarrow \beta N.
\]

By extension, we write \( D[f]E \) when \( E \in \text{diff}(N) \) is a residual of \( D \in \text{diff}(M) \) along a finite \( \beta\)-rewriting path from \( M \) to \( N \), in the expected sense, see the Appendix, \S\textsuperscript{D}. An important observation is that

\[
\text{Proposition 4: Every strongly convergent infinitary } \beta\text{-rewriting path } M \rightarrow \beta N \text{ induces a residual relation }
\]

\[
[f] \subseteq \text{diff}(M) \times \text{diff}(N).
\]

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:

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

\textbf{Proof.} In order to establish the theorem, we need to deduce the existence of a run-tree \( R_M \) of the automaton \( A = (Q, \Sigma, A, \delta, q) \) on the \( \lambda\textsuperscript{\textit{\infty}}\)-term \( M \) from the existence of a run-tree \( R_N \) of \( A \) on the \( \lambda\textsuperscript{\textit{\infty}}\)-term \( N \), using the fact that \( M \) and \( N \) are related by a strongly convergent \( \beta\)-rewriting path \( f : M \rightarrow \beta N \). The diffraction theory formulated in \S\textsuperscript{IV} plays a key role in that reconstruction. First of all, it follows from Prop. \textsuperscript{4} that every occurrence \( o \in \text{occ}(M) \) has a countable set \( o[f] \subseteq \text{diff}(N) \) 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 \( E[\text{diff}(\Sigma)] \) of diffraction patterns \( F \) in the run-tree \( R_N \). The key observation is that every such diffraction pattern \( F \in \text{diff}(\Sigma) \) comes equipped with a higher-order state \( q_F \in Q_A \), where \( A = \text{type}(M, o) \) 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 \( \text{shape}(F) \) embedded in the run-tree \( R_N \). If we write \( [f : R_N] \) for the composite relation \( [f] \circ [R_N] \), we associate in this way to every occurrence \( o \in \text{occ}(M) \) of type \( A = \text{type}(M, o) \) in \( M \) a countable set \( D_o = o[f : R_N] \) of diffraction patterns \( F \) in \( R_N \), each of them labelled by a higher-order state \( q_F \in Q_A \). A careful inspection establishes that the elements of the \( D_o \)'s combine together to define an infinite-branching run-tree \( R_M \) of the automaton \( A \) on the \( \lambda\textsuperscript{\textit{\infty}}\)-term \( M \), in the sense of \S\textsuperscript{IV}. The construction of \( R_M \) from \( D \) is a variant of the Grothendieck construction. The equivalence in \S\textsuperscript{IV} 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 (Thms. 4 and 5). This invariance theorem enables us to establish a primary decidability theorem (Thm. 3) for \( \lambda Y \)-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

\[
N ::= \lambda a_1 \ldots \lambda a_m. a N_1 \ldots N_n | \bot \Sigma_{u:A}
\]

where \( \bot \Sigma_{u:A} \) is a constant of the calculus indexed by a higher-order alphabet \( \Sigma \), a type \( A \), and a finite subset \( u \) of higher-order states of type \( \Sigma \Rightarrow A \).

Using the infinitary Church-Rosser property in [11], one establishes that given a finite set \( Q \) of ground states, every simply-typed \( \lambda^\infty \)-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 \( \bot \Sigma_{u:A} \); an \( \lambda^\infty \)-term \( M \) is called productive when its Böhm tree \( BT(M) \) does not contain any constant of the form \( \bot \Sigma_{u:A} \).

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

VII. LAMBDA-TERMS WITH BOUNDARY

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

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

**Definition 8 (Boundary):** A boundary \( \mathcal{B} \) of a simply-typed infinitary \( \lambda \)-term \( M \) is defined as a set \( \mathcal{B} \subseteq \infty \)-path(\( M \)) of infinite paths of \( M \). A simply-typed infinitary \( \lambda \)-term with boundary is a triple

\[
(M, \mathcal{B}(M), \mathcal{B}(M))
\]

consisting of a simply-typed infinitary \( \lambda \)-term \( (M, \mathcal{B}(M)) \) together with a boundary \( \mathcal{B}(M) \) of the infinitary \( \lambda \)-term \( M \).

The intuition is that an infinite path \( p \) should be considered as part of the \( \lambda \)-term \( M \) precisely when \( p \) is an element of the boundary \( \mathcal{B}(M) \), and not otherwise. The notion of boundary is inspired by the definition of a topological game whose winning condition is described by a subset \( \Omega \) (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 \( \mathcal{B} \) on the Church encoding BinTree of the infinite binary tree of signature \( \Sigma = \{ a : 2 \} \).

The purpose of \( \lambda \)-terms with boundary is to provide an interpretation of terms of the \( \lambda Y\mu \)-calculus, defined as the simply-typed \( \lambda \)-calculus extended with

- an inductive fixpoint operator \( Y_\mu : (A \Rightarrow A) \Rightarrow A \)
- a coinductive fixpoint operator \( Y_\nu : (A \Rightarrow A) \Rightarrow A \)

**Proposition 5:** Every simply-typed \( \lambda Y\mu \)-term \( M \) defines an infinitary simply-typed \( \lambda \)-term with boundary

\[
[M]_{\infty} = ([M]_\infty, \Psi([M]_\infty, P([M]_\infty))
\]

by structural induction on the \( \lambda Y\mu \)-term \( M \).

The construction is easy to describe. The infinitary simply-typed \( \lambda \)-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 \( \Psi([M]_\infty) \) is defined similarly. The boundary \( P([M]_\infty) \) is also defined by structural induction on \( M \). The two important cases by induction are the definition of the boundary of a \( \lambda Y\mu \)-term of the form \( Y_\mu M \) or \( Y_\nu M \). The \( \lambda Y\mu \)-terms \( Y_\mu M \) and \( Y_\nu M \) are unfolded as the same infinitary \( \lambda \)-term \( [Y_\mu M]_\infty = [Y_\nu M]_\infty \), depicted as

![Diagram](13)
of linear logic where of the translation of a simply-typed $\lambda$-term $\lambda Y$ by the question whether the infinitary simply-typed $\omega_1$ decidability theorem (Thm. 7) just established in §VII for automaton boundary enables us to refine Thm. 6 and to establish a simply-typed $\lambda Y$-term $M$ with boundary $P_M$ induces a boundary $P_N$ on the simply-typed $\lambda Y$-term $N$.

The boundary $P_N$ is defined as the set of all the infinite paths $q \in \infty$-path($N$) such that every infinite path $p \in \infty$-path($M$) necessary to $q$ along $f : M \rightarrow \beta N$ 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 $\lambda Y$-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 $\lambda Y$-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 $A$ and a simply-typed $\lambda Y_{\mu \nu}$-term $M$ of same higher-order alphabet $\Sigma$ and of same type $A$. Then, the question whether the infinitary simply-typed $\lambda Y$-term with boundary $N = BT(M)$ generated by $M$ is accepted by $A$ is decidable.

### VIII. HIGHER-ORDER PARITY AUTOMATA

In this section, we deduce the decidability theorem for $\lambda Y$-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 $\lambda Y$-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:

$$\square A = (A \times \ldots \times A)$$ (14)

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

$$[1] = 1 \quad [A \times B] = [A] \times [B]$$

and the translation $[-]$ on $\lambda$-terms applies the recipe of the translation of a simply-typed $\lambda$-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 $\&$, 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

$$\delta_A : \square A \rightarrow \square \square A \quad \varepsilon_A : \square A \rightarrow A$$
defined as the simply-typed $\lambda$-terms $\delta_A$ and $\varepsilon_A$ below:

$$(a_1, \ldots, a_k) : \square A \vdash (\{(a_{\max(h,i)}|1 \leq i \leq k\})_{1 \leq j \leq k} : \square A$$

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

$$F : A , \Sigma \vdash P : A$$
defines a simply-typed $\lambda$-term $\lbrack P \rbrack$ of alphabet and type

$$F_1 : A, F_2 : A, \ldots, F_k : A, \square \lbrack \Sigma \rbrack \vdash \lbrack P \rbrack : \lbrack A \rbrack$$

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:

$$N = Y_{\mu} Y_{\mu} Y_{\nu} \ldots Y_{\nu} (\lbrack P \rbrack)$$

$k$ fixpoints

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

**Proposition 7:** A higher-order parity automaton $A$ with higher-order alphabet $\Sigma$ and type $A$ is the same as a higher-order automaton $\lbrack A \rbrack$ with higher-order alphabet $\lbrack \Sigma \rbrack$ and type $\lbrack A \rbrack$.

Moreover, the acceptance of the simply-typed $\lambda Y$-term $M$ by a higher-order parity automaton $A$ is equivalent to the acceptance of the $\lambda Y_{\mu \nu}$-term $N$ defined above by the higher-order automaton $\lbrack A \rbrack$. The decidability theorem (Thm. 7) 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 \( \lambda \)-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 Mellies [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], [14], most notably strong convergence, diffraction patterns and infinitary \( \lambda \)-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 [23] 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 \( \lambda \)-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, Étienne Lozes, Sylvain Salvati, Olivier Serre and Igor Walukiewicz for discussions and feedback on this work.

REFERENCES

**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 $\Sigma$ consisting of a unique letter $\con$ of higher-order arity

$$\con : \omega \Rightarrow \varnothing \Rightarrow \varnothing$$

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

$$[n_0, n_1, n_2, \ldots] = \con(n_0, \con(n_1, \con(n_2, \ldots)))$$

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

$$\con : \omega \Rightarrow \varnothing \Rightarrow \varnothing \vdash M : \con$$

the question whether $M$ generates an infinite stream $N = BT(M)$ of completed natural numbers accepted by the automaton $A$ 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 $\Sigma$ and transition function $\delta$ of the automaton may be altered (and in fact extended) in the course of the exploration of the simply-typed $\lambda^\infty$-term. This typically happens when a higher-order automaton with alphabet $\Sigma$, transition function $\delta$ and initial state $q_0 = \{q_1, \ldots, q_n\} \leadsto q$ reaches a simply-typed $\lambda^\infty$-term of the form

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

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

**Abstraction**

$$\Sigma, a : A \vdash M : B$$

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

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 $\delta$ with

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

This ability to extend the alphabet in the course of exploration of the simply-typed $\lambda^\infty$-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 $A = (Q, \Sigma, A, \delta, q_0)$ against a simply-typed $\lambda^\infty$-term $\Sigma \vdash M : A$ of context $\Sigma$ and of type $A$. By definition, the set of run-trees is only defined when:

- the context $\Sigma$ of the simply-typed $\lambda^\infty$-tree $M$ coincides with the alphabet $\Sigma$ of the automaton $A$,
- the type $A$ of the simply-typed $\lambda^\infty$-tree $M$ coincides with the type $A$ of the automaton $A$.

For that reason, we generally find convenient to write

$$(\Sigma \vdash M : A | \delta, q)$$

for the set of run-trees of the higher-order automaton $A = (Q, \Sigma, \delta, q_0)$ of same type $A \Rightarrow B$. By definition, the initial state $q_0$ of the automaton $A$ is a higher-order state of type $A \Rightarrow B$, and it is thus of the form

$$q_0 = \{q_1, \ldots, q_n\} \leadsto q$$

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

$$(\Sigma \vdash \lambda a. M : A \Rightarrow B | \delta, \{q_1, \ldots, q_n\} \leadsto q) \quad (15)$$

is defined coinductively as

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

The intuition behind this definition is that a run-tree exploring $\lambda a. M$ with initial state $\{q_1, \ldots, q_n\} \leadsto q$ starts by declaring the letter $a$ with transition function $\delta(a) = \{q_1, \ldots, q_n\}$, and then explores the simply-typed $\lambda^\infty$-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 $\text{(Lam, q)} \cdot Z$ stands for the set $\{(\text{Lam}, q) \times Z | \text{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 $\lambda^\infty$-term

$$\Sigma \vdash \text{App}(M, N) : B$$

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

**Application**

$$\Sigma \vdash M : A \Rightarrow B \quad \Sigma \vdash N : A$$

$$\Sigma \vdash \text{App}(M, N) : B$$

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

$$(\Sigma \vdash \text{App}(M, N) : B | \delta, q) \quad (16)$$
is defined coinductively as the set
\[ \prod_{u \in \mathcal{P}_{\nu a}(Q_A)} (\text{App}, u \rightarrow q) \cdot (\Sigma \vdash M | \delta, u \rightarrow q) \times \prod_{i=1}^{n} (\Sigma \vdash N | \delta, q_i) \]

The intuition behind this definition is that a run-tree exploring \text{App}(M, N) with initial state \(q\) is provided by a run-tree exploring the simply-typed \(\lambda\)-term \(M\) with initial state \(u \rightarrow q\) 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 \(\lambda\)-term \(N\), each run-tree starting with the initial state \(q_i \in Q_A\), for \(1 \leq i \leq n\). Note the state \(u \rightarrow q\) 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 \((\text{App}, q)\cdot Z\) stands for the set \{\(\{(\text{App}, q)\}\)\} \times Z where \text{App} is a tag, \(q\) is a state and \(Z\) is a set.

3) \textbf{Variable}: Suppose given a simply-typed \(\lambda\)-term \(\Sigma, a : A \vdash a : A\) with underlying typing derivation \(\Gamma_a\) is defined by a \textbf{Variable} rule. Suppose given an automaton \(A = (Q, \Sigma, \delta, q)\) of the same alphabet and of same type \(A\). In that case, the set of run-trees of the automaton is defined as
\[ \langle \Sigma, a : A \vdash a : A \mid \delta + a \rightarrow \{q_1, \ldots, q_n\}, q \rangle = \{\text{Var}, q\} \]
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
\[ \langle \Sigma, a : A \vdash a : A \mid \delta + a \rightarrow \{q_1, \ldots, q_n\}, q \rangle = \emptyset \]
otherwise, when \(q\) is not an element of \(\{q_1, \ldots, q_n\}\).

Here, the notation \{\text{Var}, q\} stands for the singleton set \{\{\text{Var}, q\}\} where \text{Var} is a tag and \(q\) is a state.

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

C. \textbf{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 \(\lambda\text{-terms}\) (Def. 2) with a subtyping relation \(\sqsubseteq_A q'\) in the \textbf{Var} rule
\[ \text{Var} \vdash q \sqsubseteq_A q' \rightarrow q' \in \delta(a) \]
\[ (\Sigma, a : A \vdash a : A \mid \delta, q) \]

By way of illustration, this subtyping relation \(\sqsubseteq_A q'\) appearing in the \textbf{Var} rule ensures that the automaton \(A\) with empty alphabet, with type \(A \Rightarrow B\) \Rightarrow A \Rightarrow B\)
and with initial state \(\{\{\} \rightarrow q_B\} \rightarrow \{q_A\} \rightarrow q_B\)
accepts the simply-typed \(\lambda\)-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 \(\eta\)-expansion, the simply-typed \(\lambda\)-term
\[ \vdash \lambda f. \lambda a. \text{App}(f, a) \]

The subtyping relation \(q \sqsubseteq_A q'\) appearing in the rule \(\text{Var}\) is thus necessary to have a typing system invariant under \(\eta\)-conversion, even when the preorder \(\sqsubseteq = \sqsubseteq_o\) on the set \(Q\) of ground states is defined as the identity relation:
\[ \forall q, q' \in Q, \quad q \sqsubseteq q' \iff q = q' \]

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. \textbf{Diffraction patterns in run-trees}

In our proof of the backward preservation theorem (Thm. 5 in [6]) we use the notion of diffraction pattern
\[ D \in \text{diff}(\mathcal{R}) \]
living in a run-tree \(\mathcal{R}\) of an automaton \(A\) on a simply-typed \(\lambda\)-term \(M\). The notion of diffraction pattern in a run-tree
\[ \mathcal{R} \in (\Sigma \vdash M : B | \delta, q) \]
(17)
is defined just the same way as the notion of diffraction pattern in a simply-typed \(\lambda\)-term
\[ \Sigma \vdash M : B \]
using a structural induction on the type \(A\) of the diffraction pattern \(D\). In particular, every diffraction pattern \(D \in \text{diff}(\mathcal{R})\) comes equipped with a monotone function
\[ \text{embed}_D : \langle \text{shape}(D), \preceq_D \rangle \rightarrow (\text{occ}(\mathcal{R}), \preceq) \]
which transports every node of the rooted tree \text{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
\[ [\mathcal{R}] \subseteq \text{diff}(M) \times \text{diff}(\mathcal{R}) \]
which relates a diffraction pattern \(D \in \text{diff}(M)\) to its residuals in \(\text{diff}(\mathcal{R})\), defined in the expected way. Note that every residual \(E \in \text{diff}(\mathcal{R})\) of the diffraction pattern \(D \in \text{diff}(M)\) has the same underlying rooted tree
\[ \text{shape}(E) = \text{shape}(D) \]

Moreover, the embedding of a node in \text{shape}(E) is an occurrence of \(\mathcal{R}\) above the embedding of the corresponding node in \text{shape}(D).
E. Residuals of diffraction patterns

Suppose given a β-redex \( R = (M, o, N) \) between simply-typed \( \lambda \)-terms \( M \) and \( N \). The residual relation between diffraction patterns

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

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

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

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

- an occurrence residual \( o_E \in \text{occ}(N) \) of the occurrence \( o_D \in \text{occ}(M) \) 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 \( E = \{ D_{A,i} \mid i \in I \} \rightarrow D_B \) where \( D_B = o_B \) is the unique occurrence of type \( B \) in \( N \) such that \( o = o_B \cdot \) 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 \( \lambda \)-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 \( D = \{ D_{A,i} \mid i \in I \} \rightarrow D_B \) along a β-redex \( R \) are defined as

- the copies \( E \) of the diffraction pattern \( D \) when the head occurrence \( \text{hdocc}(D) \) is duplicated by the β-redex \( R \),
- otherwise, the unique residual \( E = \{ E_{A,i} \mid j \in J \} \rightarrow E_B \) 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

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

associated to a finite \( \beta \)-rewriting path

\[
f = M \xrightarrow{R_0} M_1 \xrightarrow{R_1} \cdots \rightarrow M_{p-1} \xrightarrow{R_{p-1}} N
\]

is simply defined by composing the residual relations together \([f] = [R_0] \circ \cdots \circ [R_{p-1}]\).
For notational convenience, we identify from now on $[M]_\infty$ with the original $\lambda Y$-term $M$. We also indicate in the picture the occurrences $o_R$, $o_S$ and $o_T$ of the three $\beta$-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 $\langle G \rangle$ described in the introduction, and depicted in (2). The diffraction patterns $D$, $E$ and $F$ residual in $N = BT(M)$ of the occurrence $o_R \in \text{diff}(M)$ is depicted by attaching every node $x$ of the rooted tree shape($D$) to its occurrence $\text{embed}_D(x) \in \text{occ}(N)$ in the infinitary $\lambda$-term $N$, as follows:

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

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

In the same way, the diffraction pattern $E$ residual in $N = BT(M)$ of the occurrence $o_S$ of the $\beta$-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 $\beta$-redex $T$ in $M$ is represented as follows:
Note that the diffraction patterns $D, E$ and $F$ indicate very clearly which part each $\beta$-redex $R, S$ and $T$ took in the construction of the infinitary $\lambda$-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 $A$ 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 $\beta$-rewriting path $f : M \to^\beta N$ 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 $G$ below:

$$G = \begin{cases} S & \to F b c \\ \lambda F. \lambda z. F z & \to y (F z) \end{cases} \tag{19}$$

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

$$\langle G \rangle = \begin{array}{c} b \\ b \\ b \\ b \\ \vdots \end{array} \tag{20}$$

The higher-order recursion scheme (19) may be encoded as the following $\lambda Y$-term

$$M = \left( Y [\lambda F. \lambda z. b (F z)] \right) c \tag{21}$$

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

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

$$[M]_\infty = \begin{array}{c} E \\ E_1 \end{array}$$

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

$$N = \begin{array}{c} E \end{array}$$
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.