Automata theory approach to predicate intuitionistic logic

MACIEJ ZIELENKIEWICZ, Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland.

ALEKSY SCHUBERT, Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland.

Abstract

Predicate intuitionistic logic is a well-established fragment of dependent types. Proof construction in this logic, as the Curry–Howard isomorphism states, is the process of program synthesis. We present automata that can handle proof construction and program synthesis in full intuitionistic first-order logic. Given a formula, we can construct an automaton such that the formula is provable if and only if the automaton has an accepting run. As further research, this construction makes it possible to discuss formal languages of proofs or programs, the closure properties of the automata and their connections with the traditional logical connectives.

Keywords: First-order logic, automata theory, Curry–Howard isomorphism, proof search, program synthesis

1 Introduction

Dependent types is a successfully evolving branch of functional programming in which types serve as expressions that describe the input–output behaviour of λ-terms, i.e. the basic programming units of functional programs. Intuitionistic first-order logic is the primary form of dependent types and its studies shed light on this programming paradigm. In particular the studies on proof construction in natural deduction proof systems of intuitionistic first-order logic are, due to Curry–Howard isomorphism, studies on program synthesis in functional programming. We propose to extend the selection of study methods available in this stream of research by giving a general notion of automata that are strong enough to accept proofs in first-order logic.

Investigations in automata theory lead to abstraction of algorithmic processes of various kinds. This enables analysis of their strength both in terms of their expressibility (i.e. which problems can be solved with them) and in terms of resources they consume (e.g. time or space). They also make it possible to shed a different light on the original problem (e.g. the linguistic problem of languages generated by grammars can be reduced to the analysis of pushdown automata), which makes it possible to conduct analysis that was not possible before. In addition, automata became a particular compact data structure that can in itself, when defined formally, be subject to further investigation, as finite or pushdown automata are in automata theory.

Typically, design of automata requires one to select a finite control over the process of interest. This is not always immediate when λ-calculi come into play as λ-terms can contain bound variables from an infinite set. One possibility consists of restricting the programming language so that there is no need to introduce binders. This method was used in the work by Düdder et al. [5], which was powerful enough to synthesize λ-terms that were programs in a simple but expressive functional language.
Another approach would be to restrict the program search to programs in total discharge form. In programs of this form, it is needed to keep track of types of available library calls, but not of the call names themselves. This idea was explored by Takahashi et al. [15] who defined context-free grammars that can be used for proof search in propositional intuitionistic logic, which is, by Curry–Howard isomorphism, equivalent to program search in the simply typed \( \lambda \)-calculus. Actually, the grammars can be viewed as performing program search using tree automata by means of the known correspondence between grammars and tree automata. However, the restriction to total discharge form can be avoided by means of techniques developed by Schubert, Dekkers and Barendregt [12].

A different approach to abstract machinery behind program search process was proposed by Broda and Damas [4] who developed a formula-tree proof method. This technique provides a realization of the proof search procedure for a particular propositional formula as a data structure, which can be further subject to algorithmic manipulation.

In addition to these investigations for intuitionistic propositional logic there was a proposal of applying automata theoretic notions to proof search in first-order logic [8]. In his paper, Hetzl characterizes a class of proofs in intuitionistic first-order logic recognizable by tree automata with global equalities and disequalities [7]. The characterization makes it possible to recognize proofs that are not necessarily in normal form, but is also limited to certain class of tautologies as the emptiness problem for the automata is decidable unlike provability in first-order logic.

In this paper we propose an automata-theoretical abstraction of the proving process in full intuitionistic first-order logic. Its advantages can be best expressed by stating which implicit but crucial features of the proof search process become explicit. In our automata the following elements of the proving process are exposed.

- The finite control of the proving process is made explicit.
- A binary internal structure of the control is explicated where one component corresponds to a subformula of the original formula and one to the internal operations that should be done to handle the proof part relevant for the subformula. As a by-product of this formulation, it becomes apparent how crucial the role of the subformula property is in the proving process.
- The resource that serves to represent eigenvariables that occur in the process is distinguished. This abstraction is important as the variables play a crucial role in complexity results concerning the logic [13, 14].
- The automata enable the possibility of getting rid of the particular syntactic form of formulas and instead work on more abstract structures.
- The definition of automaton distils the basic instructions necessary to conduct the proof process, which brings into the view more elementary operations the proving process depends on.

Although the work is formulated in terms of logic, it can be viewed as synthesis of programs in a restricted class of dependently typed functional programs.

A preliminary version of the paper was published in the Proceedings of LOPSTR’16 conference [17].

**Organization of the paper** We present the notation and intuitionistic first-order logic in Section 2. Next, we define our automata in Section 3 and use them as a target for a translation from first-order logic formulas in Section 4. In Section 5 we prove that the non-emptiness problem for the resulting automata is equivalent to provability in the logic. We summarize the account in Section 6.
2 Preliminaries

We present the notation and the basic facts about intuitionistic first-order logic. The notation \( A \rightarrow B \) is used to denote the type of partial functions from \( A \) to \( B \). We write \( \text{dom}(f) \) for the domain of the function \( f : A \rightarrow B \). For two partial functions \( f, g \) we define \( f \ll g = f \cup \{(x, y) \in g \mid x \notin \text{dom}(f)\} \). The set of all subsets of a set \( A \) is \( P(A) \).

We say that a partial order \( \leq \) is tree-like when for each \( x, y \) such that \( x \not\leq y \) and \( y \not\leq x \) there is no \( z \) such that \( x \leq z \) and \( y \leq z \). A tree is a tuple \( \langle A, \leq, \varepsilon, L, l \rangle \) where \( A \) is a finite set, called the carrier of the tree, \( \leq \) is a tree-like partial order on \( A \) that has the least element \( \varepsilon \), \( L \) is the set of labels and \( l : A \rightarrow L \) is the labelling function. Whenever the order, the set of labels and the labelling function are clear from the context, we abbreviate the quintuple to the symbol \( A \). At times it is convenient to define the order using the notion of successor. We say that a node \( v \) is a successor of the node \( w \), written \( w \succ v \), when \( w \neq v, w \leq v \) and there is no \( v' \) such that \( v' \neq w, v' \neq v \) and \( w \leq v' \leq v \). Once all the successors of a tree are determined, the order \( \leq \) is defined as the transitive-reflexive closure of the successor relation. We interpret formulas as trees in the standard way so that each internal node is labelled with a logical connective and the node has as many children as there are arguments of the connective. The leaves are labelled with variables. We identify formulas with trees obtained in this way. Since the formula notation makes it easy, we sometimes use a subtree \( \psi \) to actually denote a node in \( \psi \) at which \( \varphi \) starts. It should be clear from the context that this convention is used.

2.1 Intuitionistic first-order logic

The basis for our study is the intuitionistic first-order logic (for more details see e.g. the work of Urzyczyn [16]). We decided here to work with a system that does not have function symbols. The main reason is that logic of our choice has fewer basic constructs. Since one can introduce a theory in the logic with no function symbols in which certain predicates represent functions, the formalism of our choice constitutes a more fundamental system. However, it is worth noting here that formulas that encode the fundamental properties of function symbols give rise to bigger quantifier complexity of the resultant theory. They have to express in particular that for each argument the result of a function is uniquely defined and this necessarily involves quantifier alternation. On the other hand, representation of terms that use functional symbols in formulas with predicates only is proportional to the size of the term.

The system of first-order logic we work with here is in the natural deduction fashion as opposed to Hilbert-style calculi or sequent calculi. The main advantage of this format is for us the fact that it immediately corresponds to expressions in functional programming languages and provides basis not only for proof search, but also for program synthesis.

To define the syntax of intuitionistic first-order logic, we assume that we have a set of predicates \( \mathcal{P} \) that can be used to form atomic formulas and an infinite set \( X_1 \) of first-order variables, usually noted as \( X, Y, Z \) etc. with possible annotations. Each element \( P \) of \( \mathcal{P} \) has an arity, denoted \( \text{arity}(P) \). The formulas of the system are:

\[
\varphi, \psi ::= P(X, \ldots, X) \mid \varphi \land \psi \mid \varphi \lor \psi \mid \varphi \rightarrow \psi \mid \forall X.\varphi \mid \exists X.\varphi \mid \bot
\]

where \( P \) is an \( n \)-ary predicate and \( X, X_1, \ldots, X_n \in X_1 \). We follow Prawitz and introduce negation as a notation defined \( \neg \varphi ::= \varphi \rightarrow \bot \). A formula of the form \( P(X_1, \ldots, X_n) \) is called an atom. We
borrow from Urzyczyn [16] the notion of pseudo-atom formula, which is a formula of one of the three forms: atom formula, a formula of the form $\exists X. \varphi$ or a formula of the form $\varphi_1 \lor \varphi_2$.

We do not include parentheses in the grammar since we actually understand the formulas as abstract syntax trees instead of strings, i.e. the derivation trees of the formulas according to the grammar. The nodes of such a tree are labelled with the cases of the above-mentioned grammar and each node has as many successors as there are non-terminal symbols in its case. In addition, we use in writing traditional disambiguation conventions for $\land$, $\lor$ and insert parentheses to further disambiguate whenever this is necessary. The connective $\rightarrow$ is understood as right associative so that $\varphi_1 \rightarrow \varphi_2 \rightarrow \varphi_3$ is equivalent to $\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_3)$. We also use the convention that quantifiers bind weaker than connectives so in an expression $\forall X. \varphi$ (and $\exists X. \varphi$) the range of quantification spans over the whole formula $\varphi$. In a formula $\varphi = \varphi_1 \rightarrow \cdots \rightarrow \varphi_n \rightarrow \varphi'$, where $\varphi'$ is a pseudo-atom, the formula $\varphi'$ is called the target of $\varphi$.

Each time we use the term subformula $\psi$ of $\varphi$, we implicitly mean a particular occurrence of $\psi$ in $\varphi$. This occurrence is in our text either unimportant or obvious from the context.

The set of free first-order variables in a formula $\varphi$, written $\text{FV}_1(\varphi)$, is defined as follows:

- $\text{FV}_1(\text{P}(X_1, \ldots, X_n)) = \{X_1, \ldots, X_n\}$,
- $\text{FV}_1(\varphi_1 \ast \varphi_2) = \text{FV}_1(\varphi_1) \cup \text{FV}_1(\varphi_2)$ where $\ast \in \{\land, \lor, \rightarrow\}$,
- $\text{FV}_1(\forall X. \varphi) = \text{FV}_1(\varphi) \setminus \{X\}$ where $\forall \in \{\exists, \forall\}$,
- $\text{FV}_1(\bot) = \emptyset$.

Other variables that occur in a formula are bound. Formulas that differ only in renaming of bound variables are $\alpha$-equivalent and we do not distinguish between them. To describe the binding structure of a formula we use a special bind function.

**Definition 2.1 (binding).**
We define the partial function $\text{bind}_\varphi$ recursively as follows:

- $\text{bind}_\varphi(\varphi, X) = \varphi$.
- If $\text{bind}_\varphi(\psi_1 \ast \psi_2, X) = \psi$ where $\ast \in \{\land, \lor, \rightarrow\}$ then $\text{bind}_\varphi(\psi_i, X) = \psi$ for $i = 1, 2$.
- If $\text{bind}_\varphi(\forall Y. \psi', X) = \psi$ where $\forall \in \{\exists, \forall\}$ and $X \neq Y$, then $\text{bind}_\varphi(\psi', X) = \psi$.
- If $\text{bind}_\varphi(\forall X. \psi', X) = \psi$ where $\forall \in \{\exists, \forall\}$, then $\text{bind}_\varphi(\psi', X) = \forall X. \psi'$.

In other cases the function is undefined.

For example $\text{bind}_{\bot \rightarrow \exists Y. \bot \rightarrow P(X)}(P(X), X) = \exists Y. \bot \rightarrow P(X)$.

This function has the following interesting properties.

**Proposition 2.2 (Properties of bind).**

1. The result of the partial function $\text{bind}_\varphi(\psi, X)$ is defined if and only if $\psi$ is a subformula of $\varphi$.
2. If $\text{bind}_\varphi(\psi, X) = \psi'$ and $X \in \text{FV}_\psi$ then $\psi'$ is the least subformula of $\varphi$ such that
   - $\psi$ is a subformula of $\psi'$ and
   - for each proper subformula $\psi''$ of $\psi'$ that contains $\psi$ as a subformula, $X \in \text{FV}_\psi''$.

**Proof.** In case (1), the proof for the part ‘if’ is by induction on the difference of the size between $\varphi$ and $\psi$. The proof of the part ‘only if’ is by induction over the number of recursive steps necessary to obtain the result.

In case (2), an easy induction on the difference between the size of $\psi$ and the size of $\psi'$.
Automata theory approach

For the definition of proof terms we assume that there is an infinite set of proof term variables $\chi_p$, usually noted as $x, y, z$ etc. with possible annotations. These can be used to form the following terms.

$$M, N, P ::= x \ | \ (M, N) \ | \ \pi_1 M \ | \ \pi_2 M \ | \ \text{in} \_1, \varphi_1 \ \varphi_2 M \ | \ \text{in} \_2, \varphi_1 \ \varphi_2 M \ | \ \text{case} \ M \ \text{of} \ [x : \varphi] N, [y : \psi] P \ | \ \lambda x : \varphi. M \ | \ MN \ | \ \lambda x : \varphi. M \ | \ MX \ | \ \text{pack} \ M, Y \ \text{to} \ \exists X. \varphi \ | \ \text{let} \ x : \varphi \ \text{be} \ M : \exists X. \varphi \ \text{in} \ N \ | \ \bot : \varphi M$$

where $x$ is a proof term variable, $\varphi, \psi$ are first-order formulas and $X, Y$ are first-order variables.

Due to Curry–Howard isomorphism the proof terms can serve as programs in a functional programming language. Their operational semantics is given in terms of reductions. Their full account of the meaning of the terms. In particular, $\langle M_1, M_2 \rangle$ represents the product aggregation construct and $\pi_i M$ for $i = 1, 2$ decomposition of the aggregation by means of projections. The terms $\text{in} \_1, \varphi_1 \ \varphi_2 M, \ \text{in} \_2, \varphi_1 \ \varphi_2 M$ reinterpreted the value of $M$ as one of type $\varphi_1 \lor \varphi_2$. At the same time $\text{case} \ M \ \text{of} \ [x : \varphi_1] N_1, [y : \varphi_2] N_2$ construct offers the possibility to make case analysis of a value in an $\lor$-type. This construct is available in functional programming languages in a more general form of algebraic types. The terms $\lambda x : \varphi. M$ and $M_1 M_2$ represent traditional function abstraction and application, respectively. The proof terms that represent universal quantifier manipulation make it possible to parametrize type with a particular value $\lambda x : \varphi. M$ and use the parametrized term for a particular case $M X$. The proof terms that introduce existential quantifier, $\text{pack} \ M, Y \ \text{to} \ \exists X. \varphi$, make it possible to hide behind a variable $X$ an actual realization of a construction that uses another individual variable $Y$. The abstraction obtained in this way can be exploited using $\text{let} \ x : \varphi \ \text{be} \ M : \exists X. \varphi \ \text{in} \ M_2$. At last the term $\bot : \varphi M$ corresponds to the break instruction.

We consistently annotate our proof terms with types although their use in programming languages would suggest to omit them. Still, there are at least two reasons for keeping the annotation. The first reason is that terms with full annotations are in bijection with inference rules of the logic. In this way they can serve as a handy notation for proofs. The second reason is that when one omits annotations the questions such as typechecking (i.e. given $\Gamma, M$ and $\varphi$, decide if $\Gamma \vdash M : \varphi$ is derivable) start to be unobvious. In particular the known algorithm for the typechecking problem in the fragment with $\forall$ and $\rightarrow$ only is known to be doubly exponential [9].

The environments ($\Gamma, \Delta$, etc. with possible annotations) in the proving system are finite sets of pairs $x : \varphi$ that assign formulas to proof variables. We assume here the restriction that if $x : \varphi \in \Gamma$ and $x : \psi \in \Gamma$ then $\varphi = \psi$. Given an environment $\Gamma = \{x_1 : \varphi_1, \ldots, x_n : \varphi_n\}$, we sometimes write by abuse of notation $\psi \in \Gamma$ to express that there is some $x$ such that $x : \psi \in \Gamma$. Similarly, we write $\Gamma \cup \{\psi\}$ for the environment $\Gamma \cup \{y : \psi\}$ where $y \notin \{x_1, \ldots, x_n\}$. At last we write $\Gamma \rightarrow \psi$ for the formula $\varphi_1 \rightarrow \cdots \rightarrow \varphi_n \rightarrow \psi$.

We write $\Gamma \vdash M : \varphi$ to express that the judgement is indeed derivable. The inference rules of the logic are presented in Figure 1. We have two kinds of free variables, namely free proof term variables and free first-order variables. The set of free proof-term variables is defined inductively as follows:

- $\text{FV}x = \{x\}$,
- $\text{FV}(M_1, M_2) = \text{FV}M_1 M_2 = \text{FV}M_1 \cup \text{FV}M_2$,
- $\text{FV}\pi_1 M = \text{FV}\pi_2 M = \text{FV}\text{in} \_1, \varphi \ | \ \varphi_2 M = \text{FV}\text{in} \_2, \varphi_1 \ | \ \varphi_2 M = \text{FV}\lambda x \ \\ \text{FV}\lambda x : \varphi. M = \text{FV}M \ \text{and} \ \text{FV}\text{pack} \ M, Y \ \text{to} \ \exists X. \varphi = \text{FV} \bot : \varphi M = \text{FV}M$,
- $\text{FV}\text{case} \ M \ \text{of} \ [x : \varphi_1] N_1, [y : \varphi_2] N_2 = \text{FV}M \cup (\text{FV}N_1 \ \backslash \ \{x\}) \cup (\text{FV}N_2 \ \backslash \ \{y\})$,
- $\text{FV}\lambda x : \varphi. M = \text{FV}M \ \backslash \ \{x\}$,
\[
\Gamma, x : \varphi \vdash x : \varphi \quad (\text{var})
\]
\[
\Gamma \vdash M_1 : \varphi_1 \quad \Gamma \vdash M_2 : \varphi_2 \quad (\land I)
\]
\[
\Gamma \vdash M : \varphi_1 \land \varphi_2 \quad (\land E1)
\]
\[
\Gamma \vdash M_1 : \varphi_1 \land \varphi_2 \quad \Gamma \vdash \pi_1 M : \varphi_1 \quad (\land E2)
\]
\[
\Gamma \vdash M : \varphi_1 \quad (\lor I) \quad \Gamma \vdash \text{in}_1, \varphi_1 \lor \varphi_2 M : \varphi_1 \lor \varphi_2 \quad (\lor I2)
\]
\[
\Gamma \vdash M_1 : \varphi_1 \lor \varphi_2 \quad \Gamma, x_1 : \varphi_1 \vdash N_1 : \varphi \quad \Gamma, x_2 : \varphi_2 \vdash N_2 : \varphi \quad (\lor E)
\]
\[
\Gamma, x : \varphi_1 \vdash M : \varphi_2 \quad \quad \Gamma \vdash M_1 : \varphi_1 \rightarrow \varphi_2 \quad \Gamma \vdash M_2 : \varphi_1 \quad (\rightarrow I)
\]
\[
\Gamma \vdash \lambda x : \varphi_1. M : \varphi_1 \rightarrow \varphi_2 \quad \Gamma \vdash M_1 : \varphi_1 \rightarrow \varphi_2 \quad \Gamma \vdash M_2 : \varphi_1 \quad (\rightarrow E)
\]
\[
\Gamma \vdash M : \varphi \quad \Gamma \vdash \lambda X M : \forall X. \varphi \quad (\forall I) \quad \Gamma \vdash M : \forall X. \varphi \quad \Gamma \vdash MY : \varphi[X := Y] \quad (\forall E)
\]
\[
\Gamma \vdash M : \varphi \quad \quad \Gamma \vdash \text{pack} M, Y \text{ to } \exists X. \varphi : \exists X. \varphi \quad (\exists I) \quad \Gamma \vdash M_1 : \exists X. \varphi \quad \Gamma, x : \varphi \vdash M_2 : \psi \quad (\exists E)\]
\[
\Gamma \vdash M : \bot \quad \Gamma \vdash \bot M : \varphi \quad (\bot E)
\]

* Under the eigenvariable condition \( X \notin FV(\Gamma, \psi) \).

**FIGURE 1** The rules of the intuitionistic first-order logic.

- \( \text{FVlet } x : \varphi \text{ be } M_1 : \exists X. \varphi \text{ in } M_2 = \text{FVM}_1 \cup (\text{FVM}_2 \setminus \{x\}) \).

Again, the terms that differ only in names of bound proof-term variables are considered \( \alpha \)-equivalent and are not distinguished. We extend the notation \( \text{FV}_1(M) \) to refer to all free term variables that occur in a term \( M \). This set is defined by induction as follows:

- \( \text{FV}_1(x) = \emptyset \),
- \( \text{FV}_1((M_1, M_2)) = \text{FV}_1(M_1M_2) = \text{FV}_1(M_1) \cup \text{FV}_1(M_2) \),
- \( \text{FV}_1(\pi_1 M) = \text{FV}_1(\pi_2 M) = \text{FV}_1(M) \),
- \( \text{FV}_1(\lambda X M) = \text{FV}_1(M) \cup \{X\} \),
- \( \text{FV}_1(\text{in}_1, \varphi_1 \lor \varphi_2 M) = \text{FV}_1(\text{in}_2, \varphi_1 \lor \varphi_2 M) = \text{FV}_1(\varphi_1) \cup \text{FV}_1(\varphi_2) \cup \text{FV}_1(M) \),
- \( \text{FV}_1(\lambda X : \varphi. M) = \text{FV}_1(\exists X. \varphi) = \text{FV}_1(M) \cup \text{FV}_1(\exists X. \varphi) \),
- \( \text{FV}_1(\text{case } M \text{ of } [x : \varphi_1] N_1, [y : \varphi_2] N_2) = \text{FV}_1(M) \cup \text{FV}_1(\varphi_1) \cup \text{FV}_1(N_1) \cup \{y\} \cup \text{FV}_1(N_2) \),
- \( \text{FV}_1(\text{let } x : \varphi \text{ be } M_1 : \exists X. \varphi \text{ in } M_2) = \text{FV}_1(\varphi) \cup \text{FV}_1(M_1) \cup \text{FV}_1(\exists X. \varphi) \cup \text{FV}_1(M_2) \).
Proof or program reconstruction based upon the rules of natural deduction presented in Figure 1 faces difficulty since one formula can be derived in several ways. The first ambiguity arises from the fact that a formula introduction rule can be directly followed by its elimination. Such situations can be eliminated by proof manipulation rules called traditionally $\beta$-reduction rules.

$$\begin{align*}
\pi_i(M_1, M_2) &\rightarrow_\beta M_i \\
\text{case in}_{i, \varphi_1 \lor \varphi_2} M \text{ of } [x_1 : \varphi_1] N_1, [x_2 : \varphi_2] N_2 &\rightarrow_\beta N_i[x_i := M] \\
&\text{ for } i = 1, 2 \\
(\lambda x : \varphi. M) N &\rightarrow_\beta M[x := N] \\
(\lambda M) Y &\rightarrow_\beta M[X := Y] \\
\text{let } x : \varphi \text{ be (pack } M_1, Y \text{ to } \exists X. \varphi) : \exists X. \varphi \text{ in } M_2 &\rightarrow_\beta M_2[x := M_1[Y := X]]
\end{align*}$$

These rules form the basic computation mechanism in the system understood as a programming language.

The second source of ambiguity comes from the fact that sometimes the presence of one kind of elimination blocks another one from meeting matching introduction. For instance application of a case analysis term to another one

$$\Gamma \vdash (\text{case } M \text{ of } [x : \varphi_1] N_1, [y : \varphi_2] N_2) N : \psi$$

can block the application to meet $\lambda$ abstraction while the form

$$\Gamma \vdash \text{case } M \text{ of } [x : \varphi_1] N_1 N, [y : \varphi_2] N_2 N : \psi$$

(provided that $x, y \notin \text{FV}(N)$) enables reduction of $N_1N$ and $N_2N$. This transformation can be written as a reduction rule

$$\begin{align*}
(\text{case } M \text{ of } [x : \varphi_1] N_1, [y : \varphi_2] N_2) N &\rightarrow_p \text{case } M \text{ of } [x : \varphi_1] N_1 N, [y : \varphi_2] N_2 N.
\end{align*}$$

Such reduction rules give potential to eliminate $\beta$ steps that are not immediately visible. Such rules are called permutation rules. The full set of these rules for our logic can be found in the work by de Groote [6] or Urzyczyn [16].

The first crucial property of the reduction rules is that the expression after reduction proves the same formula as the original one. The work of de Groote [6] contains (implicitly) the following result.

**THEOREM 2.3 (Subject reduction).**

Intuitionistic first-order logic has the subject reduction property, i.e. if $\Gamma \vdash M : \phi$ and $M$ reduces to $N$ using $\beta$ and permutation rules then $\Gamma \vdash N : \phi$.

Once none of the $\beta$ and permutation reduction rules can be applied the resulting term is called normal form. The following theorem shown by de Groote [6] states that each provable formula has a proof in normal form, which means that if we focus on normal forms only, we do not lose any possible programs in our program synthesis approach.

**THEOREM 2.4 (Normalization).**

Intuitionistic first-order logic is a strongly normalizing system with respect to $\beta$ and permutation reductions i.e. each reduction with help of these rules has a finite number of steps.
These theorems make it possible to conclude that each provable formula has a proof in a regulated, unambiguous normal form, called long normal form. We can define such a normal form syntactically, and this makes the proof or program search process easier.

2.2 Long normal forms

We restrict our attention to terms which are in long normal form (lnf in short). This restriction is not essential as the automata also accept terms in other forms. However this restriction makes it possible to prove the correspondence with first-order logic. The idea of long normal form for our logic is best explained by an example ([16] section 2.2), here expressed using propositional logic, a traditional abstracted form of our first-order formalism: suppose \( x : r \) and \( y : r \rightarrow p \lor q \). The long normal form of \( yx \) is

\[
\text{case } yx \text{ of } [a : p] \lnf_{p \lor q} a, [b : q] \lnf_{p \lor q} b.
\]

In this long normal form, the proof of the type \( p \lor q \) is represented by a term, the main constructor of which serves to build objects of disjunctive types, unlike in the original version. In this way the proof construction process may be more predictable as we can always assume that a proof of a disjunction is realized by a case term.

Our definitions follow those of Urzyczyn [16]. We can obtain normal forms from the symbol \( N \) of the following grammar:

\[
N \ ::= \ I \mid P \mid Q
\]

\[
I \ ::= \ \lambda X N \mid \lambda x : \varphi . N \mid \langle N, N \rangle \mid \lnf_{p \lor q} N \mid \lnf_{q \lor p} N \mid \text{pack } N, y \text{ to } \exists X. \varphi
\]

\[
P \ ::= \ x \mid PN \mid \pi_i P \mid PX
\]

\[
Q \ ::= \ \bot \varphi(P) \mid \text{case } P \text{ of } [x : \varphi] N, [y : \psi] N \mid \text{let } x : \varphi \text{ be } N : \exists X. \varphi \text{ in } P
\]

Terms obtained from the symbol

- \( I \) are called introductions,
- \( P \) are called proper eliminators and
- \( Q \) are called improper eliminators.

The long normal forms (lnfs) are defined recursively, using an extended version of BNF, as follows.

\[
L \ ::= \ E^0 \mid \lambda X L \mid \langle L, L \rangle \mid \lnf_{p \lor q} L \mid \lnf_{q \lor p} L \mid \text{pack } L, y \text{ to } \exists X. \varphi \mid \text{case } E \text{ of } [x : \varphi] L, [y : \psi] L \mid \bot : \psi(E) \mid \text{let } x : \varphi \text{ be } E : \exists X. \varphi \text{ in } L \mid E
\]

\[
E \ ::= \ x \mid EL \mid \pi_i E \mid EX
\]

In the grammar above, we use the symbol \( E^0 \) to denote elements generated from \( E \) that derive atomic formulas.

Notably, terms generated from the symbol \( E \) have an important proof theoretic property:

**Proposition 2.5 (Initial variable).**

If \( \Gamma \vdash N : \varphi \) is derivable and \( N \) is generated by the rule \( E \) in definition of long normal form then there is \( x : \psi \in \Gamma \) such that \( \varphi \) is a subformula of \( \psi \).
PROOF. The proof is by induction over the size of \( N \) and through the case analysis of the way \( N \) was generated. In case the size is 1, the conclusion immediately follows as \( N = x \) and \( x : \varphi \) must belong to \( \Gamma \) for the judgement to be derivable.

In case the size of \( N \) is greater than 1, we observe that the forms of proof terms available in the rule \( E \) require that some judgement \( \Gamma \vdash N' : \varphi' \) is derivable using rules in Figure 1 where \( N' \) is a subterm of \( N \) and \( \varphi \) is a subformula of \( \varphi' \). This combined with the result of the induction hypothesis applied to \( N' \) gives the required conclusion.

It is enough to consider only these forms when one wants to obtain provability of a formula.

**Proposition 2.6** (Long normal forms). If \( \Gamma \vdash M : \varphi \) then there is a long normal form \( N \) such that \( \Gamma \vdash N : \varphi \).

**Proof.** The proof is an adaptation of the proof of Corollary 2.26 in the work by Urzyczyn [16]. It is implicitly used in section 5 of the same work by Urzyczyn.

The design of automata that handle proof search in the first-order logic requires us to find out what are the actual resources the proof search should work with. We observe here that the proof search process — as it is in the case of the propositional intuitionistic logic — can be restricted to formulas that occur only as subformulas in the initial formula and the initial context. Of course this search process — as it is in the case of the propositional intuitionistic logic — can be restricted to what are the actual resources the proof should work with. We observe here that the proof search process — as it is in the case of the propositional intuitionistic logic — can be restricted to formulas that occur only as subformulas in the initial formula and the initial context. Of course this time we have to take into account first-order variables. The following proposition, which we know how to prove for long normal forms only, sets the observation in precise terms.

**Proposition 2.7**
Consider a derivation of \( \Gamma \vdash M : \varphi \) such that \( M \) is in the long normal form. Each judgement \( \Delta \vdash \psi \) that occurs in this derivation has the property that, for each formula \( \chi \in \Delta \cup \{ \psi \} \), \( \chi = \bot \) or there exists a subformula \( \chi' \) of formulas in \( \Gamma \cup \{ \varphi \} \) such that \( \chi = \chi' [X_1 := Y_1, \ldots, X_n := Y_n] \) where \( \text{FV} \chi' = \{ X_1, \ldots, X_n \} \) and where \( Y_1, \ldots, Y_n \) are some first-order variables.

**Proof.** Induction over the difference \( n_M - n_N \) where \( n_M \) is the size of \( M \) and \( n_N \) is the size of \( N \).

In case \( n_M - n_N = 0 \), we have \( M = N \) as all other terms in the derivation must be subterms of \( M \). Therefore, conclusion immediately follows.

In case \( n_M - n_N > 0 \), we have \( M \neq N \) and we observe that \( \Delta \vdash \psi \) is present in an assumption of some rule

\[
\frac{\cdots \Delta \vdash \psi \quad \cdots (R)}{\Delta' \vdash N' : \psi'}
\]

that occurs in the derivation of \( \Gamma \vdash M : \varphi \). By the induction hypothesis, we obtain that for each formula \( \chi \in \Delta' \cup \{ \psi' \} \), there exists a subformula \( \chi' \) of \( \Gamma \vdash \varphi \) such that \( \chi = \chi' [X_1 := Y_1, \ldots, X_n := Y_n] \) and \( X_1, \ldots, X_n \) are some first-order variables. In some of the cases, i.e. the rules \((\forall I)\), \((\land I)\), \((\lor I)\), \((\lor 2)\), all assumptions of the rule \((R)\) are such that \( \Delta = \Delta' \) and \( \psi \) is a subformula of \( \psi' \) or \( \bot \). In these cases the proof follows immediately. The rule \((\to I)\) is very similar, in this case \( \Delta' = \Delta, x : \psi'' \) and \( \psi \) is a subformula of \( \psi' \), but also \( \psi'' \) is a subformula of \( \psi' \) so the conclusion follows.

The remaining rules can be divided into two kinds such that the proof in each of the cases goes along the same lines.

The first kind contains the rules \((\forall I)\), \((\exists I)\). These rules have in their premises judgements of the form \( \Gamma \vdash M : \varphi[X := Y] \) (in case \((\forall I)\) this is so because of the renaming of the bound variable in the conclusion) where \( \Gamma \) and \( QX \psi \), where \( Q \in \{ \forall, \exists \} \), occur in the conclusion. By the induction hypothesis there is some subformula \( \chi \) of formulas in \( \Gamma \cup \{ \psi \} \) such that \( QX \psi = \chi [X_1 :=
The formula \( Y \) in (2) \( = \chi \). Consequently \( \chi = QX.\chi \) and \( \chi \) is a subformula of formulas in \( \Gamma \cup \{ \psi \} \) and we have \( \psi[X := Y] = \chi'[X := Y, X_1 := Y_1, \ldots, X_n := Y_n] \), which concludes the proof for this case.

The second kind contains all the remaining rules, i.e. \((\land E1), (\land E2), (\lor E), (\rightarrow E), (\forall E), (\exists E), (\bot E)\). Some of the assumptions in these rules fall under the case that \( \Delta \vdash N : \psi \) is such that \( \Delta = \Delta' \) and \( \psi \) is a subformula of \( \psi' \) or \( \bot \). In these cases the proof goes as in the previous cases. It has remaining cases \( \Delta = \Delta' \), but \( \psi \) is not a subformula of \( \psi' \). However, \( N' \) is in long normal form. An analysis of the rules brings us the conclusion that the term \( N \) must be generated by the rule \( E \) in definition of long normal forms (see page 11). We can now use Proposition 2.5 to obtain \( x : \psi'' \in \Delta \) such that \( \psi \) is a subformula of \( \psi'' \). By the induction hypothesis there is some subformula \( \chi'' \) of formulas in \( \Gamma \cup \{ \psi \} \) such that \( \psi'' = \chi'[X_1 := Y_1, \ldots, X_n := Y_n] \). Since \( \psi \) is a subformula of \( \psi'' \) and \( \chi'' \) differs from \( \psi'' \) only in the names of some object variables there is a subformula \( \chi \) of \( \chi'' \) that has the property that \( \psi = \chi[X_1 := Y_1, \ldots, X_n := Y_n] \), which concludes the proof in this case.

Further details are left to the reader. \( \square \)

We can generalize the property expressed in the proposition above and introduce a suitable indication that certain formulas emerge from other ones.

**Definition 2.8** (Emerging formulas). We say that a formula \( \psi \) emerged from \( \varphi \) when \( \psi = \bot \) or there exists a subformula \( \psi_0 \) of \( \varphi \) and a substitution \([X_1 := Y_1, \ldots, X_n := Y_n]\) with \( \text{FV}_1(\psi_0) = \{X_1, \ldots, X_n\} \) such that \( \psi = \psi_0[X_1 := Y_1, \ldots, X_n := Y_n] \). We say that an environment \( \Gamma \) emerged from \( \varphi \) when, for each its element \( x : \psi \), the formula \( \psi \) emerged from \( \varphi \).

### 3 Definition of Arcadian automata

In this section we define the notion of abstract automata that do not use the syntax of first-order logic, but are strong enough to handle the proving process in this formal system.

Our *Arcadian automaton* \( A \) is defined as a tuple \( \langle A, Q, q^0, \varphi^0, I, i, \text{fv} \rangle \), where

- \( A = \langle A, \leq, e, L, l \rangle \) is a finite tree, which formally describes a division of the automaton control into intercommunicating modules;
- \( Q \) is a set of states;
- \( q^0 \in Q \) is an initial state of the automaton;
- \( \varphi^0 \in A \) is an initial tree node of the automaton;
- \( I \) is a set of all instructions;
- \( i : Q \rightarrow \mathcal{P}(I) \) is a function that gives the set of instructions available in a given state; the function \( i \) must be such that every instruction from \( I \) is available in exactly one state;
- \( \text{fv} : A \rightarrow P(A) \) is a function that describes the binding; we assume that this function has the property that for each non-leaf node \( v \) of \( A \) it holds that \( \text{fv}(v) = \bigcup_{w \in B(v)} \text{fv}(w) \) where \( B(v) = \{ w \mid v \text{ succ } w \} \).

The set of states \( Q \) is partitioned into the set of existential states, written \( Q^\exists \), and universal ones, written \( Q^\forall \). Consequently, we have that \( Q = Q^\exists \cup Q^\forall \), and \( Q^\exists \cap Q^\forall = \emptyset \). Similarly, \( Q_a \) is the set of

---

\( ^1 \)The name Arcadian automata stems from the fact that a slightly different and weaker notion of *Eden automata* was developed before [14] to deal with the fragment of the intuitionistic first-order logic with \( \forall \) and \( \rightarrow \) and in which the universal quantifier occurs only on positive positions.
states that are available in \( a \in A \). As a result, \( \bigcup_{a \in A} Q_a = Q \) and \( Q_a \cap Q_b = \emptyset \) for \( a \neq b \). In addition we let \( Q_a^\gamma = Q_a \cap Q^\gamma \) and \( Q_a^\theta = Q_a \cap Q^\theta \). This implies that \( Q^\gamma = \bigcup_{a \in A} Q_a^\gamma \) and \( Q^\theta = \bigcup_{a \in A} Q_a^\theta \).

The requirement that instructions are assigned to exactly one state stems from the natural correspondence between automata and graphs with labelled edges. In such graphs nodes are states and the instructions are edges.

**Operational semantics of the automaton**

An instantaneous description (ID) of \( A \) is a tuple \( \langle q, \kappa, V, w, w', S \rangle \) where

- \( q \in Q \) is the current state;
- \( \kappa \) is the current node in \( A \);
- \( V \) is the working domain of the automaton, i.e. a set of eigenvariables, which can be represented for example as natural numbers;
- \( w : A \rightarrow V \) is the interpretation of bindings associated with \( \kappa \) by \( \text{fv}(\kappa) \); we sometimes call it the value of the first register of the automaton; we require here that \( \text{fv}(\kappa) \subseteq \text{dom}(w) \);
- \( w' : A \rightarrow V \) is the auxiliary interpretation of bindings that can be stored in this location of the ID to implement some operations; the interpretation \( w' \) is the value of the second register of the automaton, the role of which is discussed later;
- \( S \) is the store of the automaton, which is a set of pairs \( \langle \rho, v \rangle \) where \( \rho \in A \) and \( v : A \rightarrow V \) and we require that \( \text{fv}(\rho) \subseteq \text{dom}(v) \).

The description above contains certain well-formedness conditions, which must be assumed for these descriptions to be well defined.

**Definition 3.1**

Well-formed ID We say that an ID \( \langle q, \kappa, V, w, w', S \rangle \) is well-formed when all of the following conditions hold

- \( V \neq \emptyset \),
- \( \text{fv}(\kappa) \subseteq \text{dom}(w) \),
- \( \text{fv}(\kappa) \subseteq \text{dom}(w') \),
- for each \( \langle \rho, v \rangle \in S \) it holds that \( \rho \in A \) and \( v : A \rightarrow V \), and \( \text{fv}(\rho) \subseteq \text{dom}(v) \).

We choose as the initial ID the tuple \( \langle q^0, \varphi^0, V_0, \emptyset, \emptyset, \emptyset \rangle \) where \( V_0 \) is some non-empty set.

Intuitively speaking the automaton works as a device that discovers the knowledge contained in the tree \( A \). It can distinguish new items of interest in the domain of the discourse and these are stored in the set \( V \) while the facts concerning the elements of \( V \) are stored in \( S \). Traditionally, the control of the automaton is represented by the current state \( q \), which belongs to a ‘module’, i.e. element of \( A \), indicated by \( \kappa \). We can imagine the automaton as a device that tries to check if a particular piece of information encoded in the tree \( A \) is correct. In this view the piece of information that is being checked for correctness at a given point is represented by the current node \( \kappa \) combined with current interpretation of bindings \( w \). The auxiliary interpretation of bindings \( w' \) is used to temporarily hold an interpretation of some bindings.

At a given ID the automaton is in an internal state \( q \) looking at the node \( \kappa \) of the tree \( A \). The value of the function \( w \) for a given free variable \( \gamma \) is the element of \( V \) (a first-order variable) that \( \gamma \) is currently bound to. The set \( S \) is a store of facts combined with associated bindings, and elements of \( V \) are used to denote variables in the proof.

We have 7 kinds of instructions in our automata. We give here their operational semantics. Let us assume that we are in a current ID of the form \( \langle q, \kappa, V, w, w', S \rangle \). The operation of the instructions is
defined as follows, where we assume \( q' \in Q, \rho, \rho' \in A \).

1. \( q : \text{store} \ \rho, \rho', q' \) turns the current ID into 
\( (q', \rho', V, w, \emptyset, S \cup \{(\rho, (w \ll w')_{|\text{fv}(\rho)})\}) \),
i.e. it moves control to the state \( q' \) and changes the current node in \( A \) to \( \rho' \), adding a new fact 
\( (\rho, (w \ll w')_{|\text{fv}(\rho)}) \) to the store;

2. \( q : \text{jmp} \ \rho, q' \) turns the current ID into \( (q', \rho, V, w'', \emptyset, S) \), where 
\( (w \ll w')_{|\text{fv}(\kappa)} \subseteq w'' \) and \( \text{fv}(\rho) \subseteq \text{dom} w \ll w' \),
i.e. it moves control to the state \( q' \) and it changes the current node in \( A \) to \( \rho \), with appropriate change of the variable interpretation;

3. \( q : \text{new} \ \rho, q' \) turns the current ID into \( (q', \rho, V \cup \{X\}, w, \emptyset, S) \), where \( X \notin V \),
i.e. it extends the discourse domain \( V \) with a new element \( X \);

4. \( q : \text{check} \ \rho, \rho', q' \) turns the current ID into \( (q', \rho', V, w, \emptyset, S) \), the instruction is applicable only when there is a pair \( (\rho, v) \in S \) such that \( v(\kappa) = w(\kappa) \),
i.e. it moves control to the state \( q' \) and it changes the current node in \( A \) to \( \rho' \), but with no change of the variable interpretation, the main feature of the instruction is that it is applicable only when appropriate fact is stored in \( S \) and thus it can provide alternative flows of the computation;

5. \( q : \text{instL} \ \rho, \rho', q' \) turns the current ID into \( (q', \rho', V \cup \{X\}, w, \emptyset, S \cup \{(\rho, w''|_{\text{fv}(\rho)})\}) \), the instruction is applicable only when there is a node \( \rho'' \in A \) such that \( \rho'' \succ \rho \) and where \( w'' = ([\rho'' := X] \ll w) \ll w' \) and \( X \notin V \),
i.e. it moves control to the state \( q' \) and it changes the current node in \( A \) to \( \rho' \), but also it synchronously extends the discourse domain \( V \) with a new element \( X \), adds a new fact 
\( (\rho, w''|_{\text{fv}(\rho)}) \) to the store and checks for presence of an appropriate predecessor of \( \rho \) in \( A \);

6. \( q : \text{instR} \ \rho, q' \) turns the current ID into \( (q', \rho, V, w'', \emptyset, S) \), the instruction is applicable only when an additional condition is met that \( \kappa \succ \rho \) and where \( w'' = [\kappa := X] \ll w|_{\text{fv}(\rho)} \) and \( \gamma \in \text{fv}(\rho) \backslash \text{fv}(\kappa) \) and \( X \in V \),
i.e. it moves control to the state \( q' \) and checks if the current node is an appropriate successor of \( \rho \);

7. \( q : \text{load} \ \rho, q' \) turns the current ID into \( (q', \rho, V, w'', v, S) \), where 
\( (w \ll w')_{|\text{fv}(\kappa)} \subseteq w'' \) and \( \text{fv}(\rho) \subseteq \text{dom} w'', \) and \( v : A \rightarrow V \),
i.e. it moves control to the state \( q' \) and loads an appropriately guessed interpretation to the additional register.

Instructions above are applicable only if the resulting ID is well formed.

Note that the only instruction that fills the second register of the automaton is \text{load}. All other instructions take into account the value of the register, but also erase its content. Therefore, the register can be exploited only by execution of \text{load} immediately followed by some other instruction.

It is also interesting to observe that the set of instructions contains, in addition to standard assembly-like instructions, two instructions \text{instL} and \text{instR} that deal with pattern instantiation.

The automata are alternating devices. This means that universal states split the computation into parallel threads (that do not communicate one with another), one for each applicable instruction, while in existential ones any applicable instruction can be used. The instructions \text{new} and \text{instL} extend the discourse domain in an uncontrolled way, which may lead to undecidability of many natural properties of the automata.

The following notion of acceptance is defined inductively. We say that the automaton \( A \) \textit{eventually accepts} from an ID \( a = (q, \kappa, V, w, w', S) \) when one of the conditions holds.
PROPOSITION 3.2 (Register monotonicity).

If the automaton $\mathcal{A}$ eventually accepts from a well-formed ID $w$ and the automaton $\mathcal{A}'$ in $w'$ eventually accepts, or if $w$ is well-formed and the automaton $\mathcal{A}$ eventually accepts from a well-formed ID $w'$ eventually accepts, where $c$ turns the ID $a$ into $a'$.

The definition above actually defines inductively a certain kind of tree, the nodes of which are IDs and children of a node are determined by the IDs obtained by executing available instructions. Such a tree is called a run of $\mathcal{A}$. In essence, we can view the process described above not only as a process of reaching acceptance, but also as a process of accepting the tree of IDs. In this light the automaton is eventually accepting from an initial ID if and only if the language of its ‘runs’ is not empty. As a result we can talk about the acceptance of such automata by referring to the emptiness problem. The set of all eventually accepting runs of an automaton $\mathcal{A}$ is written $L(\mathcal{A})$.

Here is a basic monotonicity property of Arcadian automata concerning extending $w$ to $\hat{w}$ while keeping bindings from $w$.

**PROPOSITION 3.2 (Register monotonicity).**

If the automaton $\mathcal{A}$ eventually accepts from a well-formed ID $a = \langle q, \kappa, V, w, w', S \rangle$ and $w \subseteq \hat{w}$ as well as $\hat{w}(x) = w'(x)$ for $x \in \text{dom} \hat{w} \cap \text{dom} w \setminus \text{dom} w$ then the ID $\hat{a} = \langle q, \kappa, V, \hat{w}, w', S \rangle$ is wellformed and the automaton $\mathcal{A}$ eventually accepts from $\hat{a}$.

**PROOF.** We inductively check that extending $w$ to $\hat{w}$ does not break the accepting run of the automaton, here seen as an evidence that the automaton eventually accepts. Formally:

Our proof is by induction on the run. Suppose $\mathcal{A}$ eventually accepts from $\langle q, \kappa, V, w, w', S \rangle$. We examine the conditions that made it possible to conclude so and show how they can be met in case $w$ is replaced by $\hat{w}$. Note that such replacement turns a well-formed ID $a$ into a well-formed ID $a_m$.

- For case 1 from the definition of acceptance (this is the base for induction), we need to show that adding elements to $w$ does not make new instructions available. But the set of available instructions depends only on $q$, which is in both cases the same. Therefore if there were no instructions available, there are still no instructions available and the state is still accepting.
- For case 2, analogously to the previous case, we have no new instructions available. Now we have to show that we can successfully execute the same instructions with $\hat{w}$ that we could with $w$:

1. **store**: the ID $a'$ resulting from execution of this instruction from $a$ differs from $a'_m$ resulting from execution of the same instruction from $a_m$ only in $w$ being replaced by $\hat{w}$. Indeed, the only remaining potential difference may be due to the fact that

   $$(w \ll w')|_{\text{fv}(\rho)} \neq (\hat{w} \ll w')|_{\text{fv}(\rho)}.$$  

   However, the condition $w \subseteq \hat{w}$ implies that the two functions are equal on $x \in \text{dom} w$, while the condition $\hat{w}(x) = w'(x)$ for $x \in \text{dom} \hat{w} \cap \text{dom} w' \setminus \text{dom} w$ implies that they are equal for $x \in \text{dom} w'$. Note that the conditions $w \subseteq \hat{w}$ and $\hat{w}(x) = \emptyset(x)$ hold for $x \in \text{dom} \hat{w} \cap \text{dom} \emptyset$ so we can apply the inductive hypothesis to $a'$ and $a'_m$.

2. **jmp**: this case holds by essentially the same proof as in case 1.

3. **new**: this case holds by essentially the same proof as in case 1.

4. **check**: $\hat{w}$ extends $w$, i.e. for arguments where $w$ was defined the value does not change, so $\nu(\rho) = \hat{w}(\kappa)$ still holds and the rest goes as in the case of the instruction.
Translation from formulas to automata

Automata theory approach

5. instL: this case holds by essentially the same proof as in case 1. The additional statement to note here is that condition \( \hat{w}(x) = w'(x) \) for \( x \in \text{dom} \hat{w} \cap \text{dom} w' \setminus \text{dom} w \) implies that \( ([\rho] := X) \ll w \ll ([\rho'] := X) \ll w' \).

6. instR: this case holds by essentially the same proof as in case 1. The additional statement to note here is that condition \( w \subseteq \hat{w} \) implies that \( [\gamma := X] \ll w|_{fv(\rho)} = [\gamma := X] \ll \hat{w}|_{fv(\rho)} \), assuming that \( a \) is well formed.

7. load: this case holds by essentially the same proof as in case 1. We have to observe here that the first register \( \hat{w}'' \) of the ID \( a' \) resulting from application of the instruction to \( a \) fulfils the condition \( (w \ll w')|_{fv(\kappa)} \subseteq w'' \). We observe here that the well formedness of \( a \) and the conditions that \( w \subseteq \hat{w} \) and that \( \hat{w}(x) = w'(x) \) for \( x \in \text{dom} \hat{w} \cap \text{dom} w' \setminus \text{dom} w \) imply together that \( (w \ll w')|_{fv(\kappa)} = (\hat{w} \ll w')|_{fv(\kappa)} \).

- For case 3, analogously to the previous case, the instructions that were available are still available and we can use the same one when \( \hat{w} \) is used in place of \( w \). □

4 Translation from formulas to automata

We can now define an Arcadian automaton \( \mathcal{A}_\varphi \) = \( \langle A, Q, q_0^Y, \varphi, I, i, fv \rangle \) that faithfully simulates the proof process for the formula \( \varphi \). For technical reasons we assume that the formula \( \varphi \) is closed and each bound variable is unique (which is possible by \( \alpha \)-conversion). This restriction is not essential since the provability of a formula with free variables is equivalent to the provability of its universal closure. The automaton works in such a way that its run reflects a derivation in the system presented in Figure 1. At each moment, the ID \( \langle q, \varphi, V, w, w', S \rangle \) represents the content of a suitable judgement \( \Gamma \vdash M : \psi \) in such a way that \( \Gamma \) is encoded in \( S \), and \( \psi \) is encoded in \( \varphi \) and \( w \). The automaton in an existential state chooses a rule that can be applied to derive the current formula \( \psi \) and then some additional work is done to implement the rule that concludes by entering a universal state that directs the work to handling the assumptions of the chosen derivation rule.

The components of the automaton are defined below. Let us stress that in annotations of the states that we use below we identify different occurrences of the same subformula (see Negative example below for explanation).

- \( A = \langle A, \leq \rangle \) is the syntax tree of the formula \( \varphi \).
- \( Q = Q^Y \cup Q^3 \) where

\[
Q^Y = Q^{Y,0} \cup Q^{Y,\lor} \cup Q^{Y,\to} \cup Q^{Y,\exists} \cup Q^{Y,\bot} \cup Q^{Y,\text{axiom}}\quad \text{with}
\]

- \( Q^{Y,0} = \{ q_\psi^Y \mid \text{for all subformulas } \psi \text{ of } \varphi \}, \) states of this form serve to direct control of the automaton to all the assumptions in a derivation rule that is handled,
- \( Q^{Y,\lor} = \{ q_\psi^Y, \theta \lor \chi \mid \text{for all subformulas } \psi, \theta \lor \chi \text{ of } \varphi \}, \) states of this form serve to handle additional work necessary to implement the \( (\lor E) \) rule,
- \( Q^{Y,\to} = \{ q_\psi^Y, \theta \to \chi \mid \text{for all subformulas } \psi, \theta \to \chi \text{ of } \varphi \}, \) states of this form serve to handle additional work necessary to implement the \( (\to E) \) rule,
- \( Q^{Y,\exists} = \{ q_\psi^Y, \exists X. \xi \mid \text{for all subformulas } \psi, \exists X. \xi \text{ of } \varphi \}, \) states of this form serve to handle additional work necessary to implement the \( (\exists E) \) rule,
- \( Q^{Y,\bot} = \{ q_\psi^Y, \bot \mid \text{for all subformulas } \psi \text{ of } \varphi \}, \) states of this form serve to handle additional work necessary to implement the \( (\bot E) \) rule,
- \( Q^{Y,\text{axiom}} = \{ q_\psi^Y \text{axiom} \}, \) states of this form serve to terminate run.

- \( Q^3 = \{ q_\psi^3 \mid \text{for all subformulas } \psi \text{ of } \varphi \}, \) states of this form serve to choose a derivation rule to handle the current target formula.
Automata theory approach

### Structural decomposition instructions

1. \( \varphi_1 \to \varphi_2 \)  
   \( q_{\varphi_1 \to \varphi_2} : \text{store } \varphi_1, \varphi_2, q_{\varphi_2}^3 \)  
   \( \Rightarrow (q_{\varphi_1 \to \varphi_2}, \varphi_1 \to \varphi_2, V, w, \emptyset, S) \to \)  
   \( (q_{\varphi_2}^3, \varphi_2, V, w, \emptyset, S \cup \{ (\varphi_1, w | \text{fv}(\varphi_1)) \}) \)

2. \( \varphi_1 \land \varphi_2 \)  
   \( q_{\varphi_1 \land \varphi_2} : \text{jmp } \varphi_1, q_{\varphi_1}^3 \)  
   \( \Rightarrow (q_{\varphi_1 \land \varphi_2}, \varphi_1 \land \varphi_2, V, w, \emptyset, S) \to (q_{\varphi_2}^3, \varphi_1, V, w, \emptyset, S) \)

3. \( \varphi_1 \lor \varphi_2 \)  
   \( q_{\varphi_1 \lor \varphi_2} : \text{jmp } \varphi_1, q_{\varphi_1}^3 \)  
   \( \Rightarrow (q_{\varphi_1 \lor \varphi_2}, \varphi_1 \lor \varphi_2, V, w, \emptyset, S) \to (q_{\varphi_2}^3, \varphi_1, V, w, \emptyset, S) \)

4. \( \forall X. \varphi \)  
   \( q_{\forall X. \varphi} : \text{new } \varphi, q_{\varphi}^3 \)  
   \( \Rightarrow (q_{\forall X. \varphi}, \forall X. \varphi, V, w, \emptyset, S) \to \)  
   \( (q_{\varphi}^3, \forall X. \varphi, V \cup \{ Y \}, [\forall X. \varphi := Y] \ll w, \emptyset, S) \)  
   where \( Y \not\in V \)

5. \( \exists X. \varphi \)  
   \( q_{\exists X. \varphi} : \text{instR } \varphi, q_{\varphi}^3 \)  
   \( \Rightarrow (q_{\exists X. \varphi}, \exists X. \varphi, V, w, \emptyset, S) \to \)  
   \( (q_{\varphi}^3, \exists X. \varphi, V, [\exists X. \varphi := Y] \ll w | \text{fv}(\exists X. \varphi), \emptyset, S) \)  
   where \( Y \in V \)

**Figure 2** Structural decomposition instructions of the automaton.

In addition \( Q'_\varphi = \{ q_{\varphi}^v \mid q_{\varphi}^v \in Q^v \} \cup \{ q_{\forall X. \varphi}^v, q_{\exists X. \varphi}^v \in Q^v \} \) as well as \( Q^3_\varphi = \{ q_{\varphi}^3 \mid q_{\varphi}^3 \in Q^3 \} \).

- \( q_{\varphi}^3 \) is the initial state (which reflects that the goal of the proving process is some \( \varphi' \) obtained from \( \varphi \) by an application of a suitable substitution) and the initial tree node is \( \varphi \).
- \( I \) and \( i \) are presented in Figures 2 and 3. We describe them in more detail below.
- \( \text{fv} : A \to P(A) \) is defined so that \( \text{fv}(\varphi) = \{ \text{bind}_\varphi(\psi, X) \mid X \in \text{FV}(\psi) \} \).

Figures 2 and 3 present patterns of possible instructions in \( I \). Each of the instruction patterns starts with a state of the form \( q_{\varphi}^v \) or of the form \( q_{\varphi}^v \), where \( \varphi \) is a quantifier (\( \forall \) or \( \exists \)), \( \psi \) is a subformula of \( \varphi \) and \( \bullet \) is a subformula of \( \varphi \) with the main symbols being one of \( \lor, \land, \land, \exists \). For each of the patterns we assume \( I \) contains all the instructions that result from instantiating the pattern with all possible subformulas that match the form of \( \psi \) (e.g. in case \( \psi = \psi_1 \to \psi_2 \) we take all the subformulas with \( \to \) as the main symbol). The function \( i : Q \to P(I) \) is defined so that for a state \( q_{\varphi}^v \) it returns all the instructions which start from this state. In addition to the instructions, the figures present the way an ID is transformed by each of the instructions. This serves to facilitate understanding of the proofs.

As the figures suggest, the instructions of the automaton can be divided into two groups — structural decomposition instructions and non-structural ones. The structural instructions are used to decompose a formula into its structural subformulas. On the left-hand side of each of the structural
Non-structural instructions

(6) \( q^\varphi_\varphi^\varphi : \text{jmp} \varphi, q^\varphi_\varphi \)
\( \Rightarrow \langle q^\varphi_\varphi, \varphi, V, w, \emptyset, S \rangle \rightarrow \langle q^\varphi_\varphi, \varphi, V, w, \emptyset, S \rangle \)

(7) \( q^\varphi_\varphi : \text{jmp} \varphi_1 \land \varphi_2, q^\varphi_\varphi \land \varphi_\varphi \) for \( i = 1, 2 \)
\( \Rightarrow \langle q^\varphi_\varphi, \varphi, V, w, \emptyset, S \rangle \rightarrow \langle q^\varphi_\varphi \land \varphi_\varphi, \varphi_1 \land \varphi_2, V, w', \emptyset, S \rangle \)

(8) \( q^\varphi_\varphi : \text{load} \varphi, q^\varphi_\varphi, \varphi \_{\mathcal{X}} \)
\( \Rightarrow \langle q^\varphi_\varphi, \varphi, V, w, \emptyset, S \rangle \rightarrow \langle q^\varphi_\varphi, \varphi, V, w, w', S \rangle \)

(9) \( q^\varphi_\varphi : \text{jmp} \varphi, q^\varphi_\varphi \rightarrow \varphi \)
\( \Rightarrow \langle q^\varphi_\varphi, \varphi, V, w, \emptyset, S \rangle \rightarrow \langle q^\varphi_\varphi \rightarrow \varphi, \varphi, V, w, \emptyset, S \rangle \) where \( w \subseteq \bar{w} \)

(10) \( q^\varphi_\varphi : \text{jmp} \forall \mathcal{X}, \varphi, V \_{\mathcal{X}} \varphi \)
\( \Rightarrow \langle q^\varphi_\varphi, \varphi, V, w, \emptyset, S \rangle \rightarrow \langle q^\varphi_\forall \mathcal{X}, \varphi, V, w, \emptyset, S \rangle \)

(11) \( q^\varphi_\varphi : \text{load} \varphi, q^\varphi_\exists \mathcal{X}, \varphi \)
\( \Rightarrow \langle q^\varphi_\varphi, \varphi, V, w, \emptyset, S \rangle \rightarrow \langle q^\forall \exists \mathcal{X}, \varphi, V, w, w', S \rangle \)

(12) \( q^\varphi_\varphi : \text{jmp} \varphi, q^{\top}_\varphi \)
\( \Rightarrow \langle q^\varphi_\varphi, \varphi, V, w, \emptyset, S \rangle \rightarrow \langle q^{\top}_\varphi, \varphi, V, w, \emptyset, S \rangle \)

(13) \( q^\varphi_\varphi : \text{check} \varphi, \varphi, q^\varphi_\text{axiom} \)
\( \Rightarrow \langle q^\varphi_\varphi, \varphi, V, w, \emptyset, S \rangle \rightarrow \langle q^\varphi_\text{axiom}, \varphi, V, w, \emptyset, S \rangle \)

(14) \( q^{\top}_\varphi : \text{jmp} \theta \lor \mathcal{X}, q^\varphi_\theta \)
\( \Rightarrow \langle q^{\top}_\varphi, \theta \lor \mathcal{X}, \varphi, V, w, w', S \rangle \rightarrow \langle q^\varphi_\theta, \theta \lor \mathcal{X}, V, w, \emptyset, S \rangle \)

(15) \( q^{\top}_\varphi : \text{store} \theta, \varphi, q^\top_\varphi \)
\( \Rightarrow \langle q^{\top}_\varphi, \theta \lor \mathcal{X}, \varphi, V, w, w', S \rangle \rightarrow \langle q^\varphi_\theta, \varphi, V, w', \emptyset, S' \rangle \)
where \( S' = S \cup \{ \langle \theta, w'' \rangle_{\text{fv}(\theta)} \} \), \( w'' = w \ll w' \)

(16) \( q^{\top}_\varphi : \text{store} \mathcal{X}, \varphi, q^\top_\varphi \)
\( \Rightarrow \langle q^{\top}_\varphi, \theta \lor \mathcal{X}, \varphi, V, w, S \rangle \rightarrow \langle q^\varphi_\theta, \varphi, V, w', \emptyset, S' \rangle \)
where \( S' = S \cup \{ \langle \mathcal{X}, w'' \rangle_{\text{fv}(\mathcal{X})} \} \), \( w'' = w \ll w' \)

(17) \( q^\varphi_\psi, \text{jmp} \psi \rightarrow \varphi, q^\varphi_\psi \rightarrow \varphi \)
\( \Rightarrow \langle q^\varphi_\psi, \psi \rightarrow \varphi, \psi, V, w, \emptyset, S \rangle \rightarrow \langle q^\varphi_\psi \rightarrow \varphi, \psi, V, w, \emptyset, S \rangle \)

(18) \( q^\varphi_\psi, \text{jmp} \psi, q^\psi_\psi \)
\( \Rightarrow \langle q^\varphi_\psi, \psi \rightarrow \varphi, \psi, V, w, \emptyset, S \rangle \rightarrow \langle q^\psi_\psi, \psi, V, w, \emptyset, S \rangle \)

(19) \( q^\varphi_\exists \mathcal{X}, \psi : \text{ jmp } \exists \mathcal{X}, \psi, q^\exists \mathcal{X}, \psi \)
\( \Rightarrow \langle q^\varphi_\exists \mathcal{X}, \psi, \psi, V, w', S \rangle \rightarrow \langle q^\exists \mathcal{X}, \psi, \exists \mathcal{X}, \psi, V, w', \emptyset, S' \rangle \)

(20) \( q^\varphi_\exists \mathcal{X}, \psi : \text{inst} \psi, \psi, q^\psi_\psi \)
\( \Rightarrow \langle q^\varphi_\exists \mathcal{X}, \psi, \psi, V, w', S \rangle \rightarrow \langle q^\exists \mathcal{X}, \psi, \exists \mathcal{X}, \psi, V \cup \{ Y \}, w'', \emptyset, S' \rangle \)
where \( w'' = \{ \exists \mathcal{X}, \psi := Y \} \ll w' \ll w', S' = S \cup \{ \psi, w'' \}_{\text{fv}(\psi)} \}, Y \notin V \)

(21) \( q^{\top, \bot}_\varphi : \text{ jmp } \bot, q^{\top, \bot}_\varphi \)
\( \Rightarrow \langle q^{\top, \bot}_\varphi, \bot, \psi, V, w, \emptyset, S \rangle \rightarrow \langle q^{\top, \bot}_\varphi, \bot, V, w, \emptyset, S \rangle \)

Figure 3 Non-structural instructions of the automaton.
Automata theory approach

FIGURE 4 Syntax tree of the formula $\phi_{\text{pos.}}$

instructions we present the formula the instruction decomposes. The other rules represent operations that manipulate other elements of an ID with possible change of the goal formula, which is illustrated in the following example.

Example Consider the formula: $\varphi_{\text{pos}} = (\forall X. P(X)) \to \forall Y. \exists X. P(X)$.

In order to build the Arcadian automaton for that formula first we have to build the tree $A$ of it, which is shown in Figure 4. In the figure we attach to each node a unique capital letter (A, B, etc.) and use it further in the text to refer to the subformula starting at the corresponding node.

The set $I$ of available instructions consists of elements (the numbers in parentheses are the numbers of corresponding patterns presented in Figures 2 or 3):

(1) $q^\forall_A$: store B, D, $q^3_D$ (19) $q^\forall_{A, \exists}$: jmp E, $q^3_E$
(4) $q^\forall_B$: new C, $q^3_C$ (19) $q^\forall_{D, \exists}$: jmp E, $q^3_E$
(4) $q^\forall_D$: new E, $q^3_E$ (19) $q^\forall_{E, \exists}$: jmp E, $q^3_E$
(5) $q^\forall_E$: instR F, $q^3_F$ (20) $q^\forall_{A, \exists}$: instL E, A, $q^3_A$
(10) $q^\forall_F$: jmp B, $q^3_B$ (20) $q^\forall_{D, \exists}$: instL E, D, $q^3_D$
(10) $q^\forall_F$: jmp B, $q^3_B$ (20) $q^\forall_{E, \exists}$: instL E, E, $q^3_E$

The instructions available for any $a \in A$ are:

(6) $q^3_a$: jmp $a, q^\forall_a$ (8) $q^3_a$: load $a, q^\forall_a$
(9) $q^\forall_a$: jmp $a, q^\forall_a$ (11) $q^3_a$: load $a, q^\forall_a$
(12) $q^3_a$: jmp $a, q^\forall_a$ (13) $q^3_a$: check $a, a, q^\forall_a$
(21) $q^\forall_a$: jmp $\bot, q^3_\bot$

The set of states can be easily written using the definition. To define $fv$ we need to determine bind function first. We let $\text{bind}_A(C, x) = B$ and $\text{bind}_A(F, x) = E$; therefore $fv(C) = \{B\}$, $fv(F) = \{E\}$ and $fv(\psi) = \emptyset$ for $\psi \neq C, \psi \neq F$. We let $q^0 = q^3_A$ and $\varphi^0 = \varphi_{\text{pos.}}$. The initial ID is

$$\{q^3_A, A, \{c\}, \emptyset, \emptyset, \emptyset\}.$$
A successful run of the automaton is as follows:

- \textbf{jmp} A, q_A^\forall (rule (6), initial instruction leads to the structural decomposition of the main connective $\to$);
- \textbf{store} B, D, q_D^\exists (rule (1), as the result of the decomposition, the formula $B$ is moved to the environment, and the formula $D$ becomes the proof goal);
- \textbf{jmp} D, q_D^\forall (rule (6), we progress to the structural decomposition of $\forall$);
- \textbf{new} E, q_E^\forall (rule (4), we introduce fresh eigenvariable, say $X_1$, for the universal quantifier);
- \textbf{jmp} E, q_E^\exists (rule (6), we progress to the structural decomposition of $\exists$);
- \textbf{instR} F, q_F^\exists (rule (5), we produce a witness for the existential quantifier, which can be just $X_1$);
- \textbf{jmp} B, q_B^\exists (rule (10), we progress now with the non-structural rule that handles instantiation of the universal assumption from the node $B$);
- and now we can conclude with \textbf{check} B, B, q_{axiom} (rule (13)) that directly leads to acceptance.

The detailed description of the automaton IDs in the run is presented in Figure 5.

\textbf{Negative example} Consider the formula

$$\phi_{div} = (((\forall X. Q(X)) \to P) \to P) \to P;$$

its tree is presented in Figure 6. Obviously it does not have an inhabitant (i.e. proof). The run of the corresponding automaton is infinite, and its main loop of states begins from $q_A^\exists$. We proceed first with the instruction

1. \textbf{store} B, H, q_H^\exists (rule 1) and then to
2. \textbf{jmp} H, q_H^\forall (rule 9), which leads to
3. \textbf{jmp} C, q_C^\exists (rule 18) in one of the branches of the run, which in turn leads to
4. \textbf{store} D, F, q_F^\exists, (rule 1).
We observe that the subformula at the node \( F \) is identical to the subformula at the node \( H \) so \( q^3_F = q^3_H \) and the control moves to the point (2) above so the loop is closed. Note that any run of the automaton contains this loop.

**From derivability questions to IDs** A proof search process in the style of Ben-Yelles [2] works by solving derivability questions of the form \( \Gamma \vdash \psi \). We relate this style of proof search to our automata model by a translation of such a question into an ID of the automaton. Suppose that the initial closed formula is \( \phi \). We define the ID of \( A_{\phi} \) that corresponds to \( \Gamma \vdash \psi \) by exploiting the conclusion of Proposition 2.7. This proposition makes it possible to associate a substitution \( w_\psi \) and a subformula \( \psi_0 \) of the original one such that \( w_\psi(\psi_0) = \psi \) as well as substitutions \( w_\chi \) and subformulas \( \chi_0 \) such that \( w_\chi(\chi_0) = \chi \) for each assignment \( x: \chi \in \Gamma \). The resulting ID is

\[
a_{\Gamma,\psi} = (q^3_{\psi_0}, \psi_0, V_{\Gamma,\psi}, w_\psi, \emptyset, S_{\Gamma,\psi})
\]

where \( S_{\Gamma,\psi} = \{ (\chi_0, \psi_\chi) \mid x: \chi \in \Gamma \} \) and \( V_{\Gamma,\psi} = \text{FV}_1(\Gamma, \psi) \cup \{ X_\ast \} \), where \( X_\ast \not\in \text{FV}_1(\Gamma, \psi) \), as well as \( w_\psi(\psi_0) = \psi \).

## 5 Properties of the translation

We can now demonstrate that the proving process in first-order logic leads to a run of an Arcadian automaton.

**Lemma 5.1** For each \( \varphi \) if \( \Gamma \vdash M: \psi \) is derivable, \( M \) is a long normal form and \( \Gamma \) as well as \( \psi \) emerged from \( \varphi \) then \( A_{\varphi} \) eventually accepts from the ID \( (q^3_{\psi_0}, \psi_0, V_{\Gamma,\psi}, w_\psi, \emptyset, S_{\Gamma,\psi}) \), where \( w_\psi(\psi_0) = \psi \) and \( \text{dom} w_\psi = \text{fv}(\psi_0) \).

**Proof.** The proof is by induction over the derivation of \( M \).

**If the last rule is the \((\text{var})\) rule** then we can apply the non-structural instruction (13) that checks that the formula \( w_\psi(\psi_0) \) is in the store \( S_{\Gamma,\psi} \). Then the resulting state \( q^Y_{\text{axiom}} \) is an accepting state.

**If the last rule is the \((\land)\) rule** then \( \psi = \psi_1 \land \psi_2 \) and we have shorter derivations for \( \Gamma \vdash M_1: \psi_1 \) and \( \Gamma \vdash M_2: \psi_2 \), which by the induction hypothesis implies that \( A_{\varphi} \) eventually accepts from
the IDs

\[ \langle q_{\psi_{i0}}, \psi_{i0}, V_{\gamma, \psi_i}, w_{\psi_{i}}, \emptyset, S_{\gamma, \psi_i} \rangle \]

for \( i = 1, 2 \) where we note that \( w_{\psi_i} = w_\psi, S_{\gamma, \psi_i} = S_{\gamma, \psi} \) and \( V_{\gamma, \psi_i} = V_{\gamma, \psi} \). We can now use the rule (6) to turn the existential state \( q_{\psi_i}^{\exists} \) into the universal one \( q_{\psi_i}^{\forall} \) for which there are two instructions available in (2), and these turn the ID to the corresponding one among two mentioned above.

If the last rule is the \((\land E\!i)\) rule for \( i = 1, 2 \) then we know that \( \psi = \psi_i \) for one of \( i = 1, 2 \) and \( \Gamma \vdash M' : \psi_1 \land \psi_2 \) is derivable through a shorter derivation, which means by the induction hypothesis that \( \mathbb{A}_\psi \) eventually accepts from the ID \( \langle q_{\exists}^{\exists}, \psi_{10}, \psi_{10}, \psi_{10}, V_{\gamma, \psi_{1}}, w_{\psi_{1}}, \emptyset, S_{\gamma, \psi_{1}} \rangle \) where actually \( w_{\psi_{1}} \subseteq w_{\psi} \) and \( \psi_{1} \subseteq \text{dom} w_{\psi_{1}} \) for both \( i = 1, 2 \). Moreover, \( S_{\gamma, \psi_{1}} = S_{\gamma, \psi} \) and \( V_{\gamma, \psi_{1}} = V_{\gamma, \psi} \). This ID can be obtained from the current one using respective non-structural instruction presented at (7).

If the last rule is the \((\lor \! i)\) rule for some \( i = 1, 2 \) then we know that \( \Gamma \vdash M' : \psi_i \) is derivable by a shorter derivation, which means by the induction hypothesis that \( \mathbb{A}_\psi \) eventually accepts from the ID

\[ \langle q_{\psi_{1}}, \psi_{10}, V_{\gamma, \psi_i}, w_{\psi_i}, \emptyset, S_{\gamma, \psi_i} \rangle \]

where actually \( S_{\gamma, \psi_i} = S_{\gamma, \psi} \) and \( V_{\gamma, \psi_i} = V_{\gamma, \psi} \). Moreover \( w_{\psi_i} \subseteq w_{\psi} \) so by Proposition 3.2 for the automaton \( \mathbb{A}_\psi \) it eventually accepts from the ID \( \langle q_{\exists}^{\exists}, \psi_{1}, V_{\gamma, \psi_i}, w_{\psi_i}, \emptyset, S_{\gamma, \psi_i} \rangle \). We can now use the instruction (3) to reach this ID from the current one and obtain the required conclusion.

If the last rule is the \((\lor E)\) rule then

\[ M = \text{case } M_1 \text{ of } [x_1 : \psi_1] M_2, [x_2 : \psi_2] M_3 \]

and we have shorter derivations for \( \Gamma_1 \vdash M_1 : \psi_1 \lor \psi_2, \Gamma_2 \vdash M_2 : \psi \) and \( \Gamma_3 \vdash M_3 : \psi \) where \( \Gamma_1 = \Gamma, x_1 : \psi_1, \Gamma_2 = \Gamma, x_2 : \psi_2 \) and \( x_1, x_2 \) are fresh proof variables. Therefore, we know by the induction hypothesis that \( \mathbb{A}_\psi \) eventually accepts from the IDs

\[
\langle q_{\exists}^{\exists}, \psi_{10}, \psi_{20}, V_{\gamma, \psi_{1} \lor \psi_{2}}, w_{\psi_{1} \lor \psi_{2}}, \emptyset, S_{\gamma, \psi_{1} \lor \psi_{2}} \rangle, \\
\langle q_{\psi_{0}, \psi_{10}}, \psi_{0}, V_{\gamma, \psi_{1}}, w_{\psi_{1}}, \emptyset, S_{\gamma, \psi_{1}} \rangle, \\
\langle q_{\psi_{0}, \psi_{1}}, \psi_{1}, V_{\gamma, \psi_{2}}, w_{\psi_{2}}, \emptyset, S_{\gamma, \psi_{2}} \rangle, \\
\langle q_{\psi_{0}}, \psi_{2}, V_{\gamma, \psi_{1} \lor \psi_{2}}, w_{\psi_{1} \lor \psi_{2}}, \emptyset, S_{\gamma, \psi_{1} \lor \psi_{2}} \rangle, \\
\langle q_{\psi_{0}}, \psi_{1}, V_{\gamma, \psi_{2}}, w_{\psi_{2}}, \emptyset, S_{\gamma, \psi_{2}} \rangle \]

which is obtained by the induction hypothesis applied to the shorter derivations starting at the assumptions of the rule \((\lor E)\).

We can reach these three IDs from the current one in the following way. We execute the instruction (8) first. This instruction stores non-deterministically a partial function in the second register. This non-deterministic guess can be done so that the function is \( w_{\psi_{1} \lor \psi_{2}} \), note that \( w_{\psi_{1} \lor \psi_{2}} \) is consistent on variables in \( \text{dom} w_{\psi_{1} \lor \psi_{2}} \). The resulting ID is

\[ \langle q_{\psi_{0}}, \psi_{10}, \psi_{20}, \psi, V_{\gamma, \psi}, w, w_{\psi_{1} \lor \psi_{2}}, S_{\gamma, \psi} \rangle \]

We can see that instruction (14) turns this ID to the first ID in (I), (15) to the second ID in (I), and (16) to the third ID in (I).

If the last rule is the \((\to I)\) rule then \( \psi = \psi_1 \to \psi_2 \) and we have a shorter derivation for \( \Gamma, x : \psi_1 \vdash M_1 : \psi_2 \), which by the induction hypothesis implies that \( \mathbb{A}_\psi \) eventually accepts from
implies that

\[
(q_{\psi_0}^{3}, \psi_2, \psi_0, V_{\psi_2}, w_{\psi_2}, \emptyset, S_{\psi_2}),
\]

where \( \Gamma' = \Gamma, \chi : \psi_1 \) and \( w_{\psi_2}(\psi_0) = \psi_2 \). We note here that \( S_{\psi_2} = S_{\psi} \cup \{ (\psi_1, w_{\psi_1}) \} \) where \( w_{\psi_1}(\psi_0) = \psi_1 \) and \( V_{\psi_2} = V_{\psi} \).

We observe now that the instruction (6) transforms the current ID to \( (q_{\psi_0}^{3}, \psi_0, V_{\psi}, w_{\psi}, \emptyset, S_{\psi}) \) where \( \psi_0 = \psi_1 \to \psi_2 \) and \( w_{\psi}(\psi_0) = \psi \). The instruction (1) adds an appropriate element to \( S_{\psi} \) and turns the ID to the awaited \( (q_{\psi_0}^{3}, \psi_2, V_{\psi_2}, w_{\psi_2}, \emptyset, S_{\psi_2}) \).

**If the last rule is the \((\to E)\) rule** then we have shorter derivations for \( \Gamma \vdash M_1 : \psi' \to \psi \) and \( \Gamma \vdash M_2 : \psi' \). Note that by Proposition 2.7, both \( \psi' \to \psi \) and \( \psi' \) emerged from \( \psi \).

The induction hypothesis implies here that \( A_{\psi} \) eventually accepts from the IDs

\[
(q_{\psi_0}^{3}, \psi_0, V_{\psi}, w_{\psi}, \emptyset, S_{\psi}),
\]

\[
(q_{\psi_0}^{3}, \psi_0, V_{\psi}, w_{\psi}, \emptyset, S_{\psi}').
\]

Note that \( S_{\psi} = S_{\psi} \) and \( V_{\psi} = V_{\psi} \). We can now use the instruction (9) to turn the current ID into

\[
(q_{\psi_0}^{3}, \psi_0, V_{\psi}, w_{\psi}, \emptyset, S_{\psi}),
\]

which can be turned into the desired two IDs with the instructions (17) and (18), respectively.

**If the last rule is the \((\forall I)\) rule** then \( \psi = \forall X.\psi_1 \) and we have a shorter derivation for \( \Gamma \vdash M_1 : \psi_1 \) (where \( X \) is a fresh variable by the eigenvariable condition), which by the induction hypothesis implies that \( A_{\psi} \) eventually accepts from the ID

\[
(q_{\psi_0}^{3}, \psi_0, V_{\psi}, w_{\psi}, \emptyset, S_{\psi}),
\]

where \( w_{\psi}(\psi_1) = \psi_1, S_{\psi} = S_{\psi} \) and \( V_{\psi} = V_{\psi} \cup \{ X \} \).

We observe now that the instruction (6) transforms the current ID to \( (q_{\psi_0}^{3}, \psi, V_{\psi}, w_{\psi}, \emptyset, S_{\psi}) \) and then the instruction (4) adds appropriate element to \( V_{\psi} \) and turns the ID to the awaited one.

**If the last rule is the \((\forall E)\) rule** then we know that \( \psi = \psi_1[X := Y] \) where \( \Gamma \vdash M' : \forall X.\psi_1 \) is derivable through a shorter derivation, which means by the induction hypothesis that \( A_{\psi} \) eventually accepts from the ID

\[
(q_{\psi_0}^{3}, \forall X.\psi_0, V_{\psi}, \forall X.\psi_1, w_{\psi}, \emptyset, S_{\forall X.\psi_1}),
\]

where \( w_{\psi}(\forall X.\psi_1) = \forall X.\psi_1 \). We observe in addition that it holds that \( w_{\forall X.\psi_1}(\forall X.\psi_1) \subseteq w_{\psi} \) and \( \text{fv}(\forall X.\psi_1) \subseteq \text{dom}w_{\psi} \) as well as \( S_{\forall X.\psi_1} = S_{\psi} \) and \( V_{\forall X.\psi_1} = V_{\psi} \). Moreover, Proposition 3.2 implies that in addition \( A_{\psi} \) eventually accepts from the ID

\[
(q_{\psi_0}^{3}, \forall X.\psi_0, V_{\psi}, \forall X.\psi_1, w_{\psi}, \emptyset, S_{\forall X.\psi_1}).
\]

This ID can be obtained from the current one using the instruction (10).

**If the last rule is the \((\exists I)\) rule** then \( \psi = \exists X.\psi_1 \) and we have a shorter derivation for \( \Gamma \vdash M_1 : \psi_1[X := Y] \) for some \( Y \) that occurs in \( \Gamma \). Note that \( \psi_1[X := Y] \) emerged from \( \psi \) by Proposition 2.7.
Now, the induction hypothesis implies that $A_\psi$ eventually accepts from the ID

$$\langle q^3_\psi, \psi_0, V_\Gamma, \psi_1, w_\psi[X := Y], \emptyset, S_\Gamma, \psi_1 \rangle,$$

where $w_\psi[X := Y](\psi_0) = \psi_1[X := Y]$, $S_\Gamma, \psi_1 = S_\Gamma, \psi$ and $V_\Gamma, \psi_1 = V_\Gamma, \psi$.

We observe now that the instruction (6) transforms the current ID to $\langle q^4_\psi, \psi, V_\Gamma, w_\psi, \emptyset, S_\Gamma, \psi \rangle$. The instruction (5) turns subsequently the ID to the awaited one.

**If the last rule is the ($\exists E$) rule** then we know that $\Gamma \vdash M_1 : \exists X. \psi_1$ and $\Gamma, x : \psi_1 \vdash M_2 : \psi$ are derivable through shorter derivations, which means by the induction hypothesis, that $A_\psi$ eventually accepts from IDs

$$\langle q^3_{\exists X. \psi_0}, \exists X. \psi_0, V_\Gamma, \exists X. \psi_1, w_{\exists X. \psi_1}, \emptyset, S_\Gamma, \exists X. \psi_1 \rangle,$$

$$\langle q^3_{\psi_0}, \psi_0, V_\Gamma, \psi_1, w_\psi, \emptyset, S_\Gamma, \psi_1 \rangle$$

where $w_{\exists X. \psi_1}(\exists X. \psi_0) = \exists X. \psi_1$, $w_\psi(\psi_0) = \psi$ and $\Gamma' = \Gamma, x : \psi_1$, and this consequently means that $S_{\Gamma', \psi} = S_{\Gamma, \psi} \cup \{(\psi_0, \psi')\}$ and $V_{\Gamma', \psi} = V_{\Gamma, \psi} \cup \{X\}$ where $w' = [\exists X. \psi_0 := X] \ll w_{\exists X. \psi_1}[w(\psi_0)]$. Note that $x$ is a fresh proof variable by definition and $X$ is a fresh variable by the eigenvariable condition.

We observe that the current ID can be transformed to

$$\langle q^\psi_{\psi_0, \exists X. \psi_0}, \psi_0, V_\Gamma, \psi_1, w_\psi, \emptyset, S_\Gamma, \psi_1 \rangle$$

by the non-structural instruction (11). This, in turn, is transformed to the IDs (III) by non-structural instructions (19) and (20), respectively.

**If the last rule is the ($\wedge E$) rule** then we know $\Gamma \vdash M' : \bot$ is derivable through a shorter derivation, which means by the induction hypothesis that $A_\psi$ eventually accepts from the ID $\langle q^3_{\bot}, \bot, V_{\Gamma, \bot}, w_\bot, \emptyset, S_{\Gamma, \bot} \rangle$. Note that $w_\bot = \emptyset$. By Proposition 3.2 the automaton eventually accepts from the ID $\langle q^3_{\bot}, \bot, V_{\Gamma, \bot}, w_\psi, \emptyset, S_{\Gamma, \bot} \rangle$. This ID is accessible from the current ID by subsequent execution of instructions (12) and then (21).

We need a proof in the other direction. To express the statement of the next lemma we define $\Gamma_S = x_{\psi_1, w_1} : w_1(\psi_1), \ldots, x_{\psi_n, w_n} : w_n(\psi_n)$ where $S = \{\psi_1, w_1, \ldots, \psi_n, w_n\}$.

**Lemma 5.2**

If $A_\psi$ eventually accepts from the ID $\langle q^3_\psi, \psi, V, w, \emptyset, S \rangle$ then there is a proof term $M$ such that $\Gamma_S \vdash M : w(\psi)$.

**Proof.** The proof is by induction over the definition of the eventually accepting ID with cases depending on the first instruction that is executed. A more condensed form of the proof is given as tables in Figures 7 and 8. The tables show how to construct the term $M$ bottom-up from the run of the automaton. The tables include rules of the form

$$M_1, \ldots, M_k \leftrightarrow M$$

where $M_1, \ldots, M_k$ are terms reconstructed from states reached in one step from the current state. The expression $M$ shows how to reconstruct the resulting term for the current state, based upon terms $M_1, \ldots, M_k$.

Before we present the details of the proof, observe that only instructions (3), (6), (7), (8), (9), (10), (11), (12), (13) are available for states of the form $q^3_\psi$. 


Structural decomposition instructions: term reconstruction

(1) \( \varphi_1 \rightarrow \varphi_2 \quad q_{\varphi_1 \rightarrow \varphi_2}^\gamma : \text{store } \varphi_1, \varphi_2, q_{\varphi_2}^\exists \\
M \leftarrow \lambda x.M \)

(2) \( \varphi_1 \land \varphi_2 \quad q_{\varphi_1 \land \varphi_2}^\gamma : \text{jmp } \varphi_1, q_{\varphi_1}^\exists \\
M \leftarrow M \\
q_{\varphi_1 \land \varphi_2}^\gamma : \text{jmp } \varphi_2, q_{\varphi_2}^\exists \\
M \leftarrow M \)

(3) \( \varphi_1 \lor \varphi_2 \quad q_{\varphi_1 \lor \varphi_2}^\gamma : \text{jmp } \varphi_1, q_{\varphi_1}^\exists \\
M \leftarrow \text{in}_1 M \\
q_{\varphi_1 \lor \varphi_2}^\gamma : \text{jmp } \varphi_2, q_{\varphi_2}^\exists \\
M \leftarrow \text{in}_2 M \)

(4) \( \forall X. \varphi \quad q_{\forall X. \varphi}^\gamma : \text{new } \varphi, q_{\varphi}^\exists \\
M \leftarrow \lambda X.M \)

(5) \( \exists X. \varphi \quad q_{\exists X. \varphi}^\gamma : \text{instR } \varphi, q_{\varphi}^\exists \\
M \leftarrow \text{pack } M, Y \text{ to } \exists X. \varphi[Y := X] \)

Figure 7 Structural decomposition instructions of the automaton together with term generation rules.

We can immediately see that if one of the instructions (3) from Figure 2 is used then the induction hypothesis applied to resulting IDs brings the assumption of the respective rule \((\lor I_i)\) for \(i = 1, 2\) and we can apply it to obtain the conclusion.

Then taking the non-structural instruction (6) moves control to one of the instructions present in Figure 2 and these move control to IDs from which the induction hypothesis gives the assumptions of the introduction rules \((\rightarrow I), (\land I), (\lor I), (\exists I)\) respectively.

Next taking the non-structural instructions (8), (9), (11), (12) moves control to further non-structural rules in Figure 3 and these move control to IDs from which the induction hypothesis gives the assumptions of the elimination rules \((\lor E), (\rightarrow E), (\exists E)\) and \((\land E)\), respectively. At the same time the instructions (7), (10) move control directly to IDs from which the induction hypothesis gives the assumptions of the elimination rules \((\land E), (\forall E)\), respectively.

At last the non-structural instruction (13) directly represents the use of the \((\var)\) rule.

More details of the reasoning can be observed by referring to relevant parts in the proof of Lemma 5.1 and adapting them to the current situation. We provide here explicitly one interesting case.

If the first executed instruction is (8) then it turns the current ID to

\( \langle q_{\varphi, \psi_1 \lor \psi_2}^\gamma, \varphi, V, w', w'', S \rangle \)

for some interpretation \(w''\) and some \(w' \supseteq w\). As \(q_{\varphi, \psi_1 \lor \psi_2}^\gamma\) is a universal state, all available instructions must be executed, i.e. (14), (15) and (16). Therefore, we obtain three respective IDs

\( \langle q_{\psi_1 \lor \psi_2}^\gamma, \psi_1 \lor \psi_2, V, w_{\psi_1 \lor \psi_2}, \emptyset, S \rangle, \langle q_{\varphi}^\exists, \varphi, V, w_{\varphi}, \emptyset, S_1 \rangle, \langle q_{\varphi}^\exists, \varphi, V, w_{\varphi}, \emptyset, S_2 \rangle. \)
Non-structural instructions: term reconstruction

(6) \[ q_{\varphi}^3 : \text{jmp } \varphi, q_{\varphi}^\triangledown \]
\[ M \mapsto M \]
the correct variant is chosen in corresponding structural instruction

(7) \[ q_{\varphi^i}^3 : \text{jmp } \varphi_1 \land \varphi_2, q_{\varphi^1 \land \varphi_2}^3 \text{ for } i = 1, 2 \]
\[ M \mapsto \pi_i M \text{ where } \varphi_i = \varphi \]

(8) \[ q_{\varphi}^3 : \text{load } \varphi, q_{\varphi, \theta \land X}^\triangledown \]
\[ M_{(14), M_{(15)}, M_{(16)} \leftarrow \text{case } M_{(14)} \text{ of } [x_1 : \theta] M_{(15)} x_1, [x_2 : \chi] M_{(16)} x_2 \]

(9) \[ q_{\varphi}^3 : \text{jmp } \varphi, q_{\varphi, \psi \rightarrow \varphi}^\triangledown \]
\[ M_1, M_2 \mapsto M_1 M_2 \]

(10) \[ q_{\varphi}^3 : \text{jmp } \forall X \cdot \varphi, q_{\forall X \cdot \varphi}^3 \]
\[ M \mapsto MY \]
where \( Y = w(\forall X \cdot \varphi) \)

(11) \[ q_{\varphi}^3 : \text{load } \varphi, q_{\varphi, \exists X \cdot \theta} \]
\[ M_1, M_2 \mapsto \text{let } x \text{ be } M_1 \text{ in } M_2 x \]

(12) \[ q_{\varphi}^3 : \text{jmp } \varphi, q_{\varphi, \bot}^\triangledown \]
\[ M \mapsto \bot M \]

(13) \[ q_{\varphi}^3 : \text{check } \varphi, \varphi, q_{\text{axiom}}^\triangledown \]
\[ x_{(\rho, \nu)} \text{ when } \nu(\rho) = w(\varphi) \]

(14) \[ q_{\varphi, \theta \land X}^\triangledown : \text{jmp } \theta \land X, q_{\theta \land X}^3 \]
\[ M \mapsto M \]

(15) \[ q_{\varphi, \theta \land X}^\triangledown : \text{store } \theta, \varphi, q_{\varphi}^3 \]
\[ M \mapsto \lambda x_{(\theta, w'_1)_{|\nu|}}^\triangledown M \]

(16) \[ q_{\varphi, \theta \land X}^\triangledown : \text{store } \chi, \varphi, q_{\varphi}^3 \]
\[ M \mapsto \lambda x_{(\chi, w'_1)_{|\nu|}}^\triangledown M \]

(17) \[ q_{\varphi, \psi \rightarrow \varphi}^\triangledown : \text{jmp } \psi \rightarrow \varphi, q_{\psi \rightarrow \varphi}^3 \]
\[ M \mapsto M \]

(18) \[ q_{\varphi, \psi \rightarrow \varphi}^\triangledown : \text{jmp } \psi, q_{\psi}^3 \]
\[ M \mapsto M \]

(19) \[ q_{\varphi, \exists X \cdot \psi}^\triangledown : \text{jmp } \exists X \cdot \psi, q_{\exists X \cdot \psi}^3 \]
\[ M \mapsto M \]

(20) \[ q_{\varphi, \exists X \cdot \psi}^\triangledown : \text{instL } \psi, \varphi, q_{\varphi}^3 \]
\[ M \mapsto \lambda x_{(\psi, w''_{|\nu|})_{|\nu|}}^\triangledown M \]

(21) \[ q_{\psi, \bot}^\triangledown : \text{jmp } \bot, q_{\bot}^3 \]
\[ M \mapsto M \]

Figure 8 Non-structural instructions of the automaton together with term generation rules. The table shows how to reconstruct terms given the reconstruction of terms from subsequent states. We use here notation from Figure 3.

where \( S_1 = S \cup \{\psi_{10}, w^3_{|\nu|}(\psi_{10})\} \), \( S_2 = S \cup \{\psi_{20}, w^3_{|\nu|}(\psi_{20})\} \) with \( w^3 = w' \ll w'' \). Moreover, \( w' \ll w'' \ll w_\psi \psi_1 \psi_2 \) and \( w' \ll w'' \ll w_\psi \psi_1 \psi_2 \) by definition of the jmp and store instructions.
Consequently, \( w_\varphi(\varphi) = w_\varphi'(\varphi) = w(\varphi) \) and \( w_{\psi_1 \lor \psi_2}(\psi_{10} \lor \psi_{10}) = w_{\psi_1 \lor \psi_2}(\psi_{10}) \lor w_{\psi_1 \lor \psi_2}(\psi_{10}) = w^3_{\text{fv}(\psi_{10})}(\psi_{10}) \lor w^3_{\text{fv}(\psi_{20})}(\psi_{20}) \).

We obtain now from these three IDs by the induction hypothesis terms \( M_1, M_2, M_3 \), respectively, such that \( \Gamma_3 \vdash M_1 : w_{\psi_1 \lor \psi_2}(\psi_{10} \lor \psi_{20}) \), \( \Gamma_{S_1} \vdash M_2 : w_\varphi(\varphi) \) and \( \Gamma_{S_2} \vdash M_3 : w_\varphi'(\varphi) \).

We can now use the rule \((\lor E)\) to derive

\[
\Gamma_3 \vdash M : w(\varphi)
\]

where \( M = \text{case } M_1 \text{ of } [x_1 : \psi_1](M_2[x := x_1]), [x_2 : \psi_2](M_3[x' := x_2]) \) with \( \psi_1 = w_{\psi_1 \lor \psi_2}(\psi_{10}) \) and \( x \) being the variable in \( \Gamma_{S_1} \) to which \( \psi_1 \) is assigned as well as \( \psi_2 = w_{\psi_1 \lor \psi_2}(\psi_{20}) \) and \( x' \) being the variable in \( \Gamma_{S_2} \) to which \( \psi_2 \) is assigned. This concludes the proof in this case. \( \square \)

We can now combine the two recently proved lemmas into our main theorem.

**Theorem 5.3 (Main theorem).**

The provability in intuitionistic first-order logic is reducible to the emptiness problem for Arcadian automata, i.e. given a closed formula \( \varphi \) of intuitionistic first-order logic there is an automaton \( A_\varphi \) such that \( \vdash \varphi \) is derivable if and only if \( L(A_\varphi) \) is non-empty.

**Proof.** Let \( \varphi \) be a formula of the intuitionistic first-order logic. The emptiness problem for \( A_\varphi \) is equivalent to checking if the initial ID of this Arcadian automaton is eventually accepting. In case there is a proof of \( \varphi \) then there is also one in long normal form by Proposition 2.6. We can now apply Lemma 5.1 and obtain a run of \( A_\varphi \). On the other hand if the language accepted by \( A_\varphi \) is non-empty then \( A_\varphi \) eventually accepts from the initial ID \((q_0^3, \varphi, \{c\}, \emptyset, \emptyset, \emptyset)\). This implies by Lemma 5.2 that \( \vdash M : \varphi \) is derivable for some term \( M \). \( \square \)

### 6 Conclusions and further work

We propose a notion of automata that can simulate search for proofs in normal form in the full intuitionistic first-order logic, which can be viewed by the Curry–Howard isomorphism as a program synthesis for a simple functional language. This notion enables the possibility to apply automata theoretic techniques to inhabitant search in this logic system. Although the emptiness problem for such automata is undecidable (as the logic is, \([13]\)) the notion brings a new perspective to the proof search process, which can reveal new classes of formulas for which the proof search can be made decidable. In particular those automata, together with earlier investigations \([13, 14]\), bring to the attention that decidable procedures must constrain the growth of the subset \( V \) in IDs of automata presented here.

By design, our automata find only terms in *total discharge convention* \([11]\), i.e. such that if there is more than one variable of a given type available for use at some point in program, the most recently introduced one is used. This does not influence completeness of the search for inhabitants, but it has an effect on program synthesis. In order to check that it is not a big limiting factor we checked how many of the functions in real world programs are in total discharge convention by analysing the source code of the Glasgow Haskell Compiler (GHC). It turns out that 74% of the functions defined there are in total discharge convention, so using it does not excessively restrict program synthesis.\(^2\) This limitation can also be avoided by application of the technique presented by Barendregt et al. \([12]\).

\(^2\)The code and instructions needed to reproduce our results are available at [http://www.mimuw.edu.pl/~maciekz/HaskellTdcStats](http://www.mimuw.edu.pl/~maciekz/HaskellTdcStats).
Although the main value of the presented automata is in exposing natural resources that are inherent to the proving process, the efficiency of this approach is also of interest. However, it needs further investigation since this would go beyond the range of the current study. In particular, a comparison to other approaches (semantic tableau methods [3] or connection calculus [10]) is worth developing.

Another interesting path of study is to use the model of automata presented here to conduct proof development in classical logic. We believe that the classical lambda calculus $\lambda C$ of Ariola, Herbelin and Sabry [1] can be adapted to work with first-order logic and our automata can model proof search in such a system.

References

[1] Z. M. Ariola, H. Herbelin and A. Sabry. A type-theoretic foundation of continuations and prompts. In Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming, ICFP ’04, pp. 40–53. ACM, 2004.
[2] C.-B. Ben-Yelles. Type-Assignment in the Lambda-Calculus; Syntax and Semantics. PhD Thesis, Mathematics Department, University of Wales, Swansea, UK, 1979.
[3] E. W. Beth. Semantic Entailment and Formal Derivability. Noord-Hollandsche Uitg. Mij., Amsterdam, 1955.
[4] S. Broda and L. Damas. On long normal inhabitants of a type. Journal of Logic and Computation, 15, 353–390, 2005.
[5] B. Düdder, M. Martens and J. Rehof. Staged composition synthesis. In Proceedings of ESOP 2014, vol. 8410 of LNCS, Z. Shao., ed., pp. 67–86. Springer, 2014.
[6] P. de Groote. On the strong normalisation of intuitionistic natural deduction with permutation-conversions. Information and Computation, 178, 441–464, 2002.
[7] E. Filiot, J.-M. Talbot and S. Tison. Tree automata with global constraints. International Journal of Foundations of Computer Science, 21, 571–596, 2010.
[8] S. Hetzl. Applying tree languages in proof theory. In Proceedings of LATA 2012, vol. 7183, A.-H. Dediu and C. Martin-Vide., eds, pp. 301–312. Springer, 2012.
[9] A. Kozubek. Problem Sprawdzania Typu dla Logiki Pierwszego Rzędu. Master’s Thesis, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, 2006.
[10] J. Otten. Non-clausal connection-based theorem proving in intuitionistic first-order logic. In Proceedings of the 2nd International Workshop Automated Reasoning in Quantified Non-Classical Logics (ARQNL 2016) affiliated with the International Joint Conference on Automated Reasoning (IJCAR 2016)., Coimbra, Portugal, July 1, 2016, vol. 1770 of CEUR Workshop Proceedings, C. Benzmüller and J. Otten., eds, pp. 9–20. CEUR-WS.org, 2016.
[11] D. Prawitz. Natural Deduction. Almqvist and Wiksell, Sweden, 1965.
[12] A. Schubert, W. Dekkers and H. P. Barendregt. Automata theoretic account of proof search. In 24th EACSL Annual Conference on Computer Science Logic (CSL 2015), vol. 41 of LIPIcs, S. Kreutzer., ed, pp. 128–143, Dagstuhl, 2015.
[13] A. Schubert, P. Urzyczyn and K. Zdanowski. On the Mints hierarchy in first-order intuitionistic logic. In Proceedings of FoSSaCS 2015, vol. 9034 of LNCS, A. Pitts., ed., pp. 451–465. Springer, 2015.
[14] A. Schubert, P. Urzyczyn and D. Walukiewicz-Chrzaszcz. Restricted positive quantification is not elementary. In Proceedings of TYPES 2014, vol. 39 of LIPIcs, H. Herbelin, P. Letouzey and M. Sozeau., eds, pp. 251–273, Dagstuhl, 2015.
Automata theory approach

[15] M. Takahashi, Y. Akama and S. Hirokawa. Normal proofs and their grammar. *Information and Computation*, **125**, 144–153, 1996.

[16] P. Urzyczyn. Intuitionistic games: determinacy, completeness, and normalization. *Studia Logica*, **104**, 957–1001, 2016.

[17] M. Zielenkiewicz and A. Schubert. Automata theory approach to predicate intuitionistic logic. In *Proc. of LOPSTR ’2016*, vol. **10184** of LNCS, M. Hermenegildo and P. López-García., eds. Springer, 2017.

Received 30 October 2017