Automata Theory Approach to Predicate Intuitionistic Logic

Maciej Zielenkiewicz(✉) and Aleksy Schubert

Institute of Informatics, University of Warsaw, Warsaw, Poland
{maciekz,alx}@mimuw.edu.pl

Abstract. Predicate intuitionistic logic is a well established fragment of dependent types. According to the Curry-Howard isomorphism proof construction in the logic corresponds well to synthesis of a program the type of which is a given formula. We present a model of automata that can handle proof construction in full intuitionistic first-order logic. The automata are constructed in such a way that any successful run corresponds directly to a normal proof in the logic. This 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.

1 Introduction

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, the automata become 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 of Düdder et al. [3], which was powerful enough to synthesise 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. [11] who defined context-free grammars that can be used for proof search in propositional intuitionistic logic, which is, by

M.V. Hermenegildo and P. Lopez-Garcia (Eds.): LOPSTR 2016, LNCS 10184, pp. 345–360, 2017.
DOI: 10.1007/978-3-319-63139-4_20

Curry-Howard isomorphism, equivalent to program search in the simply typed λ-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 limitation to the total discharge form can be avoided by means of techniques developed by Schubert, Dekkers and Barendregt [8].

A different approach to abstract machinery behind program search process was proposed by Broda and Damas [2] who developed a formula-tree proof method. This technique provides a realisation 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 [6]. In this paper, Hetzl characterises a class of proofs in intuitionistic first-order logic recognisable by *tree automata with global equalities and disequalities* (TAGED) [4]. The characterisation makes it possible to recognise 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.

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 in 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 a role the subformula property plays in the proving process.
- The resource that serves to represent eigenvariables which occur in the process is distinguished. This abstraction is important as the variables play a crucial role in complexity results concerning the logic [9,10].
- The automata enable the possibility of getting rid of the particular syntactical 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.

Organisation of the paper. We fix the notation and present intuitionistic first-order logic in Sect. 2. Next, we define our automata in Sect. 3. We summarise the account in Sect. 4.

2 Preliminaries

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

A prefix closed set of strings $\mathbb{N}^*$ over $\mathbb{N}$ is called a *carrier of a tree*. A tree is a tuple $\langle A, \leq, L, l\rangle$ where A is a carrier of the tree, $\leq$ is the prefix order on $\mathbb{N}^*$, L is the set of labels and $l : A \to L$ is the *labelling function*. Whenever the set of labels and the labelling function are clear from the context, we abbreviate the quadruple to the tuple $\langle A, \leq\rangle$. Since the formula notation makes it easy, we sometimes use a subtree φ of A to actually denote a node in A at which φ starts.

2.1 Intuitionistic First-Order Logic

The basis for our study is the first-order intuitionistic logic (for more details see e.g. the work of Urzyczyn [12]). We assume that we have a set of predicates $\mathcal{P}$ that can be used to form atomic formulae and an infinite set $\mathcal{X}_1$ of first-order variables, usually noted as X, Y, Z etc. with possible annotations. Each element P of $\mathcal{P}$ has an arity, denoted arity(P). The formulae of the system are:

$$\varphi, \psi ::= \mathrm{P}(X_1, \ldots, X_n) \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \to \varphi_2 \mid \forall X.\varphi \mid \exists X.\varphi \mid \bot$$

where P is an n-ary predicate and $X, X_1, \ldots, X_n \in \mathcal{X}_1$. We follow Prawitz and introduce negation as a notation defined $\neg\varphi ::= \varphi \to \bot$. A formula of the form $\mathrm{P}(X_1, \ldots, X_n)$ is called an *atom*. A *pseudo-atom formula* 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 $\varphi_1 \vee \varphi_2$. We do not include parentheses in the grammar since we actually understand the formulas as abstract syntax trees instead of strings. The tree is traditionally labelled with the cases of the above mentioned grammar. We assume that, for a given case in the grammar, the corresponding node of the tree has as many sons as there are non-terminal symbols in the case. In addition, we use in writing traditional disambiguation conventions for $\wedge, \vee$ and insert parentheses to further disambiguate whenever this is necessary. The connective $\to$ is understood as right-associative so that $\varphi_1 \to \varphi_2 \to \varphi_3$ is equivalent to $\varphi_1 \to (\varphi_2 \to \varphi_3)$. In a formula $\varphi = \varphi_1 \to \cdots \to \varphi_n \to \varphi'$, where φ' is a pseudo-atom, the formula φ' is called the *target of* φ.

Each time we use the term *subformula* ψ of φ, we implicitly mean a particular occurrence of ψ in φ. This occurrence is in our text either unimportant or obvious from the context.

We define the set of *free first-order variables* in a formula φ:

- $\mathrm{FV}_1(\mathrm{P}(X_1, \ldots, X_n)) = \{X_1, \ldots, X_n\}$,
- $\mathrm{FV}_1(\varphi_1 * \varphi_2) = \mathrm{FV}_1(\varphi_1) \cup \mathrm{FV}_1(\varphi_2)$ where $* \in \{\wedge, \vee, \to\}$,
- $\mathrm{FV}_1(\triangledown X.\varphi) = \mathrm{FV}_1(\varphi) \backslash \{X\}$ where $\triangledown \in \{\exists, \forall\}$,
- $\mathrm{FV}_1(\bot) = \emptyset$.

Other variables that occur in a formula are bound. Terms that differ only in renaming of bound variables are α-equivalent and we do not distinguish between them. To describe the binding structure of a formula we use a special bind function. Let us assume that a formula φ has no free variables (i.e. $\mathrm{FV}_1(\varphi) = \emptyset$) and let ψ be its subformula together with a variable X free in ψ. We define $\mathsf{bind}_\varphi(\psi, X)$ as the subformula of φ that binds the free occurrences of X in ψ, i.e. the least subformula φ' of φ such that, for each its proper subformula ψ'' that contains ψ as a subformula, $X \in \mathrm{FV}(\psi'')$. For instance $\mathsf{bind}_{\bot\to\exists X.\bot\to P(X)}(P(X), X) = \exists X.\bot \to P(X)$.

$$\frac{}{\Gamma, x{:}\varphi \vdash x : \varphi}\ (var)$$

$$\frac{\Gamma \vdash M_1 : \varphi_1 \quad \Gamma \vdash M_2 : \varphi_2}{\Gamma \vdash \langle M_1, M_2\rangle : \varphi_1 \wedge \varphi_2}\ (\wedge I)$$

$$\frac{\Gamma \vdash M : \varphi_1 \wedge \varphi_2}{\Gamma \vdash \pi_1 M : \varphi_1}\ (\wedge E1) \qquad \frac{\Gamma \vdash M : \varphi_1 \wedge \varphi_2}{\Gamma \vdash \pi_2 M : \varphi_2}\ (\wedge E2)$$

$$\frac{\Gamma \vdash M : \varphi_1}{\Gamma \vdash \underline{\mathrm{in}}_{1\,\varphi_1\vee\varphi_2} M : \varphi_1 \vee \varphi_2}\ (\vee I1) \qquad \frac{\Gamma \vdash M : \varphi_2}{\Gamma \vdash \underline{\mathrm{in}}_{2\,\varphi_1\vee\varphi_2} M : \varphi_1 \vee \varphi_2}\ (\vee I1)$$

$$\frac{\Gamma \vdash M : \varphi_1 \vee \varphi_2 \quad \Gamma, x_1 : \varphi_1 \vdash N_1 : \varphi \quad \Gamma, x_2 : \varphi_2 \vdash N_2 : \varphi}{\Gamma \vdash \underline{\mathrm{case}}\ M\ \underline{\mathrm{of}}\ [x_1 : \varphi_1]\, N_1,\ [x_2 : \varphi_2]\, N_2 : \varphi}\ (\vee E)$$

$$\frac{\Gamma, x{:}\varphi_1 \vdash M : \varphi_2}{\Gamma \vdash \lambda x : \varphi_1.M : \varphi_1 \to \varphi_2}\ (\to I) \qquad \frac{\Gamma \vdash M_1 : \varphi_1 \to \varphi_2 \quad \Gamma \vdash M_2 : \varphi_1}{\Gamma \vdash M_1 M_2 : \varphi_2}\ (\to E)$$

$$\frac{\Gamma \vdash M : \varphi}{\Gamma \vdash \lambda X M : \forall X.\varphi}\ (\forall I)^* \qquad \frac{\Gamma \vdash M : \forall X.\varphi}{\Gamma \vdash M Y : \varphi[X := Y]}\ (\forall E)$$

$$\frac{\Gamma \vdash M : \varphi[X := Y]}{\Gamma \vdash \underline{\mathrm{pack}}\, M, Y\ \underline{\mathrm{to}}\ \exists X.\,\varphi : \exists X.\varphi}\ (\exists I) \qquad \frac{\Gamma \vdash M_1 : \exists X.\varphi \quad \Gamma, x{:}\varphi \vdash M_2 : \psi}{\Gamma \vdash \underline{\mathrm{let}}\ x{:}\varphi\ \underline{\mathrm{be}}\ M_1 : \exists X.\varphi\ \underline{\mathrm{in}}\ M_2 : \psi}\ (\exists E)^*$$

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash \perp\!\!\!\perp_\varphi M : \varphi}\ (\bot E)$$

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

Fig. 1. The rules of the intuitionistic first-order logic

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

$$\begin{aligned} M, N ::= {} & x \mid \langle M_1, M_2\rangle \mid \pi_1 M \mid \pi_2 M \mid \\ & \underline{\mathrm{in}}_{1\,\varphi_1\vee\varphi_2} M \mid \underline{\mathrm{in}}_{2\,\varphi_1\vee\varphi_2} M \mid \underline{\mathrm{case}}\, M\, \underline{\mathrm{of}}\, [x : \varphi_1]\, N_1,\ [y : \varphi_2]\, N_2 \mid \\ & \lambda x : \varphi.M \mid M_1 M_2 \mid \lambda X M \mid M X \mid \\ & \underline{\mathrm{pack}}\, M,\ Y\ \underline{\mathrm{to}}\, \exists X.\,\varphi \mid \underline{\mathrm{let}}\, x : \varphi\, \underline{\mathrm{be}}\, M_1 : \exists X.\varphi\, \underline{\mathrm{in}}\, M_2 \mid \perp\!\!\!\perp_\varphi M \end{aligned}$$

where x is a proof term variable, $\varphi, \varphi_1, \varphi_2$ 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 exposition can be found in the work of de Groote [5]. We omit it here, but give an intuitive 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 $\underline{\text{in}}_{1_{\varphi_1 \vee \varphi_2}} M$, $\underline{\text{in}}_{2_{\varphi_1 \vee \varphi_2}} M$ reinterpret the value of M as one in type $\varphi_1 \vee \varphi_2$. At the same time $\underline{\text{case}}\, M \,\underline{\text{of}}\, [x : \varphi_1]\, N_1,\, [y : \varphi_2]\, N_2$ construct offers the possibility to make case analysis of a value in an $\vee$-type. This construct is available in functional programming languages in a more general form of algebraic types. The terms $\lambda x : \varphi.M$, $M_1 M_2$ represent traditional function abstraction and application. The proof terms that represent universal quantifier manipulation make it possible to parametrise type with a particular value $\lambda X M$ and use the parametrised term for a particular case MX. At last $\underline{\text{pack}}\, M,\, Y \,\underline{\text{to}}\, \exists X.\, \varphi$ makes it possible to hide behind a variable X an actual realisation of a construction that uses another individual variable Y. The abstraction obtained in this way can be exploited using $\underline{\text{let}}\, x : \varphi \,\underline{\text{be}}\, M_1 : \exists X.\varphi \,\underline{\text{in}}\, M_2$. At last the term $\perp\!\!\!\perp_\varphi M$ corresponds to the break instruction.

The environments (Γ, Δ, etc. with possible annotations) in the proving system are finite sets of pairs $x : \psi$ that assign formulas to proof variables. We write $\Gamma \vdash M : A$ to express that the judgement is indeed derivable. The inference rules of the logic are presented in Fig. 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

- $\mathrm{FV}(x) = \{x\}$,
- $\mathrm{FV}(\langle M_1, M_2\rangle) = \mathrm{FV}(M_1 M_2) = \mathrm{FV}(M_1) \cup \mathrm{FV}(M_2)$,
- $\mathrm{FV}(\pi_1 M) = \mathrm{FV}(\pi_2 M) = \mathrm{FV}(\underline{\text{in}}_{1_{\varphi_1 \vee \varphi_2}} M) = \mathrm{FV}(\underline{\text{in}}_{2_{\varphi_1 \vee \varphi_2}} M) = \mathrm{FV}(\lambda X M) = \mathrm{FV}(MX) = \mathrm{FV}(\underline{\text{pack}}\, M,\, Y \,\underline{\text{to}}\, \exists X.\, \varphi) = \mathrm{FV}(\perp\!\!\!\perp_\varphi M) = \mathrm{FV}(M)$,
- $\mathrm{FV}(\underline{\text{case}}\, M \,\underline{\text{of}}\, [x : \varphi_1]\, N_1,\, [y : \varphi_2]\, N_2) = \mathrm{FV}(M) \cup (\mathrm{FV}(N_1) \backslash \{x\}) \cup (\mathrm{FV}(N_2) \backslash y)$,
- $\mathrm{FV}(\lambda x : \varphi.M) = \mathrm{FV}(X) \backslash \{x\}$,
- $\mathrm{FV}(\underline{\text{let}}\, x : \varphi \,\underline{\text{be}}\, M_1 : \exists X.\varphi \,\underline{\text{in}}\, M_2) = \mathrm{FV}(M_1) \cup (\mathrm{FV}(M_2) \backslash \{x\})$.

Again, the terms that differ only in names of bound proof-term variables are considered α-equivalent and are not distinguished. Note that we can use the notation $\mathrm{FV}_1(M)$ to refer to all free type variables that occur in M. This set is defined by recursion over the terms and taking all the free first-order variables that occur in formulas that are part of the terms so that for instance $\mathrm{FV}_1(\underline{\text{in}}_{1_{\varphi_1 \vee \varphi_2}} M) = \mathrm{FV}_1(\varphi_1) \cup \mathrm{FV}_1(\varphi_2) \cup \mathrm{FV}_1(M)$. At the same time there are naturally terms that bind first-order variables, $\mathrm{FV}_1(\lambda X M) = \mathrm{FV}_1(M) \backslash \{X\}$ and bring new free first-order ones, e.g. $\mathrm{FV}_1(MX) = \mathrm{FV}(M) \cup \{X\}$.

Traditionally, the (*cut*) rule is not mentioned among standard rules in Fig. 1, but as it is common in λ-calculi, it is included it in the system in the form of a β-reduction rule. This rule forms the basic computation mechanism in the

system understood as a programming language. We omit the rules due to the lack of space, but an interested reader can find them in the work of de Groote [5]. Still, we want to focus our attention to terms in normal form (i.e. terms that cannot be reduced further). Partly because the search for terms in such form is easier and partly because source code of programs contains virtually exclusively terms in this form. The following theorem states that this simplification does not make us lose any possible programs in our program synthesis approach.

Theorem 1 (Normalisation). *First-order intuitionistic logic is strongly normalisable i.e. each reduction has a finite number of steps.*

The same paper by de Groote contains also (implicitly) the following result.

Theorem 2 (Subject reduction). *First-order intuitionistic logic has the subject reduction property, i.e. if* $\Gamma \vdash M : \phi$ *and* $M \to_{\beta\cup p} N$ *then* $\Gamma \vdash N : \phi$.

This theorem speaks about $\to_{\beta\cup p}$ reduction that is the sum of β-reduction and permutation reduction (p stands for permutation), which makes it possible to extensively regulate the shape of normal terms. The resulting regular form defined below is the *long normal form*. As a consequence of the above two theorems we conclude that each provable formula has a proof in this regulated normal form.

2.2 Long Normal Forms

We restrict our attention to terms which are in *long normal form* (*lnf* in short). The idea of long normal form for our logic is best explained by the following example ([12], Sect. 5): suppose $X : r$ and $Y : r \to p \vee q$. The long normal form of YX is $\underline{\text{case}}\, YX \,\underline{\text{of}}\, [a : p]\, \lambda u.\, \underline{\text{in}}_1\, u,\ [b : q]\, \lambda v.\, \underline{\text{in}}_2\, v$.

Our definitions follow those of Urzyczyn, [12]. We classify normal forms into:

- introductions $\lambda X.N$, $\lambda x.N$, $\langle N1, N2\rangle$, $\underline{\text{in}}_1\, N$, $\underline{\text{in}}_2\, N$, $\underline{\text{pack}}\, N$, $y \,\underline{\text{to}}\, \exists X.\, \varphi$,
- proper eliminators X, PN, $\pi_i P$, $P(x)$,
- improper eliminators $\bot\!\!\!\bot_\varphi(P)$, $\underline{\text{case}}\, P \,\underline{\text{of}}\, [x : \varphi_1]\, N_1,\ [y : \varphi_2]\, N_2$, $\underline{\text{let}}\, x : \varphi \,\underline{\text{be}}\, N : \exists X.\varphi \,\underline{\text{in}}\, P$

where P is a proper eliminator and N is a normal form. The long normal forms (lnfs) are defined recursively with *quasi-long proper eliminators*:

- A quasi-long proper eliminator is a proper eliminator where all arguments are of pseudo-atom type.[1]
- A constructor $\lambda X.N$, $\langle N_1, N_2\rangle$, $\in_i N$, $\underline{\text{pack}}\, N$, $y \,\underline{\text{to}}\, \exists X.\, \varphi$, $\underline{\text{let}}\, x : \varphi \,\underline{\text{be}}\, N_1 : \exists X.\varphi \,\underline{\text{in}}\, N_2$ is a lnf when its arguments N, N_1, N_2 are lnfs.
- A case-eliminator $\underline{\text{case}}\, P \,\underline{\text{of}}\, [x : \varphi_1]\, N_1,\ [y : \varphi_2]\, N_2$ is a lnf when N_1 and N_2 are lnfs and P is a quasi-long proper eliminator. A miracle (*ex falso quodlibet*) $\bot\!\!\!\bot_\varphi(P)$ of a target type φ is a long normal form when P is a quasi-long proper eliminator of type $\bot$.

[1] Note that a variable is a quasi-long proper eliminator because all arguments is an empty set in this case.

– An eliminator $\underline{\text{let}}\, x : \varphi \,\underline{\text{be}}\, N : \exists X.\varphi \,\underline{\text{in}}\, P$ is an lnf when N is an lnf and P is a quasi-long proper eliminator.

The usefulness of these forms results from the following proposition, [12].

Proposition 1 (Long normal forms). *If $\Gamma \vdash M : \phi$ then there is a long normal form N such that $\Gamma \vdash N : \phi$.*

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 the case of the propositional intuitionistic logic—can be restricted to formulas that occur only as subformulas in the initial formula. 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. *Consider a derivation of $\vdash M : \varphi$ such that M is in the long normal form. Each judgement $\Gamma \vdash N : \psi$ that occurs in this derivation has the property that, for each formula ξ in Γ and for ψ, there is a subformula ξ' of φ such that $\xi = \xi'[X_1 := Y_1, \ldots, X_n := Y_n]$ where $\mathrm{FV}(\xi') = \{X_1, \ldots, X_n\}$ and $Y_1, \ldots, Y_n$ are some first-order variables.*

Proof. Induction over the size of the term N. The details are left to the reader. □

We can generalise the property expressed in the proposition above and say that a formula ψ *emerged* from φ when there is a subformula ψ_0 of φ and a substitution $[X_1 := Y_1, \ldots, X_n := Y_n]$ with $\mathrm{FV}_1(\psi_0) = \{X_1, \ldots, X_n\}$ such that $\psi = \psi_0[X_1 := Y_1, \ldots, X_n := Y_n]$. We say that a context Γ emerged from φ when, for each its element $x : \psi$, the formula ψ emerged from φ.

3 Arcadian Automata

Our *Arcadian automaton*[2] $\mathbb{A}$ is defined as a tuple $\langle \mathcal{A}, Q, q^0, \varphi^0, \mathcal{I}, i, \mathrm{fv} \rangle$, where

– $\mathcal{A} = \langle A, \leq \rangle$ is a finite tree, which formally describes a division of the automaton control into intercommunicating modules; the root of the tree is written ε; since the tree is finite we have the relation ρ succ ρ' when $\rho \leq \rho'$ and there is no $\rho'' \neq \rho$ and $\rho'' \neq \rho'$ such that $\rho \leq \rho'' \leq \rho'$;
– 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;
– $\mathcal{I}$ is a set of all instructions;

[2] The name Arcadian automata stems from the fact that a slightly different and weaker notion of *Eden automata* was developed before [10] to deal with the fragment of the first-order intuitionistic logic with $\forall$ and $\rightarrow$ and in which the universal quantifier occurs only on positive positions.

- $i\colon Q \to \mathcal{P}(\mathcal{I})$ is a function which gives the set of instructions *available* in a given state; the function i must be such that every instruction belongs to exactly one state;
- $\mathrm{fv} : A \to P(A)$ is a function that describes the binding, it has the property that for each node v of A it holds that $\mathrm{fv}(v) = \bigcup_{w \in B} \mathrm{fv}(w)$ where $B = \{w \mid v \text{ succ } w\}$.

Each state may be either existential or universal and belongs to an element $a \in A$, so $Q = Q^{\exists} \cup Q^{\forall}$, and $Q^{\forall} = \bigcup_{a \in A} Q^{\forall}_a$ and $Q^{\exists} = \bigcup_{a \in A} Q^{\exists}_a$. The set of states Q is divided into two disjoint sets $Q_\forall$ and $Q_\exists$ of, respectively, universal and existential states.

Operational semantics of the automaton. An *instantaneous description* (ID) of $\mathbb{A}$ is a tuple $\langle q, \kappa, w, w', S, V\rangle$ where

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

Predicate logic is defined in two flavours. In one of them empty structure carriers are allowed in the other one, forbidden. We choose as the initial ID the tuple $\langle q^0, \varphi^0, \emptyset, \emptyset, \emptyset, \emptyset\rangle$. This choice is correct for the version of logic with empty carriers allowed.

Intuitively speaking the automaton works as a device which discovers the knowledge accumulated in the tree $\mathcal{A}$. 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" indicated by κ. We can imagine the automaton as a device that tries to check if a particular piece of information encoded in the tree $\mathcal{A}$ is correct. In this view the piece of information which is being checked for correctness at a given point is represented by the current node κ combined with current interpretation of bindings w. The interpretation of bindings w' is used to temporarily hold an interpretation of some bindings.

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 $\langle q, \kappa, w, w', S, V\rangle$. The operation of the instructions is defined as follows, where we assume $q' \in Q$, $\rho, \rho' \in A$.

1. $q : \mathsf{store}\rho, \rho' q'$ turns the current ID into $\langle q', \rho', w, \emptyset, S \cup \{\langle \rho, (w' \ll w)|_{\mathrm{fv}(\rho)}\rangle\}, V\rangle$,
2. $q : \mathsf{jmp}\rho, q'$ turns the current ID into $\langle q', \rho, w'', \emptyset, S, V\rangle$, where $(w \ll w')|_{\mathrm{fv}(\kappa)} \subseteq w''$ and $\mathrm{fv}(\rho) \subseteq \mathrm{dom}(w'')$,
3. $q : \mathsf{new}\rho, q'$ turns the current ID into $\langle q', \rho, w, \emptyset, S, V \cup \{X\}\rangle$, where $X \notin V$,
4. $q : \mathsf{check}\rho, \rho', q'$ turns the current ID into $\langle q', \rho', w, \emptyset, S, V\rangle$, the instruction is applicable only when there is a pair $\langle \rho, v\rangle \in S$ such that $v(\rho) = w(\kappa)$,
5. $q : \mathsf{instL}\rho, \rho', q'$ turns the current ID into $\langle q', \rho', w, \emptyset, S \cup \{\langle \rho, w''|_{\mathrm{fv}(\rho)}\rangle\}, V \cup \{X\}\rangle$, the instruction is applicable only when there is a node $\rho'' \in A$ such that ρ'' succ ρ and where $w'' = ([\rho'' := X] \ll w) \ll w'$ and $X \notin V$,
6. $q : \mathsf{instR}\rho, q'$ turns the current ID into $\langle q', \rho, w'', \emptyset, S, V\rangle$, the instruction is applicable only when an additional condition is met that κ succ ρ and where $w'' = [\gamma := X] \ll w|_{\mathrm{fv}(\rho)}$ and $\gamma \in \mathrm{fv}(\rho) \backslash \mathrm{fv}(\kappa)$ and $X \in V$,
7. $q : \mathsf{load}\rho, q'$ turns the current ID into $\langle q', \rho, w'', v, S, V\rangle$, where $(w \ll w')|_{\mathrm{fv}(\kappa)} \subseteq w''$ and $\mathrm{fv}(\rho) \subseteq \mathrm{dom}(w'')$, and $v : A \rightharpoonup V$.

When an element of the resulting ID is underspecified in instruction semantics it should be understood that any of the IDs fulfilling the description can be the result.

These instructions abstract the basic operations associated with the process of proving in predicate logic. Observe that the content of the additional register loaded by the instruction load can be used only by the immediately following instruction as all the other instructions erase the content of the register.

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

The following notion of acceptance is defined inductively. We say that the automaton $\mathbb{A}$ *eventually accepts* from an ID $a = \langle q, \kappa, w, w', S, V\rangle$ when

1. q is universal and there are no instructions available in state q (i.e. $i(q) = \emptyset$, such states are called *accepting states*), or
2. q is universal and, for each instruction i available in q, the automaton started in an ID a' eventually accepts, where a' is obtained from a by executing i,
3. if q is existential and, for some instruction i available in state q the automaton started in an ID a' eventually accepts, where a' is obtained from a by executing i.

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 configurations obtained by executing available instructions. Actually, we can view the process described above not only as a process of reaching acceptance, but also as a process of accepting the ID tree. In this light the automaton is eventually accepting from an initial configuration 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*.

Here is a basic monotonicity property of Arcadian automata.

Structural decomposition instructions

(1) $\varphi_1 \to \varphi_2$	$q^{\forall}_{\varphi_1\to\varphi_2} : \mathsf{store}\ \varphi_1, \varphi_2, q^{\exists}_{\varphi_2}$ $\Rightarrow \langle q^{\forall}_{\varphi_1\to\varphi_2}, \varphi_1 \to \varphi_2, w, \emptyset, S, V\rangle \to \langle q^{\exists}_{\varphi_2}, \varphi_2, w, \emptyset, S \cup \{\langle \varphi_1, w|_{\mathrm{fv}(\varphi_1)}\rangle\}, V\rangle$
(2) $\varphi_1 \wedge \varphi_2$	$q^{\forall}_{\varphi_1\wedge\varphi_2} : \mathsf{jmp}\ \varphi_1, q^{\exists}_{\varphi_1}$ $\Rightarrow \langle q^{\forall}_{\varphi_1\wedge\varphi_2}, \varphi_1 \wedge \varphi_2, w, \emptyset, S, V\rangle \to \langle q^{\exists}_{\varphi_1}, \varphi_1, w, \emptyset, S, V\rangle$ $q^{\forall}_{\varphi_1\wedge\varphi_2} : \mathsf{jmp}\ \varphi_2, q^{\exists}_{\varphi_2}$ $\Rightarrow \langle q^{\forall}_{\varphi_1\wedge\varphi_2}, \varphi_1 \wedge \varphi_2, w, \emptyset, S, V\rangle \to \langle q^{\exists}_{\varphi_2}, \varphi_2, w, \emptyset, S, V\rangle$
(3) $\varphi_1 \vee \varphi_2$	$q^{\exists}_{\varphi_1\vee\varphi_2} : \mathsf{jmp}\ \varphi_1, q^{\exists}_{\varphi_1}$ $\Rightarrow \langle q^{\exists}_{\varphi_1\vee\varphi_2}, \varphi_1 \vee \varphi_2, w, \emptyset, S, V\rangle \to \langle q^{\exists}_{\varphi_1}, \varphi_1, w, \emptyset, S, V\rangle$ $q^{\exists}_{\varphi_1\vee\varphi_2} : \mathsf{jmp}\ \varphi_2, q^{\exists}_{\varphi_2}$ $\Rightarrow \langle q^{\forall}_{\varphi_1\vee\varphi_2}, \varphi_1 \vee \varphi_2, w, \emptyset, S, V\rangle \to \langle q^{\exists}_{\varphi_2}, \varphi_2, w, \emptyset, S, V\rangle$
(4) $\forall X.\varphi$	$q^{\forall}_{\forall X.\varphi} : \mathsf{new}\ \varphi, q^{\exists}_{\varphi}$ $\Rightarrow \langle q^{\forall}_{\forall X.\varphi}, \forall X.\varphi, w, \emptyset, S, V\rangle \to \langle q^{\exists}_{\varphi}, \varphi, [\forall X.\varphi := Y] \ll w, \emptyset, S, V \cup \{Y\}\rangle$ where $Y \notin V$
(5) $\exists X.\varphi$	$q^{\forall}_{\exists X.\varphi} : \mathsf{instR}\ \varphi, q^{\exists}_{\varphi}$ $\Rightarrow \langle q^{\forall}_{\exists X.\varphi}, \exists X.\varphi, w, \emptyset, S, V\rangle \to \langle q^{\exists}_{\varphi}, \varphi, [\exists X.\varphi := Y] \ll w|_{\mathrm{fv}(\exists X.\varphi)}, \emptyset, S, V\rangle$ where $Y \in V$

Fig. 2. Structural decomposition instructions of the automaton

Proposition 3. *If the automaton* $\mathbb{A}$ *eventually accepts from* $\langle q, \kappa, w, w', S, V\rangle$ *and* $w \subseteq \hat{w}$ *then the automaton* $\mathbb{A}$ *eventually accepts from* $\langle q, \kappa, \hat{w}, w', S, V\rangle$.

Proof. Induction over the definition of the configuration from which automaton eventually accepts. The details are left to the reader. □

3.1 From Formulas to Automata

We can now define an Arcadian automaton $\mathbb{A}_\varphi = \langle \mathcal{A}, Q, q^{\exists}_{\varphi}, \varphi, \mathcal{I}, i, \mathrm{fv}\rangle$ that corresponds to provability of the formula φ. For technical reasons we assume that the formula is closed. This restriction is not essential since the provability of a formula with free variables is equivalent to the provability of its universal closure. The components of the automaton are as follows.

- $\mathcal{A} = \langle A, \leq\rangle$ is the syntax tree of the formula φ.
- $Q = \{q^{\forall}_{\psi}, q^{\exists}_{\psi}, q^{\forall}_{\psi,\vee}, q^{\forall}_{\psi,\to}, q^{\forall}_{\psi,\exists}, q^{\forall}_{\psi,\bot} \mid$ for all subformulas ψ of $\varphi\}$. The states annotated with the superscript $\forall$ belong to $Q^{\forall}$ while the states with the superscript $\exists$ belong to $Q^{\exists}$.
- $q^{\exists}_{\varphi}$ is the initial state (which reflects that the goal of the proving process is φ).
- The initial state and initial tree node are $q^{\exists}_{\varphi}$ and φ, respectively.
- $\mathcal{I}$ and i are presented in Figs. 2 and 3. We describe them in more detail below.
- $\mathrm{fv} : A \to P(A)$ is defined so that $\mathrm{fv}(\psi) = \{\mathsf{bind}_\varphi(\psi, X) \mid X \in \mathrm{FV}(\psi)\}$.

Non-structural instructions

(6) $q^{\exists}_{\varphi} : \mathsf{jmp}\ \varphi, q^{\forall}_{\varphi}$
$\Rightarrow\ \langle q^{\exists}_{\varphi}, \varphi, w, \emptyset, S, V\rangle \to \langle q^{\forall}_{\varphi}, \varphi, w, \emptyset, S, V\rangle$

(7) $q^{\exists}_{\varphi_i} : \mathsf{jmp}\ \varphi_1 \wedge \varphi_2, q^{\exists}_{\varphi_1\wedge\varphi_2}$ for $i = 1, 2$
$\Rightarrow\ \langle q^{\exists}_{\varphi}, \varphi, w, \emptyset, S, V\rangle \to \langle q^{\exists}_{\varphi_1\wedge\varphi_2}, \varphi_1 \wedge \varphi_2, w'', \emptyset, S, V\rangle$

(8) $q^{\exists}_{\varphi} : \mathsf{load}\ \varphi, q^{\forall}_{\varphi,\vee}$
$\Rightarrow\ \langle q^{\exists}_{\varphi}, \varphi, w, \emptyset, S, V\rangle \to \langle q^{\forall}_{\varphi,\vee}, \varphi, w, w', S, V\rangle$

(9) $q^{\exists}_{\varphi} : \mathsf{jmp}\ \varphi, q^{\forall}_{\varphi,\to}$
$\Rightarrow\ \langle q^{\exists}_{\varphi}, \varphi, w, \emptyset, S, V\rangle \to \langle q^{\forall}_{\varphi,\to}, \varphi, \hat{w}, \emptyset, S, V\rangle$ where $w \subseteq \hat{w}$

(10) $q^{\exists}_{\varphi} : \mathsf{jmp}\ \forall X.\varphi, q^{\exists}_{\forall X.\varphi}$
$\Rightarrow\ \langle q^{\exists}_{\varphi}, \varphi, w, \emptyset, S, V\rangle \to \langle q^{\exists}_{\forall X.\varphi}, \forall X.\varphi, w, \emptyset, S, V\rangle$

(11) $q^{\exists}_{\varphi} : \mathsf{load}\ \varphi, q^{\forall}_{\varphi,\exists}$
$\Rightarrow\ \langle q^{\exists}_{\varphi}, \varphi, w, \emptyset, S, V\rangle \to \langle q^{\forall}_{\varphi,\exists}, \varphi, w, w', S, V\rangle$

(12) $q^{\exists}_{\varphi} : \mathsf{jmp}\ \varphi, q^{\forall}_{\varphi,\bot}$
$\Rightarrow\ \langle q^{\exists}_{\varphi}, \varphi, w, \emptyset, S, V\rangle \to \langle q^{\forall}_{\varphi,\bot}, \varphi, w, \emptyset, S, V\rangle$

---

(13) $q^{\exists}_{\varphi} : \mathsf{check}\ \varphi, \varphi, q^{\forall}_{\mathrm{axiom}}$
$\Rightarrow\ \langle q^{\exists}_{\varphi}, \varphi, w, \emptyset, S, V\rangle \to \langle q^{\forall}_{\mathrm{axiom}}, \varphi, w, \emptyset, S, V\rangle$

---

(14) $q^{\forall}_{\varphi,\vee} : \mathsf{jmp}\ \psi_1 \vee \psi_2, q^{\exists}_{\psi_1\vee\psi_2}$
$\Rightarrow\ \langle q^{\forall}_{\varphi,\vee}, \varphi, w, w', S, V\rangle \to \langle q^{\exists}_{\psi_1\vee\psi_2}, \psi_1 \vee \psi_2, w', \emptyset, S, V\rangle$

(15) $q^{\forall}_{\varphi,\vee} : \mathsf{store}\ \psi_1, \varphi, q^{\exists}_{\varphi}$
$\Rightarrow\ \langle q^{\forall}_{\varphi,\vee}, \varphi, w, w', S, V\rangle \to \langle q^{\exists}_{\varphi}, \varphi, w', \emptyset, S', V\rangle$
where $S' = S \cup \{\langle \psi_1, w'|_{\mathrm{fv}(\psi_1)}\rangle\}$

(16) $q^{\forall}_{\varphi,\vee} : \mathsf{store}\ \psi_2, \varphi, q^{\exists}_{\varphi}$
$\Rightarrow\ \langle q^{\forall}_{\varphi,\vee}, \varphi, w, w', S, V\rangle \to \langle q^{\exists}_{\varphi}, \varphi, w', \emptyset, S', V\rangle$
where $S' = S \cup \{\langle \psi_2, w'|_{\mathrm{fv}(\psi_2)}\rangle\}$

(15) and (16) should be instantiated with ψ_1 and ψ_2's which were used in (14).

---

(17) $q^{\forall}_{\varphi,\to} : \mathsf{jmp}\ \psi \to \varphi, q^{\exists}_{\psi\to\varphi}$
$\Rightarrow\ \langle q^{\forall}_{\varphi,\to}, \varphi, w, \emptyset, S, V\rangle \to \langle q^{\exists}_{\psi\to\varphi}, \psi \to \varphi, w, \emptyset, S, V\rangle$

(18) $q^{\forall}_{\varphi,\to} : \mathsf{jmp}\ \psi, q^{\exists}_{\psi}$
$\Rightarrow\ \langle q^{\forall}_{\varphi,\to}, \varphi, w, \emptyset, S, V\rangle \to \langle q^{\exists}_{\psi}, \psi, w, \emptyset, S, V\rangle$

(18) should be instantiated with ψ and φ's which were used in (17).

---

(19) $q^{\forall}_{\varphi,\exists} : \mathsf{jmp}\ \exists X.\psi, q^{\exists}_{\exists X.\psi}$
$\Rightarrow\ \langle q^{\forall}_{\varphi,\exists}, \varphi, w, w', S, V\rangle \to \langle q^{\exists}_{\exists X.\psi}, \exists X.\psi, w', \emptyset, S', V\rangle$

(20) $q^{\forall}_{\varphi,\exists} : \mathsf{instL}\ \psi, \varphi, q^{\exists}_{\varphi}$
$\Rightarrow\ \langle q^{\forall}_{\varphi,\exists}, \varphi, w, w', S, V\rangle \to \langle q^{\exists}_{\varphi}, \varphi, w'', \emptyset, S', V\rangle$
where $w'' = ([\exists X.\psi := X] \ll w') \ll w, S' = S \cup \{\langle \psi, w''|_{\mathrm{fv}(\psi)}\rangle\}$

(19) should be instantiated with ψ and φ's which were used in (20).

---

(21) $q^{\forall}_{\varphi,\bot} : \mathsf{jmp}\ \bot, q^{\exists}_{\bot}$
$\Rightarrow\ \langle q^{\forall}_{\varphi,\bot}, \varphi, w, \emptyset, S, V\rangle \to \langle q^{\exists}_{\bot}, \bot, w, \emptyset, S, V\rangle$

Fig. 3. Non-structural instructions of the automaton

Figures 2 and 3 present the patterns of possible instructions in $\mathcal{I}$. Each of the instruction patterns starts with a state of the form q^{∇}_{ψ} or of the form $q^{\nabla}_{\psi,\bullet}$ where ∇ is a quantifier ($\forall$ or $\exists$), ψ is a subformula of φ and $\bullet$ is one of the symbols $\vee, \rightarrow, \bot, \exists$. For each of the patterns we assume $\mathcal{I}$ contains all the instructions that result from instantiating the pattern with all possible subformulas that match the form of ψ (e.g. in case $\psi = \psi_1 \rightarrow \psi_2$ we take all the subformulas with $\rightarrow$ as the main symbol). The function $i : Q \rightarrow P(\mathcal{I})$ is defined so that for a state q^{∇}_{ψ} it returns all the instructions which start from the state. In addition to the instructions they present the way a configuration is transformed by each of the instructions. This serves to facilitate understanding of the proofs.

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

Example. Consider the formula $\varphi_{\text{pos}} = \forall x(P(x)) \rightarrow \forall y \exists x P(x)$. In order to build the Arcadian automaton for that formula first we have to build the tree A of it, which is shown in Fig. 4.

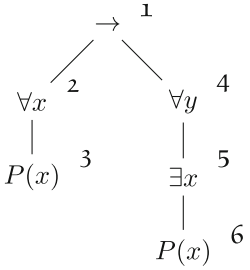

Fig. 4. Syntax tree of the formula ϕ_{pos}.

The instructions available ($\mathcal{I}$) are:

(1) $q^{\forall}_{1}$: $\mathsf{store}_2, 4, q^{\exists}_{4}$ (19) $q^{\forall}_{1,\exists}$: $\mathsf{jmp}_5, q^{\exists}_{5}$
(4) $q^{\forall}_{2}$: $\mathsf{new}_3, q^{\exists}_{3}$ (19) $q^{\forall}_{4,\exists}$: $\mathsf{jmp}_5, q^{\exists}_{5}$
(4) $q^{\forall}_{4}$: $\mathsf{new}_5, q^{\exists}_{5}$ (19) $q^{\forall}_{5,\exists}$: $\mathsf{jmp}_5, q^{\exists}_{5}$
(5) $q^{\forall}_{5}$: $\mathsf{instR}_6, q^{\exists}_{6}$ (20) $q^{\forall}_{1,\exists}$: $\mathsf{instL}_5, 1, q^{\exists}_{1}$
(10) $q^{\exists}_{3}$: $\mathsf{jmp}_2, q^{\exists}_{2}$ (20) $q^{\forall}_{4,\exists}$: $\mathsf{instL}_5, 4, q^{\exists}_{4}$
(10) $q^{\exists}_{6}$: $\mathsf{jmp}_2, q^{\exists}_{2}$ (20) $q^{\forall}_{5,\exists}$: $\mathsf{instL}_5, 5, q^{\exists}_{5}$

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

(6) $q^{\exists}_{a}$: $\mathsf{jmp}a, q^{\forall}_{a}$ (8) $q^{\exists}_{a}$: $\mathsf{load}a, q^{\forall}_{a,\vee}$ (9) $q^{\exists}_{a}$: $\mathsf{jmp}a, q^{\forall}_{a,\rightarrow}$
(11) $q^{\exists}_{a}$: $\mathsf{load}a, q^{\forall}_{a,\exists}$ (12) $q^{\exists}_{a}$: $\mathsf{jmp}a, q^{\forall}_{a,\bot}$ (13) $q^{\exists}_{a}$: $\mathsf{check}a, a, q^{\forall}_{\text{axiom}}$
(21) $q^{\forall}_{a,\rightarrow}$: $\mathsf{jmp}\bot, q^{\exists}_{\bot}$

The set of states can be easily written using the definition. To calculate fv we need to calculate binds first. We have $\mathsf{bind}_1(3, x) = 2$ and $\mathsf{bind}_1(6, x) = 5$; therefore $\text{fv}(3) = \{2\}$, $\text{fv}(6) = \{5\}$ and $\text{fv}(x) = \emptyset$ for $x \neq 3, x \neq 6$. We let $q^0 = q^{\exists}_{1}$ and $\varphi^0 = \varphi_{\text{pos}}$. The initial ID is $q = q^{\exists}_{1}$, $\kappa = 1$, and the other elements of the description are empty sets. A successful run of the automaton is as follows: $\mathsf{jmp}_1, q^{\forall}_{1}$ (rule (6), initial instruction leads to the structural decomposition of the main connective $\rightarrow$); $\mathsf{store}_2, 4, q^{\exists}_{4}$ (r. (1), as the result of the decomposition, the formula at the node 2 is moved to the context, and the formula at 4 becomes the proof goal); $\mathsf{jmp}_4, q^{\forall}_{4}$ (r. (6), we progress to the structural decomposition of $\forall$); $\mathsf{new}_5, q^{\exists}_{5}$ (r. (4), we introduce fresh eigenvariable, say X_1, for the universal quantifier); $\mathsf{jmp}_5, q^{\forall}_{5}$ (r. (6), we progress to the structural decomposition of $\exists$); $\mathsf{instR}_6, q^{\exists}_{6}$ (r. (5), we produce a witness for the existential quantifier, which can

be just X_1); $\mathsf{jmp2}, q_2^\exists$ (r. (10), we progress now with the non-structural rule that handles instantiation of the universal assumption from the node 2); and now we can conclude with $\mathsf{check2}, 2, q_{\text{axiom}}$ (r. (13)) that directly leads to acceptance.

Negative example. Consider the formula $\varphi_{\text{div}} = (((\forall x.Q(x)) \to p) \to p) \to p$; its tree is presented in Fig. 5. Obviously it does not have an inhabitant. The run of the corresponding automaton is infinite, and its main loop of states begins from $q_1^\exists$. We proceed first with the instruction (rule 1) $\mathsf{store2}, 8, q_8^\exists$, then (r. 9) $\mathsf{jmp8}, q_{8,\to}^\forall$. Now one of the applications of (17) results in $\mathsf{jmp1}, q_1^\exists$ which closes the loop. Note that there is no other way to construct a run for the automaton.

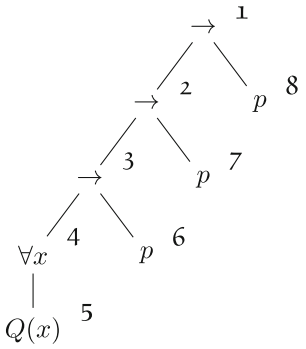

Fig. 5. Syntax tree of φ_{div}.

From derivability questions to IDs. A proof search process in the style of Ben-Yelles [1] 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 φ. We define the configuration of $\mathbb{A}_\varphi$ that corresponds to $\Gamma \vdash ? : \psi$ by exploiting the conclusion of Proposition 2. This proposition makes it possible to associate a substitution w_ψ with ψ and w_ξ with each assignment $x : \xi \in \Gamma$. The resulting configuration is $a_{\Gamma,\psi} = \langle q_{\psi_0}^\exists, \psi_0, w_\psi, \emptyset, S_{\Gamma,\psi}, V_{\Gamma,\psi}\rangle$ where $S_{\Gamma,\psi} = \{\langle \xi, \psi_\xi\rangle \mid x : \xi \in \Gamma\}$ and $V_{\Gamma,\psi} = \text{FV}_1(\Gamma, \psi)$ as well as $w_\psi(\psi_0) = \psi$.

Lemma 1. *If $\Gamma \vdash M : \psi$ is derivable and such that Γ and ψ emerged from φ then $\mathbb{A}_\varphi$ eventually accepts from the configuration $\langle q_{\psi_0}^\exists, \psi_0, w_\psi, \emptyset, S_{\Gamma,\psi}, V_{\Gamma,\psi}\rangle$, where $w_\psi(\psi_0) = \psi$ and $\text{dom}(w_\psi) = \text{fv}(\psi_0)$.*

Proof. We may assume that M is in the long normal form. The proof is by induction over the derivation of M. We give here only the most interesting cases.

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

If the last rule is the $(\wedge I)$ rule then $\psi = \psi_1 \wedge \psi_2$ and we have shorter derivations for $\Gamma \vdash M_1 : \psi_1$ and $\Gamma \vdash M_2 : \psi_2$, which by induction hypothesis give that $\mathbb{A}_\varphi$ eventually accepts from the configurations $\langle q_{\psi_{i0}}^\exists, \psi_{i0}, w_{\psi_i}, \emptyset, S_{\Gamma,\psi_i}, V_{\Gamma,\psi_i}\rangle$ 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 non-structural rule (6) to turn the existential state $q_\psi^\exists$ into the universal one $q_\psi^\forall$ for which there are two instructions available in (2), and these turn the current configuration into the corresponding above mentioned ones.

If the last rule is the $(\wedge Ei)$ rule for $i = 1, 2$ then we know that $\psi = \psi_i$ for one of $i = 1, 2$ and $\Gamma \vdash M' : \psi_1 \wedge \psi_2$ is derivable through a shorter derivation, which means by the induction hypothesis that $\mathbb{A}_\varphi$ eventually accepts from the configuration $\langle q_{\psi_1\wedge\psi_2}^\exists, \psi_1 \wedge \psi_2, w_{\psi_1\wedge\psi_2}, \emptyset, S_{\Gamma,\psi_1\wedge\psi_2}, V_{\Gamma,\psi_1\wedge\psi_2}\rangle$ where actually

$w_{\psi_1\wedge\psi_2}|_{\mathrm{fv}(\psi_i)} \subseteq w_{\psi_i}$ and $\mathrm{fv}(\psi_i) \subseteq \mathrm{dom}(w_{\psi_1\wedge\psi_2})$ for both $i = 1, 2$. Moreover, $S_{\Gamma,\psi_1\wedge\psi_2} = S_{\Gamma,\psi}$ and $V_{\Gamma,\psi_1\wedge\psi_2} = V_{\Gamma,\psi}$. This configuration can be obtained from the current one using respective non-structural instruction presented at (7).

If the last rule is the $(\to E)$ rule then we have shorter derivations for $\Gamma \vdash M_1 : \psi' \to \psi$ and $\Gamma \vdash M_2 : \psi'$. The induction hypothesis gives that $\mathbb{A}_\varphi$ eventually accepts from the configurations

$$\langle q^{\exists}_{\psi'_0\to\psi_0}, \psi'_0 \to \psi_0, w_{\psi'\to\psi}, \emptyset, S_{\Gamma,\psi'\to\psi}, V_{\Gamma,\psi'\to\psi}\rangle,$$
$$\langle q^{\exists}_{\psi'_0}, \psi'_0, w_{\psi'}, \emptyset, S_{\Gamma,\psi'}, V_{\Gamma,\psi'}\rangle.$$

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

$$\langle q^{\forall}_{\psi_0,\to}, \psi_0, w_{\psi'\to\psi}, \emptyset, S_{\Gamma,\psi}, V_{\Gamma,\psi}\rangle,$$

which can be turned into the desired two configurations 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 gives that $\mathbb{A}_\varphi$ eventually accepts from the configuration

$$\langle q^{\exists}_{\psi_{10}}, \psi_{10}, w_{\psi_1}, \emptyset, S_{\Gamma,\psi_1}, V_{\Gamma,\psi_1}\rangle,$$

where $w_{\psi_1}(\psi_{10}) = \psi_1$, $S_{\Gamma,\psi_1} = S_{\Gamma,\psi}$ and $V_{\Gamma,\psi_1} = V_{\Gamma,\psi} \cup \{X\}$.

We observe now that the non-structural instruction (6) transforms the current configuration to $\langle q^{\forall}_{\forall X.\psi_{10}}, \psi, w_\psi, \emptyset, S_{\Gamma,\psi}, V_{\Gamma,\psi}\rangle$ and then the new instruction from (4) adds appropriate element to $V_{\Gamma,\psi}$ and turns the configuration into 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 $\mathbb{A}_\varphi$ eventually accepts from configurations

$$\begin{aligned}&\langle q^{\exists}_{\exists X.\psi_{10}}, \exists X.\psi_{10}, w_{\exists X.\psi_1}, \emptyset, S_{\Gamma,\exists X.\psi_1}, V_{\Gamma,\exists X.\psi_1}\rangle,\\ &\langle q^{\exists}_{\psi_0}, \psi_0, w_\psi, \emptyset, S_{\Gamma',\psi}, V_{\Gamma',\psi}\rangle\end{aligned} \tag{I}$$

where $w_{\exists X.\psi_1}(\exists X.\psi_{10}) = \exists X.\psi_1$, $w_\psi(\psi_0) = \psi$ and $\Gamma' = \Gamma, x : \psi_1$, which consequently means that $S_{\Gamma',\psi} = S_{\Gamma,\psi} \cup \{\langle \psi_{10}, w'\rangle\}$ and $V_{\Gamma',\psi} = V_{\Gamma,\psi} \cup \{X\}$ where $w' = ([\exists X.\psi_{10} := X] \ll w_{\exists X.\psi_1})|_{\mathrm{fv}(\psi_{10})}$.

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 configuration can be transformed to

$$\langle q^{\forall}_{\psi_0,\exists}, \psi_0, w_\psi, w_{\exists X.\psi_1}, S_{\Gamma,\psi}, V_{\Gamma,\psi}\rangle \tag{II}$$

by the non-structural instruction (11). This in turn is transformed to the configurations (I) by non-structural instructions (19) and (20) respectively. Note that the correctness of the guess of $w_{\exists X.\psi_1}$ in the step to ID (II) is verified by the step to the first ID in (I) □

We need a proof in the other direction. To express the statement of the next lemma we need the notation Γ_S for a context $x_1 : w_1(\psi_1), \ldots, x_n : w_n(\psi_n)$ where $S = \{\langle \psi_1, w_1 \rangle, \ldots, \langle \psi_n, w_n \rangle\}$.

Lemma 2. *If $\mathbb{A}_\varphi$ eventually accepts from the configuration $\langle q^{\exists}_{\psi}, \psi, w, \emptyset, S, V \rangle$ 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 configuration by cases depending on the currently available instructions. Note that only instructions (3), (6), (7), (8), (9), (10), (11), (12) and (13) are available for states of the form $q^{\exists}_{\phi}$.

We can immediately see that if one of the instructions (3) from Fig. 2 is used then the induction hypothesis applied to resulting configurations brings the assumption of the respective rule $(\vee Ii)$ 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 Fig. 2 and these move control to configurations from which the induction hypothesis gives the assumptions of the introduction rules $(\rightarrow I), (\wedge I), (\forall I), (\exists I)$ respectively.

Next taking the non-structural instructions (8), (9), (11) and (12) move control to further non-structural rules in Fig. 3 and these move control to configurations from which the induction hypothesis gives the assumptions of the elimination rules $(\vee E)$, $(\rightarrow E)$, $(\exists E)$, and $(\bot E)$. At the same time the instructions (7), (10), move control directly to configurations from which the induction hypothesis gives the assumptions of the elimination rules $(\wedge E)$, $(\forall E)$.

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 1 and adapting them to the current situation. □

Theorem 3 (Main theorem). *The provability in intuitionistic first-order logic is reducible to the emptiness problem for Arcadian automata.*

Proof. Let φ be a formula of the first-order intuitionistic logic. The emptiness problem for $\mathbb{A}_\varphi$ is equivalent to checking if the initial configuration of this Arcadian automaton is eventually accepting. This in turn is by Lemmas 1 and 2 reducible to derivability of $\vdash \varphi$. □

4 Conclusions

We propose a notion of automata that can simulate search for proofs in normal form in the full first-order intuitionistic 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 type system. Although the emptiness problem for such automata is undecidable (as the logic is, [9]), the notion brings a new perspective

to the proof search process which can reveal new classes of formulae for which the proof search can be made decidable. In particular this automata, together with earlier investigations [9,10], bring to the attention that decidable procedures must constrain the growth of the subset V in IDs of automata presented here.

Our automata by design find only terms in *total discharge convention* [7], 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 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. The code and instructions needed to reproduce our results is available at http://www.mimuw.edu.pl/~maciekz/HaskellTdcStats.

References

1. Ben-Yelles, C.B.: Type-assignment in the lambda-calculus; syntax and semantics. Ph.D. thesis, Mathematics Department, University of Wales, Swansea, UK (1979)
2. Broda, S., Damas, L.: On long normal inhabitants of a type. J. Logic Comput. **15**(3), 353–390 (2005)
3. Düdder, B., Martens, M., Rehof, J.: Staged composition synthesis. In: Shao, Z. (ed.) ESOP 2014. LNCS, vol. 8410, pp. 67–86. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54833-8_5
4. Filiot, E., Talbot, J., Tison, S.: Tree automata with global constraints. Int. J. Found. Comput. Sci. **21**(4), 571–596 (2010)
5. de Groote, P.: On the strong normalisation of intuitionistic natural deduction with permutation-conversions. Inf. Comput. **178**(2), 441–464 (2002)
6. Hetzl, S.: Applying tree languages in proof theory. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 301–312. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28332-1_26
7. Prawitz, D.: Natural Deduction. Almqvist and Wiksell, Sweden (1965)
8. Schubert, A., Dekkers, W., Barendregt, H.P.: Automata theoretic account of proof search. In: Kreutzer, S. (ed.) 24th EACSL Annual Conference on Computer Science Logic (CSL 2015). LIPIcs, vol. 41, pp. 128–143. Dagstuhl (2015)
9. Schubert, A., Urzyczyn, P., Zdanowski, K.: On the mints hierarchy in first-order intuitionistic logic. In: Pitts, A. (ed.) FoSSaCS 2015. LNCS, vol. 9034, pp. 451–465. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46678-0_29
10. Schubert, A., Urzyczyn, P., Walukiewicz-Chrzaszcz, D.: Restricted positive quantification is not elementary. In: Herbelin, H., Letouzey, P., Sozeau, M. (eds.) Proceedings of TYPES 2014. LIPIcs, vol. 39, pp. 251–273. Dagstuhl (2015)
11. Takahashi, M., Akama, Y., Hirokawa, S.: Normal proofs and their grammar. Inf. Comput. **125**(2), 144–153 (1996)
12. Urzyczyn, P.: Intuitionistic games: determinacy, completeness, and normalization. Studia Logica **104**(5), 957–1001 (2016). doi:10.1007/s11225-016-9661-4