Predicate Logic with Sequence Variables and Sequence Function Symbols*

Temur Kutsia and Bruno Buchberger

Research Institute for Symbolic Computation
Johannes Kepler University Linz
A-4040 Linz, Austria
{kutsia,buchberger}@risc.uni-linz.ac.at

Abstract. We extend first-order logic with sequence variables and sequence functions. We describe syntax, semantics and inference system for the extension, define an inductive theory with sequence variables and formulate induction rules. The calculus forms a basis for the top-down systematic theory exploration paradigm.

1 Introduction

The goal of future mathematical knowledge management is the availability of significant parts of mathematical knowledge in computer-processable, verified, well-structured and semantically unambiguous form over the web and the possibility to easily expand, modify, and re-structure this knowledge according to specifications defined by the user. For this, mathematical knowledge has to be formulated in the frame of formal logics. Translation between presentations of mathematical knowledge with respect to different logics should be automatic. A natural standard for such logics is (any version of) predicate logic.

We believe that the goal can only be achieved by a systematic build-up of mathematics from scratch using systematic, flexible, algorithmic tools based on algorithmic formal logic (automated reasoning). By these tools, most of the work involved in building up well-structured and reusable mathematical knowledge should be automated or, at least, computer-assisted. We call the research field that enables this type of generation of large pieces of coherent mathematical knowledge "Computer-Supported Mathematical Theory Exploration".

The systems and projects like ALF [27], Automath [14], Coq [2], Elf [30], HOL [18], IMPS [1], Isabelle [29], Lego [26], Nuprl [13], Omega [4], Mizar [3], and others have been designed and used to formalize mathematical knowledge. Theorema [11] is one of such projects, which aims at constructing tools for computer-supported mathematical theory exploration. Since then, within the Theorema project, various approaches to bottom-up and top-down computer-supported mathematical theory exploration have been proposed and

* Supported by the Austrian Science Foundation (FWF) under Project SFB F1302, and Johann Radon Institute for Computational and Applied Mathematics, Austrian Academy of Sciences.

pursued with an emphasis on top-down methods. These approaches and first results are documented in various publications and reports (see, e.g., [8, 31, 9, 10]). The approaches are summarized in the "lazy thinking paradigm" for mathematical theory exploration introduced by the second author in [9].

In general, mathematical theory exploration requires higher-order logic. The version of predicate logic used in the case studies on theory exploration [31, 9, 10] is a higher-order predicate logic with sequence variables and sequence functions. However, the proofs in the case studies are done essentially on the first-order level. In this paper we restrict ourselves to the first-order fragment of predicate logic with sequence variables and sequence functions.

Sequence variables can be instantiated with finite sequences of terms. They add expressiveness and elegance to the language and have been used in various knowledge representation systems like KIF [16] or Common Logic [19]. Isabelle [29] implements sequent calculi using sequence variables. In programming, the language of Mathematica [32] successfully uses pattern matching that supports sequence variables and flexible arity function symbols (see [7] for more details). Sequence functions can be interpreted as multi-valued functions and have been used (under different names) in reasoning or programming systems, like, e.g., in Set-Var [5] or RelFun [6].

The following example shows how the property of a function being "orderless" can be easily defined using sequence variables: $f(\overline{x}, x, \overline{y}, y, \overline{z}) = f(\overline{x}, y, \overline{y}, x, \overline{z})$ specifies that the order of arguments in terms with the head f and any number of arguments does not matter. The letters with the overbar are sequence variables. Without them, we would need a permutation to express the same property. Definition of concatenation $\langle\overline{x}\rangle \asymp \langle\overline{y}\rangle = \langle\overline{x}, \overline{y}\rangle$ is another example of the expressiveness sequence variables.

Using sequence variables in programming helps to write elegant and short code, like, for instance, implementing bubble sort in a rule-based manner:

$$\begin{aligned} \mathsf{sort}(\langle\overline{\mathsf{x}}, \mathsf{x}, \overline{\mathsf{y}}, \mathsf{y}, \overline{\mathsf{z}}\rangle) &:= \mathsf{sort}(\langle\overline{\mathsf{x}}, \mathsf{y}, \overline{\mathsf{y}}, \mathsf{x}, \overline{\mathsf{z}}\rangle) \text{ if } \mathsf{x} > \mathsf{y} \\ \mathsf{sort}(\langle\overline{\mathsf{x}}\rangle) &:= \langle\overline{\mathsf{x}}\rangle \end{aligned}$$

Bringing sequence functions in the language naturally allows Skolemization over sequence variables: Let x, y be individual variables, $\overline{x}$ be a sequence variable, and p be a flexible arity predicate symbol. Then $\forall x \forall y \exists \overline{x}\ p(x, y, \overline{x})$ Skolemizes to $\forall x \forall y\ p(x, y, \overline{f}(x, y))$, where $\overline{f}$ is a binary Skolem sequence function symbol. Another example, $\forall \overline{y} \exists \overline{x}\ p(\overline{y}, \overline{x})$, where $\overline{y}$ is a sequence variable, after Skolemization introduces a flexible arity sequence function symbol $\overline{g}$: $\forall \overline{y}\ p(\overline{y}, \overline{g}(\overline{y}))$.

Although sequence variables and sequence functions appear in various applications, so far, to our knowledge, there was no formal treatment of full predicate logic with these constructs. (Few exceptions are [20], that considers logic with sequence variables without sequence functions, and [22], investigating equational theories, again with sequence variables, but without sequence functions.) In this paper we fill this gap, describing syntax, semantics and inference system for an extension of classical first-order logic with sequence variables and sequence functions. Although, in general, the extension is not very complicated, there are some subtle points that have to be treated carefully.

In the extended language we allow both individual and sequence variables/function symbols, where the function symbols can have fixed or flexible arity. We have also predicates of fixed or flexible arity, and can quantify over individual and sequence variables. It gives a simple and elegant language, which can be encoded as a special order-sorted first-order theory (see [25]).

A natural intuition behind sequence terms is that they represent finite sequences of individual terms. We formalize this intuition using induction, and introduce several versions of induction rules. Inductive theories with sequence variables have some interesting properties that one normally can not observe in their standard counterparts: For instance, the Herbrand universe is not an inductive domain, and induction rules can be defined without using constructors.

The calculus $\mathbf{G}^{\approx}$ that we introduce in this paper generalizes $\mathbf{LK}^{\approx}$ calculus ($\mathbf{LK}^{\approx}$ is an extension of Gentzen's $\mathbf{LK}$ calculus [17] with equality), and possesses many nice proof-theoretic properties, including the extended version of Gödel's completeness theorem. Also, the counterparts of Löwenheim-Skolem, Compactness, Model Existence theorems and Consistency lemma hold. $\mathbf{G}^{\approx}$ together with induction and cut rules forms the logical basis of the top-down theory exploration procedure [9].

The main results of this paper are the following: First, we give the first detailed description of predicate logic with sequence variables and sequence functions, clarifying the intuitive meaning and formal semantics of sequence variables that some researchers considered to be insufficiently explored (see, e.g. [12]). Second, we describe the logical foundations of the "theory exploration with lazy thinking" paradigm.

The contributions of the first author are defining syntax and semantics of languages with sequence variables and sequence functions, designing and proving the properties of the calculus $\mathbf{G}^{\approx}$, and showing relations between induction rules and intended models. The second author pointed out the importance of using sequence variables in symbolic computation (see [7]), introduced sequence variables and sequence functions in the THEOREMA system, defined various inference rules for them (including induction), and designed provers that use sequence variables.

We omit the details of proofs which can be found in the technical report [25].

2 Syntax

We consider an alphabet consisting of the following pairwise disjoint sets of symbols: individual variables $\mathcal{V}_{\mathrm{Ind}}$, sequence variables $\mathcal{V}_{\mathrm{Seq}}$, fixed arity individual function symbols $\mathcal{F}_{\mathrm{Ind}}^{\mathrm{Fix}}$, flexible arity individual function symbols $\mathcal{F}_{\mathrm{Ind}}^{\mathrm{Flex}}$, fixed arity sequence function symbols $\mathcal{F}_{\mathrm{Seq}}^{\mathrm{Fix}}$, flexible arity sequence function symbols $\mathcal{F}_{\mathrm{Seq}}^{\mathrm{Flex}}$, fixed arity predicate symbols $\mathcal{P}^{\mathrm{Fix}}$, and flexible arity predicate symbols $\mathcal{P}^{\mathrm{Flex}}$. Each set of variables is countable. Each set of function and predicate symbols is finite or countable. The binary equality predicate symbol $\approx$ is in

$\mathcal{P}^{\mathrm{Fix}}$. Besides, there are connectives $\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$, quantifiers $\exists$, $\forall$, parentheses '(', ')' and comma ',' in the alphabet.

We will use the following denotations: $\mathcal{V} := \mathcal{V}_{\mathrm{Ind}} \cup \mathcal{V}_{\mathrm{Seq}}$; $\mathcal{F}_{\mathrm{Ind}} := \mathcal{F}^{\mathrm{Fix}}_{\mathrm{Ind}} \cup \mathcal{F}^{\mathrm{Flex}}_{\mathrm{Ind}}$; $\mathcal{F}_{\mathrm{Seq}} := \mathcal{F}^{\mathrm{Fix}}_{\mathrm{Seq}} \cup \mathcal{F}^{\mathrm{Flex}}_{\mathrm{Seq}}$; $\mathcal{F}^{\mathrm{Fix}} := \mathcal{F}^{\mathrm{Fix}}_{\mathrm{Ind}} \cup \mathcal{F}^{\mathrm{Fix}}_{\mathrm{Seq}}$; $\mathcal{F}^{\mathrm{Flex}} := \mathcal{F}^{\mathrm{Flex}}_{\mathrm{Ind}} \cup \mathcal{F}^{\mathrm{Flex}}_{\mathrm{Seq}}$; $\mathcal{F} := \mathcal{F}^{\mathrm{Fix}} \cup \mathcal{F}^{\mathrm{Flex}}$; $\mathcal{P} := \mathcal{P}^{\mathrm{Fix}} \cup \mathcal{P}^{\mathrm{Flex}}$. The *arity* of $q \in \mathcal{F}^{\mathrm{Fix}} \cup \mathcal{P}^{\mathrm{Fix}}$ is denoted by $\mathcal{A}r(q)$. A function symbol $c \in \mathcal{F}^{\mathrm{Fix}}$ is called a *constant* if $\mathcal{A}r(c) = 0$.

Definition 1. *We define the notion of* term *over $\mathcal{F}$ and $\mathcal{V}$:*

1. *If $t \in \mathcal{V}_{\mathrm{Ind}}$ (resp. $t \in \mathcal{V}_{\mathrm{Seq}}$), then t is an individual (resp. sequence) term.*
2. *If $f \in \mathcal{F}^{\mathrm{Fix}}_{\mathrm{Ind}}$ (resp. $f \in \mathcal{F}^{\mathrm{Fix}}_{\mathrm{Seq}}$), $\mathcal{A}r(f) = n$, $n \geq 0$, and $t_1, \ldots, t_n$ are individual terms, then $f(t_1, \ldots, t_n)$ is an individual (resp. sequence) term.*
3. *If $f \in \mathcal{F}^{\mathrm{Flex}}_{\mathrm{Ind}}$ (resp. $f \in \mathcal{F}^{\mathrm{Flex}}_{\mathrm{Seq}}$) and $t_1, \ldots, t_n$ $(n \geq 0)$ are individual or sequence terms, then $f(t_1, \ldots, t_n)$ is an individual (resp. sequence) term.*

A term *is either an individual or a sequence term.*

We denote by $\mathcal{T}_{\mathrm{Ind}}(\mathcal{F}, \mathcal{V})$, $\mathcal{T}_{\mathrm{Seq}}(\mathcal{F}, \mathcal{V})$ and $\mathcal{T}(\mathcal{F}, \mathcal{V})$ respectively the set of all individual terms, all sequence terms and all terms over $\mathcal{F}$ and $\mathcal{V}$.

If not otherwise stated, the following symbols, with or without indices, are used as metavariables: x, y and z – over individual variables; $\overline{x}$, $\overline{y}$ and $\overline{z}$ – over sequence variables; u and v – over (individual or sequence) variables; f, g and h – over individual function symbols; $\overline{f}$, $\overline{g}$ and $\overline{h}$ – over sequence function symbols; a, b and c – over individual constants; $\overline{a}$, $\overline{b}$ and $\overline{c}$ – over sequence constants.

Example 1. Let $f \in \mathcal{F}^{\mathrm{Flex}}_{\mathrm{Ind}}$, $\overline{f} \in \mathcal{F}^{\mathrm{Flex}}_{\mathrm{Seq}}$, $g \in \mathcal{F}^{\mathrm{Fix}}_{\mathrm{Ind}}$, $\overline{g} \in \mathcal{F}^{\mathrm{Fix}}_{\mathrm{Seq}}$, $\mathcal{A}r(g) = 2$, $\mathcal{A}r(\overline{g}) = 1$.

1. $f(\overline{x}, g(x, y)) \in \mathcal{T}_{\mathrm{Ind}}(\mathcal{F}, \mathcal{V})$.
2. $\overline{f}(\overline{x}, f(x, \overline{x}, y)) \in \mathcal{T}_{\mathrm{Seq}}(\mathcal{F}, \mathcal{V})$.
3. $f(\overline{x}, \overline{g}(\overline{x})) \notin \mathcal{T}(\mathcal{F}, \mathcal{V})$, because $\overline{x}$ occurs as an argument of $\overline{g}$ which is of fixed arity.
4. $f(\overline{x}, \overline{g}(x, y)) \notin \mathcal{T}(\mathcal{F}, \mathcal{V})$, because $\overline{g}$ is unary.

Definition 2. *We define the notion of* atomic formula*, or shortly,* atom*, over $\mathcal{P}$, $\mathcal{F}$ and $\mathcal{V}$:*

1. *If $p \in \mathcal{P}^{\mathrm{Fix}}$, $\mathcal{A}r(p) = n$, $n \geq 0$, and $t_1, \ldots, t_n \in \mathcal{T}_{\mathrm{Ind}}(\mathcal{F}, \mathcal{V})$, then $p(t_1, \ldots, t_n)$ is an atom.*
2. *If $p \in \mathcal{P}^{\mathrm{Flex}}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, $n \geq 0$, then $p(t_1, \ldots, t_n)$ is an atom.*

We denote the set of atomic formulae over $\mathcal{P}$, $\mathcal{F}$ and $\mathcal{V}$ by $\mathcal{A}(\mathcal{P}, \mathcal{F}, \mathcal{V})$.

The function symbol f is called the *head* of the term $f(t_1, \ldots, t_n)$. We denote the head of $t \notin \mathcal{V}$ by $\mathcal{H}ead(t)$. The head of an atom is defined in the same way.

Definition 3. *We define the set of* formulae *$\mathcal{F}ml(\mathcal{P}, \mathcal{F}, \mathcal{V})$ over $\mathcal{P}$, $\mathcal{F}$ and $\mathcal{V}$:*

$$\begin{aligned}\mathcal{F}ml(\mathcal{P}, \mathcal{F}, \mathcal{V}) := \quad & \mathcal{A}(\mathcal{P}, \mathcal{F}, \mathcal{V}) \cup \{\neg A \mid A \in \mathcal{F}ml(\mathcal{P}, \mathcal{F}, \mathcal{V})\} \\ & \cup \{A \circ B \mid A, B \in \mathcal{F}ml(\mathcal{P}, \mathcal{F}, \mathcal{V}),\ \circ \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}\} \\ & \cup \{Qv.A \mid A \in \mathcal{F}ml(\mathcal{P}, \mathcal{F}, \mathcal{V}),\ v \in \mathcal{V},\ Q \in \{\exists, \forall\}\}.\end{aligned}$$

Free and bound variables of a formula are defined in the standard way. We denote by $\mathcal{L}_{\approx}$ the language defined by $\mathcal{F} \cup \mathcal{P}$.

3 Substitutions

Definition 4. *A* variable binding *is either a pair $x \mapsto t$ where $t \in \mathcal{T}_{\mathrm{Ind}}(\mathcal{F}, \mathcal{V})$ and $t \neq x$, or an expression $\overline{x} \mapsto \ulcorner t_1, \ldots, t_n \urcorner$*[1] *where $n \geq 0$, for all $1 \leq i \leq n$ we have $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, and if $n = 1$ then $t_1 \neq \overline{x}$.*

Definition 5. *A* substitution *is a finite set of variable bindings*

$$\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n, \overline{x}_1 \mapsto \ulcorner s^1_1, \ldots, s^1_{k_1} \urcorner, \ldots, \overline{x}_m \mapsto \ulcorner s^m_1, \ldots, s^m_{k_m} \urcorner\},$$

where $n, m \geq 0$, $x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_m$ are distinct variables.[2]

Lower case Greek letters are used to denote substitutions. The empty substitution is denoted by ε.

Definition 6. *The* instance *of a term s with respect to a substitution σ, denoted $s\sigma$, is defined recursively as follows:*

1. $x\sigma = \begin{cases} t, & \text{if } x \mapsto t \in \sigma, \\ x, & \text{otherwise.} \end{cases}$
2. $\overline{x}\sigma = \begin{cases} t_1, \ldots, t_n, & \text{if } \overline{x} \mapsto \ulcorner t_1, \ldots, t_n \urcorner \in \sigma,\ n \geq 0, \\ \overline{x}, & \text{otherwise.} \end{cases}$
3. $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$.

Example 2. $f(x, \overline{x}, \overline{y})\{x \mapsto a, \overline{x} \mapsto \ulcorner\urcorner, \overline{y} \mapsto \ulcorner a, f(\overline{x}), b \urcorner\} = f(a, a, f(\overline{x}), b)$.

By $\mathcal{V}ar(Q)$ we denote the set of all variables occurring in Q, where Q is a term or a set of terms.

Definition 7. *Let σ be a substitution.*

1. *The* domain *of σ is the set of variables $\mathcal{D}om(\sigma) = \{l \mid l\sigma \neq l\}$.*
2. *The* codomain *of σ is the set of terms $\mathcal{C}od(\sigma) = \{l\sigma \mid l \in \mathcal{D}om(\sigma)\}$.*
3. *The* range *of σ is the set of variables: $\mathcal{R}an(\sigma) = \mathcal{V}ar(\mathcal{C}od(\sigma))$.*

Note that a codomain of a substitution is a set of terms, not a set consisting of terms and sequences of terms. For instance, $\mathcal{C}od(\{x \mapsto b, \overline{x} \mapsto \ulcorner a, a, b \urcorner\}) = \{a, b\}$.

Application of a substitution on a formula is defined in the standard way. We denote an application of σ on F by $F\sigma$.

Definition 8. *A term t is* free for a variable v in a formula F *if either*

1. *F is an atom, or*
2. *$F = \neg A$ and t is free for v in A, or*

[1] To improve readability, we write sequences that bind sequence variables between $\ulcorner$ and $\urcorner$.

[2] In [24] we consider a more general notion of substitution that allows bindings for sequence function symbols as well. That, in fact, treats sequence function symbols as second-order variables. However, for our purposes the notion of substitution defined above is sufficient.

3. $F = (A \circ B)$ and t is free for v in B and C, where $\circ \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$, or
4. $F = \forall u A$ or $F = \exists u A$ and either (a) $v = u$, or (b) $v \neq u$, $u \notin \mathcal{V}ar(t)$ and t is free for v in A.

We assume that for any formula F and substitution σ, before applying σ on F all bound variables in F are renamed so that they do not occur in $\mathcal{R}an(\sigma)$. This assumption guarantees that for all $v \in \mathcal{D}om(\sigma)$ the terms in $v\sigma$ are free for v in F.

4 Semantics

For a set S, we denote by S^n the set of all n-tuples over S. In particular, $S^0 = \{\langle\rangle\}$. By S^∞ we denote the set $\cup_{i \geq 0} S^i$.

Definition 9. *A* structure $\mathfrak{S}$ *for $\mathcal{L}_\approx$ (or, in short, an $\mathcal{L}_\approx$-structure) is a pair $\langle \mathcal{D}, \mathcal{I} \rangle$, where:*

- *$\mathcal{D}$ is a non-empty set, called a* domain *of $\mathfrak{S}$, that is a disjoint union of two sets, $\mathcal{D}_{\text{Ind}}$ and $\mathcal{D}_{\text{Seq}}$, written $\mathcal{D} = \mathcal{D}_{\text{Ind}} \uplus \mathcal{D}_{\text{Seq}}$, where $\mathcal{D}_{\text{Ind}} \neq \emptyset$.*
- *$\mathcal{I}$ is a mapping, called an* interpretation *that associates:*
  - *To every individual constant c in $\mathcal{L}_\approx$ some element $c_\mathcal{I}$ of $\mathcal{D}_{\text{Ind}}$.*
  - *To every sequence constant $\overline{c}$ in $\mathcal{L}_\approx$ some element $\overline{c}_\mathcal{I}$ of $\mathcal{D}^\infty$.*
  - *To every n-ary individual function symbol f in $\mathcal{L}_\approx$, with $n > 0$, some n-ary function $f_\mathcal{I} : \mathcal{D}_{\text{Ind}}^n \to \mathcal{D}_{\text{Ind}}$.*
  - *To every n-ary sequence function symbol $\overline{f}$ in $\mathcal{L}_\approx$, with $n > 0$, some n-ary multi-valued function $\overline{f}_\mathcal{I} : \mathcal{D}_{\text{Ind}}^n \to \mathcal{D}^\infty$.*
  - *To every flexible arity individual function symbol f in $\mathcal{L}_\approx$, some flexible arity function $f_\mathcal{I} : \mathcal{D}^\infty \to \mathcal{D}_{\text{Ind}}$.*
  - *To every flexible arity sequence function symbol $\overline{f}$ in $\mathcal{L}_\approx$, some flexible arity multi-valued function $\overline{f}_\mathcal{I} : \mathcal{D}^\infty \to \mathcal{D}^\infty$.*
  - *To every n-ary predicate symbol p in $\mathcal{L}_\approx$ other than $\approx$, with $n \geq 0$, some n-ary predicate $p_\mathcal{I} \subseteq \mathcal{D}_{\text{Ind}}^n$;*
  - *To every flexible arity predicate symbol p in $\mathcal{L}_\approx$ some flexible arity predicate $p_\mathcal{I} \subseteq \mathcal{D}^\infty$;*

Definition 10. *Let $\mathfrak{S} = \langle \mathcal{D}, \mathcal{I} \rangle$ be an $\mathcal{L}_\approx$-structure. A* state σ *over $\mathfrak{S}$, denoted $\sigma^\mathfrak{S}$, is a mapping defined on variables as follows:*

- *For an individual variable x, $\sigma^\mathfrak{S}(x) \in \mathcal{D}_{\text{Ind}}$.*
- *For a sequence variable $\overline{x}$, $\sigma^\mathfrak{S}(\overline{x}) \in \mathcal{D}^\infty$.*

Definition 11. *Let $\mathfrak{S} = \langle \mathcal{D}, \mathcal{I} \rangle$ be an $\mathcal{L}_\approx$-structure and let σ be a state over $\mathfrak{S}$. A* value *of a term t in $\mathfrak{S}$ w.r.t. σ, denoted $\mathcal{V}al_\sigma^\mathfrak{S}(t)$, is defined as follows:*

- *$\mathcal{V}al_\sigma^\mathfrak{S}(v) = \sigma^\mathfrak{S}(v)$, for every $v \in \mathcal{V}$.*
- *$\mathcal{V}al_\sigma^\mathfrak{S}(f(t_1, \ldots, t_n)) = f_\mathcal{I}(\mathcal{V}al_\sigma^\mathfrak{S}(t_1), \ldots, \mathcal{V}al_\sigma^\mathfrak{S}(t_n))$, for every $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, $n \geq 0$.*

Definition 12. *Let v be a variable and $\sigma^{\mathfrak{S}}$ be a state over an $\mathcal{L}_{\approx}$-structure $\mathfrak{S}$. A state $\vartheta^{\mathfrak{S}}$ is a v-*variant *of $\sigma^{\mathfrak{S}}$ iff $\vartheta^{\mathfrak{S}}(u) = \sigma^{\mathfrak{S}}(u)$ for each variable $u \neq v$.*

The set $\mathcal{TV} = \{\mathbf{T}, \mathbf{F}\}$ is called the set of truth values. The operations $\neg_{\mathrm{TV}}$, $\vee_{\mathrm{TV}}$, $\wedge_{\mathrm{TV}}$, $\Rightarrow_{\mathrm{TV}}$, $\Leftrightarrow_{\mathrm{TV}}$ are defined on $\mathcal{TV}$ in the standard way.

Definition 13. *Let $\mathfrak{S} = \langle \mathcal{D}, \mathcal{I} \rangle$ be an $\mathcal{L}_{\approx}$-structure and σ be a state over $\mathfrak{S}$. A* truth value *of a formula F in $\mathfrak{S}$ with respect to σ, denoted $\mathcal{V}al_{\sigma}^{\mathfrak{S}}(F)$, is defined as follows:*

- $\mathcal{V}al_{\sigma}^{\mathfrak{S}}(p(t_1, \dots, t_n)) = \mathbf{T}$ *iff* $\langle \mathcal{V}al_{\sigma}^{\mathfrak{S}}(t_1), \dots, \mathcal{V}al_{\sigma}^{\mathfrak{S}}(t_n) \rangle \subseteq p_{\mathcal{I}}$.
- $\mathcal{V}al_{\sigma}^{\mathfrak{S}}(t_1 \approx t_2) = \mathbf{T}$ *iff* $\mathcal{V}al_{\sigma}^{\mathfrak{S}}(t_1) = \mathcal{V}al_{\sigma}^{\mathfrak{S}}(t_2)$.
- $\mathcal{V}al_{\sigma}^{\mathfrak{S}}(\neg F) = \neg_{\mathrm{TV}} \mathcal{V}al_{\sigma}^{\mathfrak{S}}(F)$.
- $\mathcal{V}al_{\sigma}^{\mathfrak{S}}(F_1 \circ F_2) = \mathcal{V}al_{\sigma}^{\mathfrak{S}}(F_1) \circ_{\mathrm{TV}} \mathcal{V}al_{\sigma}^{\mathfrak{S}}(F_2)$, *where* $\circ \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$
- $\mathcal{V}al_{\sigma}^{\mathfrak{S}}(\forall v F) = \mathbf{T}$ *iff* $\mathcal{V}al_{\vartheta}^{\mathfrak{S}}(F) = \mathbf{T}$ *for every* $\vartheta^{\mathfrak{S}}$ *that is a v-variant of* $\sigma^{\mathfrak{S}}$.
- $\mathcal{V}al_{\sigma}^{\mathfrak{S}}(\exists v F) = \mathbf{T}$ *iff* $\mathcal{V}al_{\vartheta}^{\mathfrak{S}}(F) = \mathbf{T}$ *for some* $\vartheta^{\mathfrak{S}}$ *that is a v-variant of* $\sigma^{\mathfrak{S}}$.

In Definition 9 we required the domain $\mathcal{D}$ to be a disjoint union of two sets $\mathcal{D}_{\mathrm{Ind}}$ and $\mathcal{D}_{\mathrm{Seq}}$. It is justified, since on the syntactic level we distinguish between individual and sequence terms, and it is natural to reflect this distinction on the semantic level. In the next section we define a calculus that is complete with respect to this semantics. Many classical results remain valid as well.

On the other side, one would naturally expect that a sequence term represents a finite sequence of individual terms. In other words, the intuition would suggest that the sequence terms should be interpreted as finite sequences of individual elements of the domain. This intuition can be easily captured by structures whose domain consists of individual elements only. We call such structures the *intended structures*. In Section 6 below we show relations between induction with sequence variables and intended structures.

5 The Gentzen-Style System $\mathbf{G}^{\approx}$

In this section we present a Gentzen-style sequent calculus for $\mathcal{L}_{\approx}$.

Definition 14. *A* sequent *is a pair of sequences of formulae. A sequent $\langle \Gamma, \Delta \rangle$ is denoted by $\Gamma \to \Delta$.*

In a sequent $\Gamma \to \Delta$, Γ is called the *antecedent* and Δ is called the *succedent.* If Γ is the empty sequence, the corresponding sequent is denoted as $\to \Delta$. If Δ is empty, the corresponding sequent is denoted as $\Gamma \to$. If both Γ and Δ are empty, we have the *inconsistent sequent* $\to$. Below the symbols Γ, Δ, Λ will be used to denote arbitrary sequences of formulae and A, B to denote formulae. Note that Γ, Δ, Λ are sequence variables on the meta level. The set of free variables of a formula A is denoted by $\mathcal{FV}ar(A)$.

Definition 15. *A* position *within a term or atom E is a sequence of positive integers, describing the path from $\mathcal{H}ead(E)$ to the head of the subterm at that position. By $E[s]_p$ we denote the term or atom obtained from E by replacing the term at position p with the term s.*

Definition 16. *The* sequent calculus $\mathbf{G}^{\approx}$ *consists of the following 17* inference rules.

$$\frac{\Gamma, A, B, \Delta \to \Lambda}{\Gamma, A \wedge B, \Delta \to \Lambda}\ (\wedge \to) \qquad \frac{\Gamma \to \Delta, A, \Lambda \quad \Gamma \to \Delta, B, \Lambda}{\Gamma \to \Delta, A \wedge B, \Lambda}\ (\to \wedge)$$

$$\frac{\Gamma, A, \Delta \to \Lambda \quad \Gamma, B, \Delta \to \Lambda}{\Gamma, A \vee B, \Delta \to \Lambda}\ (\vee \to) \qquad \frac{\Gamma \to \Delta, A, B, \Lambda}{\Gamma \to \Delta, A \vee B, \Lambda}\ (\to \vee)$$

$$\frac{\Gamma \to \Delta, A, \Lambda \quad \Gamma, B, \Lambda \to \Delta}{\Gamma, A \Rightarrow B, \Lambda \to \Delta}\ (\Rightarrow\to) \qquad \frac{\Gamma, A \to \Delta, B, \Lambda}{\Gamma \to \Delta, A \Rightarrow B, \Lambda}\ (\to\Rightarrow)$$

$$\frac{\Gamma, \Delta \to A, \Lambda}{\Gamma, \neg A, \Delta \to \Lambda}\ (\neg \to) \qquad \frac{A, \Gamma \to \Delta, \Lambda}{\Gamma \to \Delta, \neg A, \Lambda}\ (\to \neg)$$

In the quantifier rules below

1. *x is any individual variable;*
2. *y is any individual variable free for x in A and $y \notin \mathcal{FV}ar(A) \setminus \{x\}$;*
3. *t is any individual term free for x in A;*
4. *$\overline{x}$ is any sequence variable;*
5. *$\overline{y}$ is any sequence variable free for $\overline{x}$ in A and $\overline{y} \notin \mathcal{FV}ar(A) \setminus \{\overline{x}\}$;*
6. *$s_1, \ldots, s_n$, $n \geq 0$, is any sequence of terms each of them free for $\overline{x}$ in A.*

$$\frac{\Gamma, A\{x \mapsto t\}, \forall x A, \Delta \to \Lambda}{\Gamma, \forall x A, \Delta \to \Lambda}\ (\forall_{\mathrm{I}} \to) \qquad \frac{\Gamma \to \Delta, A\{x \mapsto y\}, \Lambda}{\Gamma \to \Delta, \forall x A, \Lambda}\ (\to \forall_{\mathrm{I}})$$

$$\frac{\Gamma, A\{x \mapsto y\}, \Delta \to \Lambda}{\Gamma, \exists x A, \Delta \to \Lambda}\ (\exists_{\mathrm{I}} \to) \qquad \frac{\Gamma \to \Delta, A\{x \mapsto t\}, \exists x A, \Lambda}{\Gamma \to \Delta, \exists x A, \Lambda}\ (\to \exists_{\mathrm{I}})$$

$$\frac{\Gamma, A\{\overline{x} \mapsto \ulcorner s_1, \ldots, s_n \urcorner\}, \forall \overline{x} A, \Delta \to \Lambda}{\Gamma, \forall \overline{x} A, \Delta \to \Lambda}\ (\forall_{\mathrm{S}} \to) \qquad \frac{\Gamma \to \Delta, A\{\overline{x} \mapsto \overline{y}\}, \Lambda}{\Gamma \to \Delta, \forall \overline{x} A, \Lambda}\ (\to \forall_{\mathrm{S}})$$

$$\frac{\Gamma, A\{\overline{x} \mapsto \overline{y}\}, \Delta \to \Lambda}{\Gamma, \exists \overline{x} A, \Delta \to \Lambda}\ (\exists_{\mathrm{S}} \to) \qquad \frac{\Gamma \to \Delta, A\{\overline{x} \mapsto \ulcorner s_1, \ldots, s_n \urcorner\}, \exists \overline{x} A, \Lambda}{\Gamma \to \Delta, \exists \overline{x} A, \Lambda}\ (\to \exists_{\mathrm{S}})$$

In the rule $(\approx)$ below A is an atom and p is a position.

$$\frac{\Gamma, (s \approx t \wedge A[s]_p) \Rightarrow A[t]_p \to \Delta}{\Gamma \to \Delta}\ (\approx)$$

Note that in both the $(\to \forall_{\mathrm{I}})$-rule and the $(\exists_{\mathrm{I}} \to)$-rule, the variable y does not occur free in the lower sequent. Similarly, in both the $(\to \forall_{\mathrm{S}})$-rule and the $(\exists_{\mathrm{S}} \to)$-rule, the variable $\overline{y}$ does not occur free in the lower sequent.

The *axioms* of $\mathbf{G}^{\approx}$ are all sequents $\Gamma \to \Delta$ such that Γ and Δ contain a common formula, and sequents of the form $\Gamma \to \Delta, s \approx s, \Lambda$. Validity and provability of sequents are defined in the standard way.

The following version of Gödel's extended completeness theorem holds for the calculus $\mathbf{G}^{\approx}$:

Theorem 1 (Completeness of $\mathbf{G}^{\approx}$). *A sequent (even infinite) is valid iff it is provable in* $\mathbf{G}^{\approx}$*.*

The classical results like Löwenheim-Skolem, Compactness, Model Existence theorems, and Consistency Lemma remain valid for $\mathcal{L}_{\approx}$.

The calculus $\mathbf{G}^{\approx}$ has many nice proof-theoretic properties, but is not suited for implementation because of too high non-determinism. It is a well-known problem for many sequent-based calculi (like, for instance, for the classical $\mathbf{LK}^{\approx}$ calculus). Degtyarev and Voronkov [15] survey the methods to overcome it. In our case, a variant of basic superposition with ordering and equality constraint inheritance proposed in [28] seems to be a reasonable alternative of $\mathbf{G}^{\approx}$, taking into account the fact that unification with sequence variables and sequence functions is infinitary but decidable [24]. This approach for theories with sequence variables, but without sequence functions has already been considered in [23]. To do the same for theories with sequence variables and sequence functions one needs to introduce a reduction ordering on terms involving sequence variables and functions, and an efficient algorithm for solving ordering constraints. This is a subject of further research and lies beyond the scope of this paper.

Note also that predicate logic with sequence variables and sequence function symbols can be encoded as a special order-sorted first-order theory. It requires introducing into the language an additional binary function symbol for constructing sequences, a constant for the empty sequence, and adding to the theory the corresponding axioms. Details of the translation can be found in [25].

6 Induction with Sequence Variables

In this section we develop a machinery to capture the natural intuitive meaning of sequence terms: representation of finite sequences of individual terms. On the semantic level it amounts to considering the intended structures, and on the inference rules level it leads to introducing induction.

We start with the definitions of *inductive domain* and *intended structure*.

Definition 17. *let $\mathfrak{S} = \langle \mathcal{D}, \mathcal{I} \rangle$ be a structure for the language $\mathcal{L}_{\approx}$ such that $\mathcal{D}_{\mathrm{Seq}} = \emptyset$. Then $\mathfrak{S}$ is called an* intended structure *for $\mathcal{L}_{\approx}$ and $\mathcal{D}$ is called an* inductive domain*.*

Note that Herbrand structures are *not* intended structures for $\mathcal{L}_{\approx}$. We will write $A[v]$ to indicate that the formula A contains a free occurrence of v.

Below we will use a special flexible arity function symbol f that satisfies the following axioms:

$$\begin{aligned}
\forall x \forall \overline{x} \forall \overline{y} \quad & \neg(f(\overline{x}, x, \overline{y}) \approx f()), \\
\forall x \forall y \forall \overline{x} \forall \overline{y} \quad & f(x, \overline{x}) \approx f(y, \overline{y}) \Leftrightarrow x \approx y \wedge f(\overline{x}) \approx f(\overline{y}), \\
\forall x \forall y \forall \overline{x} \forall \overline{y} \quad & f(\overline{x}, x) \approx f(\overline{y}, y) \Leftrightarrow f(\overline{x}) \approx f(\overline{y}) \wedge x \approx y, \\
\forall \overline{x} \forall \overline{y} \quad & (f(\overline{x}) \approx f(\overline{y}) \wedge A\{\overline{z} \mapsto \overline{x}\}) \Rightarrow A\{\overline{z} \mapsto \overline{y}\},
\end{aligned}$$

where A is an arbitrary formula.

Definition 18. *The well-founded*[3] *induction principle for sequence variables is formulated as follows:*

$$\forall \overline{x}\ (\forall \overline{y}\ (f(\overline{x}) \succ f(\overline{y}) \Rightarrow A\{\overline{x} \mapsto \overline{y}\}) \Rightarrow A[\overline{x}]) \Rightarrow \forall \overline{x}\ A[\overline{x}] \qquad \text{(WFI)}$$

where $\succ$ is a well-founded ordering defined for terms with the head f.

It is not hard to show that the well-founded induction principle is valid. Since well-foundedness is an undecidable property, we will develop syntactic instances of the WFI principle that avoid direct reference to arbitrary well-founded relations. We give below some practically useful examples of such instantiation that have been used in the case studies [9, 10].

We start from auxiliary notions.

Definition 19. *The* case distinction rule from the left *is the formula*

$$(A\{\overline{x} \mapsto \ulcorner\urcorner\} \wedge \forall y \forall \overline{y}\ A\{\overline{x} \mapsto \ulcorner y, \overline{y}\urcorner\}) \Rightarrow \forall \overline{x}\ A[\overline{x}] \qquad \text{(LCD)}$$

The LCD rule is not valid, as the following example shows:

Example 3. Let $A[\overline{x}]$ in LCD be the atom $p(\overline{x})$. Take a structure $\mathfrak{S} = \langle \mathcal{D}, \mathcal{I} \rangle$ such that $\mathcal{D}_{\text{Ind}} = \{a\}$, $\mathcal{D}_{\text{Seq}} = \{b\}$ and $p_{\mathcal{I}}$ contains all the finite tuples over $\mathcal{D}$ whose first element is not b. Then LCD is false in $\mathfrak{S}$.

However, the following theorem holds:

Theorem 2. LCD *is true in every intended model.*

Definition 20. *The* suffix ordering $\succ_{\text{Suf}}$ *is defined on terms with the head f:*

$$\begin{gathered}
\forall \overline{x} \forall \overline{y}\ (f(\overline{x}) \succ_{\text{Suf}} f(\overline{y}) \Leftrightarrow \\
\exists z \exists \overline{z}\ f(\overline{x}) \approx f(z, \overline{z}) \wedge (f(\overline{z}) \approx f(\overline{y}) \vee f(\overline{z}) \succ_{\text{Suf}} f(\overline{y}))) \qquad \text{(SO)}
\end{gathered}$$

Suffix ordering is well-founded and has the property that any term of the form $f(t_1, \ldots, t_n)$, $n > 0$, which is not minimal with respect to $\succ_{\text{Suf}}$, has individual terms as its first k arguments for some $1 \leq k \leq n$.

[3] Also called Nœtherian.

Definition 21. *The* structural induction from the left *is the formula*

$$(A\{\overline{x} \mapsto \ulcorner\urcorner\} \wedge \forall y \forall \overline{y}\ (A\{\overline{x} \mapsto \overline{y}\} \Rightarrow A\{\overline{x} \mapsto \ulcorner y, \overline{y}\urcorner\})) \Rightarrow \forall \overline{x}\ A[\overline{x}]. \qquad \text{(LSI)}$$

LSI can be obtained syntactically from SO and the instances of WFI and LCD: If we take

$$\begin{aligned} B[\overline{x}] &:= \forall \overline{y}\ (f(\overline{x}) \succ_{\text{Suf}} f(\overline{y}) \Rightarrow A\{\overline{x} \mapsto \overline{y}\}) \Rightarrow A[\overline{x}], \\ \text{WFI'} &:= \forall \overline{x}\ B[\overline{x}] \Rightarrow \forall \overline{x}\ A[\overline{x}], \\ \text{LCD'} &:= (B\{\overline{x} \mapsto \ulcorner\urcorner\} \wedge \forall z \forall \overline{z}\ B\{\overline{x} \mapsto \ulcorner z, \overline{z}\urcorner\}) \Rightarrow \forall \overline{x}\ B[\overline{x}], \end{aligned}$$

then the following theorem holds:

Theorem 3. *The sequent* SO, LCD', WFI' $\rightarrow$ LSI *is provable in* $\mathbf{G}^{\approx}$.

We proved this theorem with the help of one of the provers of the Theorema system. First, we proved LSI from SO, LCD', and WFI' automatically by the Theorema prover for predicate logic using metavariables, called PLM [21], and then translated the output into the $\mathbf{G}^{\approx}$ proof. (The calculus implemented in PLM is different from $\mathbf{G}^{\approx}$: it uses metavariables and has a restricted (incomplete) sequence unification algorithm for sequence metavariables.)

LSI has the following important property:

Theorem 4. LSI *is true in every intended model.*

The next theorem, in fact, shows that LCD can be proved from LSI:

Theorem 5. *The sequent* $\rightarrow$ LSI $\Rightarrow$ LCD *is provable in* $\mathbf{G}^{\approx}$.

Now we turn LSI into the inference rule:

$$\frac{\Gamma \rightarrow \Delta, A\{\overline{x} \mapsto \ulcorner\urcorner\} \wedge \forall y \forall \overline{y}\ (A\{\overline{x} \mapsto \overline{y}\} \Rightarrow A\{\overline{x} \mapsto \ulcorner y, \overline{y}\urcorner\}), \Lambda}{\Gamma \rightarrow \Delta, \forall \overline{x}\ A[\overline{x}], \Lambda} \quad (\text{SI}_{\text{left}})$$

Soundness of $\mathbf{G}^{\approx}$ and Theorem 4 imply the following result:

Theorem 6. *If a sequent is provable using* $(\text{SI}_{\text{left}})$ *and the inference rules of* $\mathbf{G}^{\approx}$*, then it is true in every intended structure.*

In the similar way we can get another instance of WFI:

Definition 22. *The* case distinction rule from the right *is the formula*

$$(A\{\overline{x} \mapsto \ulcorner\urcorner\} \wedge \forall \overline{y} \forall y\ A\{\overline{x} \mapsto \ulcorner \overline{y}, y\urcorner\}) \Rightarrow \forall \overline{x}\ A[\overline{x}] \qquad \text{(RCD)}$$

The prefix ordering $\succ_{\text{Pre}}$ *is defined on terms with the head* f *as follows:*

$$\begin{aligned} &\forall \overline{x} \forall \overline{y}\ (f(\overline{x}) \succ_{\text{Pre}} f(\overline{y}) \Leftrightarrow \\ &\exists \overline{z} \exists z\ f(\overline{x}) \approx f(\overline{z}, z) \wedge (f(\overline{z}) \approx f(\overline{y}) \vee f(\overline{z}) \succ_{\text{Pre}} f(\overline{y}))) \end{aligned} \qquad \text{(PO)}$$

The structural induction from the right *is the formula*

$$(A\{\overline{x} \mapsto \ulcorner\urcorner\} \wedge \forall \overline{y} \forall y\ (A\{\overline{x} \mapsto \overline{y}\} \Rightarrow A\{\overline{x} \mapsto \ulcorner \overline{y}, y\urcorner\})) \Rightarrow \forall \overline{x}\ A[\overline{x}]. \qquad \text{(RSI)}$$

Like LSI, we can turn RSI into an inference rule:

$$\frac{\Gamma \to \Delta, A\{\overline{x} \mapsto \ulcorner\urcorner\} \wedge \forall \overline{y} \forall y\ (A\{\overline{x} \mapsto \overline{y}\} \Rightarrow A\{\overline{x} \mapsto \ulcorner \overline{y}, y \urcorner\}), \Lambda}{\Gamma \to \Delta, \forall \overline{x}\ A[\overline{x}], \Lambda} \quad (\mathrm{SI}_{\mathrm{right}})$$

The counterpart of Theorem 6 holds for $(\mathrm{SI}_{\mathrm{right}})$.

Another useful well-founded ordering on terms with sequence variables is the *length ordering* defined as follows:

$$\begin{aligned}&\forall \overline{x} \forall \overline{y}\ (f(\overline{x}) \succ_{\mathrm{Len}} f(\overline{y}) \Leftrightarrow \exists z \exists \overline{z}\ f(\overline{x}) \approx f(z, \overline{z}) \\ &\wedge (f(\overline{y}) \approx f() \vee \exists u \exists \overline{u}\ (f(\overline{y}) \approx f(u, \overline{u}) \wedge f(\overline{z}) \succ_{\mathrm{Len}} f(\overline{u})))\end{aligned} \quad (\mathrm{LO})$$

If we instantiate the ordering $\succ$ in WFI with $\succ_{\mathrm{Len}}$, we get an instance of WFI called *well-founded induction principle with the length ordering*:

$$\forall \overline{x}\ (\forall \overline{y}\ (f(\overline{x}) \succ_{\mathrm{Len}} f(\overline{y}) \Rightarrow A\{\overline{x} \mapsto \overline{y}\}) \Rightarrow A[\overline{x}]) \Rightarrow \forall \overline{x}\ A[\overline{x}] \quad (\mathrm{WFILO})$$

The next example shows that WFILO is not valid.

Example 4. Let $A[\overline{x}]$ be an atom $p(\overline{x})$ with the flexible arity predicate symbol p. Let $\mathfrak{S} = \langle \mathcal{D}, \mathcal{I} \rangle$ be a structure whose domain $\mathcal{D}$ is the set of all ground terms built from an individual constant c, sequence constant $\overline{c}$ and a flexible arity function symbol f. The assignment $\mathcal{I}$ is defined so that it maps each ground term to itself and the predicate $\succ_{\mathrm{Len}}$ to the same predicate on $\mathcal{D}$. As for $p_{\mathcal{I}}$, let it be a flexible arity predicate on $\mathcal{D}$ that contains all the tuples over $\mathcal{D}$ except $\langle c, \overline{c} \rangle$. Then $\mathcal{V}al_{\sigma}^{\mathfrak{S}}(f(c, \overline{c}) \succ_{\mathrm{Len}} f() \Rightarrow p()) \Rightarrow p(c, \overline{c})) = \mathbf{F}$ which implies that $\mathcal{V}al_{\sigma}^{\mathfrak{S}}(\forall \overline{x}\ (\forall \overline{y}\ f(\overline{x}) \succ_{\mathrm{Len}} f(\overline{y}) \Rightarrow p(\overline{y})) \Rightarrow p(\overline{x})) = \mathbf{T}$. On the other side, $\mathcal{V}al_{\sigma}^{\mathfrak{S}}(\forall \overline{x}\ p(\overline{x})) = \mathbf{F}$. Therefore, WFILO is false in $\mathfrak{S}$ with respect to $\sigma^{\mathfrak{S}}$.

Nevertheless, WFILO is satisfied by any intended structure:

Theorem 7. WFILO *is true in any intended structure.*

Instantiating f in WFILO with the tuple constructor we get the tuple induction principle formulated in [10].

WFILO can be turned into an inference rule:

$$\frac{\Gamma, \mathrm{LO} \to \Delta, \forall \overline{x}\ (\forall \overline{y}\ (f(\overline{x}) \succ_{\mathrm{Len}} f(\overline{y}) \Rightarrow A\{\overline{x} \mapsto \overline{y}\}) \Rightarrow A[\overline{x}]), \Lambda}{\Gamma, \mathrm{LO} \to \Delta, \forall \overline{x} A[\overline{x}], \Lambda} \quad (\mathrm{WFI}_{\mathrm{len}})$$

The counterpart of Theorem 6 holds for $(\mathrm{WFI}_{\mathrm{len}})$.

The calculus $\mathbf{G}^{\approx}$ does not contain the cut rule, but we need it in induction proofs.

$$\frac{\Gamma \to \Delta, A, \Lambda \quad \Gamma, A \to \Delta, B, \Lambda}{\Gamma \to \Delta, B, \Lambda} \quad (\mathrm{Cut})$$

Cut rule forms a basis for Theorema conjecture generation algorithm and the cascade method introduced by the second author in [8]. Informally, the cascade method works as follows: Given a goal G and knowledge base K, if an attempt of proving G using K fails, cascade method tries to analyze the failing proof situation, and using the conjecture generation algorithm generates a conjecture C such that G might be provable using $K \cup \{C\}$. After that, it tries to prove C from K, in the similar manner. Thus, this procedure corresponds to the application of the cut rule, and the conjecture generation algorithm can be considered as an intelligent heuristics of selecting "useful conjectures" among infinitely many possible ones.

At the end of this section we provide an example of proving by induction with sequence variables: reverse of a reverse of a list coincides with the original list. We give a "human-readable" version of the formal proof.

Example 5. We want to prove

$$\forall \overline{x} \;\; rev(rev(\langle \overline{x} \rangle)) = \langle \overline{x} \rangle \tag{1}$$

under the assumptions

$$rev(\langle\rangle) = \langle\rangle, \tag{2}$$
$$\forall x \;\; rev(\langle x \rangle) = \langle x \rangle, \tag{3}$$
$$\forall \overline{x} \forall \overline{y} \;\; rev(\langle \overline{x}, \overline{y} \rangle) = rev(\langle \overline{y} \rangle) \asymp rev(\langle \overline{x} \rangle), \tag{4}$$
$$\forall \overline{x} \forall \overline{y} \;\; \langle \overline{x} \rangle \asymp \langle \overline{y} \rangle = \langle \overline{x}, \overline{y} \rangle, \tag{5}$$
$$\forall \overline{x} \exists \overline{y} \;\; rev(\langle \overline{x} \rangle) = \langle \overline{y} \rangle. \tag{6}$$

The formulae (2), (3) and (4) define the reverse function, (5) is a definition of concatenation, and (6) states that reversing a list gives again a list. Then the proof of (1) proceeds as follows:

Induction base:

$$\begin{aligned} rev(rev(\langle\rangle)) = &\text{ by (2)} \\ rev(\langle\rangle) = &\text{ by (2)} \\ \langle\rangle. & \end{aligned}$$

Induction hypothesis: We assume

$$rev(rev(\langle \overline{c} \rangle)) = \langle \overline{c} \rangle. \tag{7}$$

Induction step: We have to show that $rev(rev(\langle c, \overline{c} \rangle)) = \langle c, \overline{c} \rangle$:

$$\begin{aligned} rev(rev(\langle c, \overline{c} \rangle)) = &\text{ by (4)} \\ rev(rev(\langle \overline{c} \rangle) \asymp rev(\langle c \rangle)) = &\text{ by (3)} \\ rev(rev(\langle \overline{c} \rangle) \asymp \langle c \rangle) = &\text{ by (6)} \\ rev(\langle \overline{f}(\overline{c}) \rangle \asymp \langle c \rangle) = &\text{ by (5)} \\ rev(\langle \overline{f}(\overline{c}), c \rangle) = &\text{ by (4)} \\ rev(\langle c \rangle) \asymp rev(\langle \overline{f}(\overline{c}) \rangle) = &\text{ by (6)} \\ rev(\langle c \rangle) \asymp rev(rev(\langle \overline{c} \rangle)) = &\text{ by (7)} \\ rev(\langle c \rangle) \asymp \langle \overline{c} \rangle = &\text{ by (3)} \\ \langle c \rangle \asymp \langle \overline{c} \rangle = &\text{ by (5)} \\ \langle c, \overline{c} \rangle & \end{aligned}$$

which finishes the proof. Note that (6) is used once to rewrite from left to right, and in the other case from right to left.

7 Conclusion

We described a syntax, semantics and inference system for a logic with sequence variables and sequence functions. The calculus $\mathbf{G}^{\approx}$ for such a logic extends the $\mathbf{LK}^{\approx}$ calculus and has many nice proof-theoretic properties. Furthermore, we considered special structures, called intended structures, to reflect the intuition behind sequence variables: abbreviation of finite sequences of individual terms. We formalized this intuition using induction, and defined several versions of induction rules. The interesting feature of logic with sequence variables and sequence functions is that there is no need in introducing constructors to define inductive data types. This information is "built-in" into the logic itself.

References

1. An Interactive Mathematical Proof System. http://imps.mcmaster.ca/.
2. The COQ Proof Assistant. http://coq.inria.fr/.
3. The MIZAR Project. http://www.mizar.org/.
4. The OMEGA System. http://www.ags.uni-sb.de/~omega/.
5. W. W. Bledsoe and Guohui Feng. SET-VAR. *J. Automated Reasoning*, 11(3):293–314, 1993.
6. H. Boley. *A Tight, Practical Integration of Relations and Functions*, volume 1712 of *LNAI*. Springer Verlag, 1999.
7. B. Buchberger. Mathematica as a rewrite language. In T. Ida, A. Ohori, and M. Takeichi, editors, *Proc. of the 2nd Fuji Int. Workshop on Functional and Logic Programming*, pages 1–13, Shonan Village Center, Japan, 1–4 November 1996. World Scientific.
8. B. Buchberger. Theory exploration with THEOREMA. *Analele Universitatii Din Timisoara, ser. Matematica-Informatica*, XXXVIII(2):9–32, 2000.
9. B. Buchberger. Algorithm invention and verification by lazy thinking. In D. Petcu, D. Zaharie, V. Negru, and T. Jebelean, editors, *Proc. of the 5rd Int. Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'03)*, pages 2–26, Timisoara, Romania, 1–4 October 2003. Mirton.
10. B. Buchberger and A. Craciun. Algorithm synthesis by lazy thinking: Examples and implementation in THEOREMA. In *Proc. of the Mathematical Knowledge Management Symposium*, volume 93 of *ENTCS*, pages 24–59, Edunburgh, UK, 25–29 November 2003. Elsevier Science.
11. B. Buchberger, C. Dupré, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, and W. Windsteiger. The THEOREMA project: A progress report. In M. Kerber and M. Kohlhase, editors, *Proc. of Calculemus'2000 Conference*, pages 98–113, St. Andrews, UK, 6–7 August 2000.
12. E. Clarke, M. Kohlhase, J. Ouaknine, and K. Sutner. System description: ANALYTICA 2. In T. Hardin and R. Rioboo, editors, *Proc. of the 11th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus'03)*, Rome, Italy, 10–12 September. Aracne Editrice S.R.L.
13. R. Constable. *Implementing Mathematics Using the* NUPRL *Proof Development System*. Prentice-Hall, 1986.
14. N. G. de Bruijn. The mathematical language AUTOMATH, its usage, and some of its extensions. In M. Laudet, D. Lacombe, L. Nolin, and M. Schützenberger, editors, *Proc. of Symposium on Automatic Demonstration, Versailles, France*, volume 125 of *LN in Mathematics*, pages 29–61. Springer Verlag, Berlin, 1970.

15. A. Degtyarev and A. Voronkov. Equality reasoning in sequent-based calculi. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 10, pages 611–706. Elsevier Science, 2001.
16. M. R. Genesereth, Ch. Petrie, T. Hinrichs, A. Hondroulis, M. Kassoff, N. Love, and W. Mohsin. Knowledge Interchange Format, draft proposed American National Standard (dpANS). Technical Report NCITS.T2/98-004, 1998. http://logic.stanford.edu/kif/dpans.html.
17. G. Gentzen. Untersuchungen über das logische schließen. *Mathematical Zeitschrift*, 39:176–210, 1934.
18. M. Gordon and T. Melham. *Introduction to* HOL*: A Theorem Proving Environment for Higher-Order Logic.* Cambridge University Press, 1993.
19. Common Logic Working Group. Common logic: Abstract syntax and semantics. http://cl.tamu.edu/docs/cl/1.0/cl-1.0.pdf, 2003.
20. P. Hayes and C. Menzel. Semantics of knowledge interchange format. http://reliant.teknowledge.com/IJCAI01/HayesMenzel-SKIF-IJCAI2001.pdf, 2001.
21. B. Konev and T. Jebelean. Using meta-variables for natural deduction in Theorema. In M. Kerber and M. Kohlhase, editors, *Proc. of Calculemus'2000 Conference*, St. Andrews, UK, 6–7 August 2000.
22. T. Kutsia. Solving and proving in equational theories with sequence variables and flexible arity symbols. Technical Report 02-31, RISC, Johannes Kepler University, Linz, Austria, 2002.
23. T. Kutsia. Theorem proving with sequence variables and flexible arity symbols. In M. Baaz and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning. Proc. of the 9th Int. Conference, LPAR'02*, volume 2514 of *LNAI*, pages 278–291, Tbilisi, Georgia, 14–18 October 2002. Springer Verlag.
24. T. Kutsia. Solving equations involving sequence variables and sequence functions. Technical Report 04-01, RISC, Johannes Kepler University, Linz, Austria, 2004.
25. T. Kutsia and B. Buchberger. Predicate logic with sequence variables and sequence function symbols. Technical report, SFB, Linz, Austria, 2004.
26. Zh. Luo and R. Pollack. Lego proof development system: User's manual. Technical Report ECS-LFCS-92-211, University of Edinburgh, 1992.
27. L. Magnusson and B. Nordström. The ALF proof editor and its proof engine. In H. Barendregt and T. Nipkow, editors, *Types for Proofs and Programs*, volume 806 of *LNCS*, pages 213–237. Springer Verlag, 1994.
28. R. Nieuwenhuis and A. Rubio. Theorem proving with ordering and equality constrained clauses. *J. Symbolic Computation*, 19:321–351, 1995.
29. L. Paulson. Isabelle: the next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.
30. F. Pfenning. Elf: A meta-language for deductive systems. In A. Bundy, editor, *Proc. of the 12th International Conference on Automated Deduction, CADE'94*, volume 814 of *LNAI*, pages 811–815, Nancy, France, 1995. Springer Verlag.
31. W. Windsteiger. Exploring an algorithm for polynomial interpolation in the Theorema system. In T. Hardin and R. Rioboo, editors, *Proc. of the 11th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus'03)*, pages 130–136, Rome, Italy, 10–12 September 2003. Aracne Editrice S.R.L.
32. S. Wolfram. *The* Mathematica *Book.* Cambridge University Press and Wolfram Research, Inc., fourth edition, 1999.