Reports of Enlarged Sessions of the
Seminar of I. Vekua Institute
of Applied Mathematics
Volume 30, 2016

PROOF CONSTRUCTION IN UNRANKED LOGIC *

Gela Chankvetadze Lia Kurtanidze Mikheil Rukhaia

Abstract. In the paper we study proof construction methods for first-order unranked logic. Unranked languages have unranked alphabet, meaning that function and predicate symbols do not have a fixed arity. Such languages can model XML documents and operations over them, thus becoming more important in semantic web. We present a version of sequent calculus for first-order unranked logic and describe a proof construction algorithm under this calculus. We give implementation details of the algorithm. We believe that this work will be useful for the undergoing work on semantic web logic layer.

Keywords and phrases: Unranked logic, sequent calculus, proof search.

AMS subject classification (2010): 68T15, 03F03.

1 Introduction. The original conceptualization of the semantic web [2] is to make information sources available via web-like publishing mechanisms to allow computer agents (programs) to consume them in order to satisfy some high-level user goal in an autonomous fashion. In this case, the agents need to know that the information they get is reliable, accurate and trustworthy. The semantic web stack postulates that a necessary step to achieve that is to have a logic, or collection of logics, that the agent can use to reason about the knowledge it has acquired. Such logics are the Knowledge Interchange Format (KIF) [4] and Common Logic (CL) [3], which are based on unranked alphabet and using notion of sequence variables.

Unranked languages are based on unranked alphabet, where function and/or predicate symbols do not have a fixed arity. Thus, in these languages, inside a term or a formula, it is possible to have several different occurrences of the same function/predicate symbol with different numbers of arguments. Additional strength to these languages are given by notion of sequence variables and sequence functions. Sequence variables can be instantiated with finite sequences of terms, whereas sequence functions are interpreted as multi-valued functions. These constructs seem to be higher-order, but they have precise first-order semantics defined (see e.g. [3, 4, 5]).

In this paper we present a calculus **LKU**, an adapted version of the one described in [5], which is better suitable for automated proof search (we consider less logical operators in our calculus, in particular, omit rules for implication and equality). We have implemented a proof construction algorithm on Transact SQL language. The purpose, why we choose the SQL language is explained later in Section 3.

*This work was supported by the project No.: FR/51/4-102/13 of the Shota Rustaveli National Science Foundation.

$$\frac{}{a, \Gamma \vdash \Delta, a}\mathsf{ax} \qquad \frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta}\neg_{\mathsf{l}} \qquad \frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg A}\neg_{\mathsf{r}}$$

$$\frac{A, B, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta}\wedge_{\mathsf{l}} \qquad \frac{\Gamma \vdash \Delta, A \quad \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B}\wedge_{\mathsf{r}} \qquad \frac{A, \Gamma \vdash \Delta \quad B, \Gamma \vdash \Delta}{A \vee B, \Gamma \vdash \Delta}\vee_{\mathsf{l}} \qquad \frac{\Gamma \vdash \Delta, A, B}{\Gamma \vdash \Delta, A \vee B}\vee_{\mathsf{r}}$$

$$\frac{A(t), \forall x A(x), \Gamma \vdash \Delta}{\forall x A(x), \Gamma \vdash \Delta}\forall_{\mathsf{l}} \qquad \frac{\Gamma \vdash \Delta, A(u)}{\Gamma \vdash \Delta, \forall x A(x)}\forall_{\mathsf{r}}{}^{1} \qquad \frac{A(\ulcorner t_1, \ldots, t_n \urcorner), \forall \bar{x} A(\bar{x}), \Gamma \vdash \Delta}{\forall \bar{x} A(\bar{x}), \Gamma \vdash \Delta}\forall^{\mathsf{u}}_{\mathsf{l}} \qquad \frac{\Gamma \vdash \Delta, A(\bar{v})}{\Gamma \vdash \Delta, \forall \bar{x} A(\bar{x})}\forall^{\mathsf{u}}_{\mathsf{r}}{}^{2}$$

$$\frac{A(u), \Gamma \vdash \Delta}{\exists x A(x), \Gamma \vdash \Delta}\exists_{\mathsf{l}}{}^{1} \qquad \frac{\Gamma \vdash \Delta, \exists x A(x), A(t)}{\Gamma \vdash \Delta, \exists x A(x)}\exists_{\mathsf{r}} \qquad \frac{A(\bar{v}), \Gamma \vdash \Delta}{\exists \bar{x} A(\bar{x}), \Gamma \vdash \Delta}\exists^{\mathsf{u}}_{\mathsf{l}}{}^{2} \qquad \frac{\Gamma \vdash \Delta, \exists \bar{x} A(\bar{x}), A(\ulcorner t_1, \ldots, t_n \urcorner)}{\Gamma \vdash \Delta, \exists \bar{x} A(\bar{x})}\exists^{\mathsf{u}}_{\mathsf{r}}$$

[1] u is an individual variable, called an eigenvariable, not occurring in $\Gamma, \Delta, A(x)$.
[2] $\bar{v}$ is a sequence variable, called a sequence eigenvariable, not occurring in $\Gamma, \Delta, A(\bar{x})$.

Figure 1: The sequent calculus LKU.

2 Preliminaries. We follow definitions in [5]. Variable and function symbols are divided into two groups: *individual symbols* (denoted by small Latin symbols) and *sequence symbols* (denoted by Latin symbols with a bar). Both groups include a fixed and flexible arity (unranked) function symbols. There is also distinction between fixed and flexible arity predicate symbols.

The terms are built in a standard inductive way, using individual as well as sequence variables and function symbols. The only restriction is that the fixed arity function symbols can be applied only to individual terms.

The atoms are built in a standard way using predicate symbols and terms. The same restriction applies here as well: the fixed arity predicate symbols can be applied only to the individual terms. Formulas are built in a standard way from atoms, logical operators $\neg, \wedge, \vee$ and quantifiers. Quantification is allowed on both, individual as well as sequence variables.

The implication connective ($\Rightarrow$) is omitted from logical operators, because the semantics of the sequent sign $\vdash$ is defined as implication, i.e. a sequent $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$ is interpreted as $A_1 \wedge \cdots \wedge A_n \Rightarrow B_1 \vee \cdots \vee B_m$, for any $n, m \geq 0$. If both, $n = m = 0$, then $\vdash$ represents $\bot \Rightarrow \top$, i.e. falsum.

The sequent calculus LKU is given in Figure 1, where Γ, Δ represent multisets of formulas. An LKU-proof of a sequent S is a sequence of inference rules, starting from axioms (ax rule) and ending in S.

3 Implementation. The tool[1] is a multi-layer application, implemented into two disjoint parts. A proof construction algorithm (described below) is implemented in Transact SQL language. We choose SQL for several reasons: to the best of our knowledge, no one else ever tried to implement such a procedure using the SQL language. However, there is no strong reason not to do so. The Transact SQL language is Turing complete, meaning that it has the same power as other programming languages. The SQL engine

[1]The tool is available at `http://www.logic.at/staff/mrukhaia/unranked.zip`

itself is quite fast when it needs to treat a big amount of objects (frequently happening in theorem proving).

Another reason is that, in the future we plan to use our tool to analyze data stored in relational databases. We believe that having such a procedure already available inside the database will give us better performance. Finally, SQL is platform independent in the sense that any type of interface (desktop, web, etc.) can be attached to it.

The second part of the tool is Graphical User Interface (GUI), implemented in Visual Prolog. The GUI is connected to the database, passes the sequent to be proved to it and displays the proof information from it (such as the sequence of rules applied, the unifiers used, axioms reached, etc.). The GUI has basic functionality, but it can be improved in the future.

To input sequents in the GUI (and in SQL procedure as well) we use standard syntax of LaTeX for the logical operators, i.e. `\neg, \land, \lor, \forall, \exists` and `\seq` for the sequent sign. To write down formulas we have the following rule (dictated by the way SQL treats strings): do not use parenthesis before operators, but use one after them. This restriction is not essential and will be removed in the future.

Example. The sequent $\forall x(\neg P(x) \vee Q(x)) \vdash \forall x \exists y(\neg P(x) \vee Q(y))$ is represented in our syntax as

```
\forall(x,\neg(P(x))\lor(Q(x)))
    \seq(\forall(x,\exists(y,\neg(P(x))\lor(Q(y)))))
```

When an input sequent is passed to SQL, there are three different procedures: `CHECK`, `APPLY`, and `AXIOMS` running consecutively. First, the `CHECK` procedure checks the syntax of the input, resets all the tables, reads out the structure of the input sequent and stores it into a table.

Then, the `APPLY` procedure decomposes formulas occurring in the sequent according to inference rules of the LKU calculus, until axioms are reached. The sequence of applied rules are stored in another table. Note that all propositional rules in LKU are invertible, thus they can be applied freely. The priority is given to unary inference rules, since the binary rules duplicate the context. Therefore, unary rules are applied when applicable and the application of binary rules is postponed as far as possible.

To handle quantifiers, we use the similar method described in [1]. This means that the choice for the weak quantifier[2] instance term is postponed until it is obtained via unification. It works in the following way: on a decomposition step of a *weakly quantified formula* [3], we keep its original version as well (to avoid backtracking) and replace the quantified variable with an eigenvariable, until the proper term is obtained via unification. The substitution is applied to the whole proof skeleton, to replace every occurrence of the eigenvariable with the proper term.

[2] $\forall$ on the left-hand side of the sequent and $\exists$ on the right-hand side of the sequent are considered as weak quantifiers.

[3] A formula, having a weak quantifier as an outermost connective.

Unranked unification problem, even in its simplest form, involving sequence variables, can have infinitely many unifiers. Thus it is non-terminating in general. In our tool we have implemented terminating fragment of unranked unification, similar to the one described in [6].

Finally, the `AXIOMS` procedure checks that axioms are correct (using unification) and completes the proof (by applying proper substitutions).

4 Conclusions. We presented a method to construct proofs in unranked logics and its implementation. For the future we plan to investigate unranked unification problem and improve unranked unification algorithm to cover broader class of proofs inside our tool.

REFERENCES

1. Autexier, S., Mantel, H., Stephan, W. Simultaneous quantifier elimination. *KI-98: Advances in Artificial Intelligence*, (1998), 141–152.
2. Berners-Lee, T., Hendler, J., Lassila, O. The semantic web. *Scientific American*, **May 2001**, 34–43.
3. Delugach, H. Common Logic (CL) – A Framework for a Family of Logic-Based Languages. *ISO/IEC 24707 Information Technology Standard*, 2007.
4. Genesereth, M. Knowledge Interchange Format. *Draft proposed American National Standard*, 1998. Available at `http://logic.stanford.edu/kif/dpans.html`
5. Kutsia, T., Buchberger, B. Predicate logic with sequence variables and sequence function symbols. *In Proceedings of Conference on Mathematical Knowledge Management, Lecture Notes in Computer Science*, **3119** (2004), 205–219.
6. Kutsia, T., Marin, M. Solving, reasoning and programming in Common Logic. *In Proceedings of 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, IEEE(2012), 119–126.

Received 17.05.2016; revised 30.10.2016; accepted 11.11.2016.

Author(s) address(es):

Gela Chankvetadze
I.Vekua Institute of Applied Mathematics, I. Javakhishvili Tbilisi State University
University str. 2, 0186 Tbilisi, Georgia
E-mail: gelachan@hotmail.com

Lia Kurtanidze
Faculty of Informatics, Mathematics and Natural Sciences, Georgian University
I.Chavchavadze Av. 53a, 0179 Tbilisi, Georgia
E-mail: lia.kurtanidze@gmail.com

Mikheil Rukhaia
I.Vekua Institute of Applied Mathematics, I. Javakhishvili Tbilisi State University
University str. 2, 0186 Tbilisi, Georgia
E-mail: mrukhaia@logic.at