System Description: inka 5.0 - A Logic Voyager

Serge Autexier\textsuperscript{1}, Dieter Hutter\textsuperscript{2}, Heiko Mantel\textsuperscript{2}, and Axel Schairer\textsuperscript{2}

\textsuperscript{1} Saarland University, Fachbereich Informatik, Postfach 15 11 50
D-66041 Saarbrücken, Germany
autexier@cs.uni-sb.de

\textsuperscript{2} German Research Center for Artificial Intelligence, Stuhlsatzenhausweg 3
D-66123 Saarbrücken, Germany
\{hutter,mantel,schainer\}@dfki.de

1 Introduction

Originally developed as an automatic inductive theorem prover \cite{2} based on resolution and paramodulation, the inka system was redesigned in inka 4.0 in the early '90s \cite{8} to meet the requirements arising from its designated use in formal methods. Meanwhile several large industrial applications of the verification support environment (VSE) \cite{7} have been performed which gave rise to thousands of proof obligations to be tackled by its underlying deductive system inka.

The new version of inka is a result of this long experience made in formal software development. Thus, the major improvements of inka 5.0 are concerned with the requirements arising when dealing with large applications. The user database is distributed along different deductive units each of which consists of an individual logic (consequence relation) and a set of (local) axioms. In order to allow for the logical implementation of structured specifications as they are provided in languages like CASL \cite{3}, the deductive units may import the deductive reasoning of other units with the help of morphisms. Relationships between different units are also postulated with the help of morphisms between two units which give rise to various proof obligations. inka also supports the evolutionary aspect of formal software development as it incorporates a management of change. It minimizes the proof obligations arising when changing a deductive unit or defined relationships between some units. As a basis for the implementation of different logics, inka provides an annotated \(\lambda\)-calculus as an underlying meta-language. Annotations are a generalization of the colour concept \cite{6} and are used to incorporate domain knowledge into the proof search process. inka provides a uniform hierarchical proof datastructure and a generic tactic definition mechanism to implement appropriate proof search engines.

2 System Description

\textbf{Meta-level Language.} inka supports the use of formulas of different logics and proof-objects of different calculi. All these objects are encoded in a meta-level language, based on a (ML-type) polymorphic \(\lambda\)-calculus \cite{4}. \(\lambda\)-terms are
automatically kept in $\beta\eta$ long normal-form. Using $\lambda$-calculus as meta-language provides a clear concept of variables, namely bound and free variables.

The annotations in inka's $\lambda$-calculus are described in a first-order term language and are attached to occurrences of function constants and variables, to $\lambda$-abstractions and applications. The concept of annotations is a generalization of the concept of colours \cite{7}, which has been developed in the context of inductive theorem proving to encode the knowledge about the similarities between induction hypothesis and induction conclusion \cite{9}. They are used by tactics to encode domain knowledge into logical objects and the underlying annotated calculus supports the inheritance of the annotations. For a detailed description of annotated $\lambda$-calculus see \cite{10,6,7}.

The implementation of the annotated $\lambda$-calculus also comprises a unification for annotated $\lambda$-terms with free variables \cite{10,6,7}. The unification algorithm is generic in the sense, that domain specific unification algorithms can be integrated into it. Thus, specialized unification algorithms, which make use of semantic properties of defined function constants, can be implemented and are integrated in an object-oriented manner into the generic unification algorithm.

**Hierarchy of Logics.** inka provides a mechanism for user defined logics in terms of so-called logic units (cf. Figure 1). The rôle of a logic unit is to define a truth-type and a language of logical formulas. In addition to the declarative content, a logic unit may contain specific domain-knowledge. For example, a logic unit can contain a specific theory unification algorithm for formulas; for first-order logic, a specific unification algorithm is implemented, which moves quantifiers over other connectives. Each time a logic unit is used, the specific unification is linked into the generic unification from the underlying $\lambda$-calculus. This approach differs essentially from logical frameworks like LF \cite{5} or Isabelle \cite{12} in which logics, respectively, are represented as signatures in a dependent type theory or by an embedding into a meta-logic.

In inka, a calculus is represented within a calculus unit which is related to a logic unit and defines the proof objects as well as the basic calculus rules. E.g., in the sequent calculus unit for first-order logic, the proof-objects are sequents,