SKIL: A System for Programming with Proofs*

D. Galmiche and O. Hermann
CRIN - CNRS & INRIA Lorraine
Campus Scientifique - B.P. 239
54506 Vandœuvre-lès-Nancy Cedex France.
e-mail:galmiche{ohermann}@loria.fr

1 Introduction

SKIL (*Synthesizing Knowledge in Intuitionistic Logic*) is an interactive theorem prover dedicated to program synthesis and implemented in Quintus-Prolog. The object-level logic is a second-order constructive logic denoted AF2 which includes induction through the data type definitions [3]. Since the object-level is constructive, terms of λ-calculus are constructed during the proof search. These extracted terms are programs following the *programming with proofs* paradigm, having the two properties of correctness and termination [3]. That is, the system is in fact a program synthesis environment, since a specification can be regarded as a proposition and its extracted term as a program which meets this specification.

2 Programming in AF2

In this framework, the program construction is broken down in three main steps:

1) The first step consists in expressing data types in the outlined logic. For instance, to define the type *int* of integer, we can consider this second-order formula:
$x \ in \ int \ \equiv \ \forall X \ [[\forall y \ y \ in \ X \ \rightarrow \ s(y) \ in \ X] \ \rightarrow \ [0 \ in \ X \ \rightarrow \ x \ in \ X]]$

2) Secondly, we express the specification of the program by a system of equations which semantically defines the function to be computed. For example, we can specify the *Ackermann* function with the following equations:

$$ack(0,y) = s(y)$$
$$ack(s(x),0) = ack(x,s(0))$$
$$ack(s(x),s(y)) = ack(x,ack(s(x),y))$$

3) Finally, this specification is completed by the proposition stating that the function has the intended type. The program that can be viewed as an implementation of the algorithm defined by the equations, is extracted from the proof of the proposition. For the previous example, it means proving the proposition stating that if z and z_1 are two integers, $ack(z, z_1)$ is also an integer i.e.,

* In proceedings of 4th Int. Conf. on Logic Programming and Automated Reasoning, LPAR'93, LNAI 698, pp 348-350, July 1993

$$\forall z\ (\ z\ in\ int\ \rightarrow\ (\forall z_1\ (z_1\ in\ int\ \rightarrow\ ack(z,z_1)\ in\ int))).$$

The λ-term resulting from the proof is a program which, given two integers z and z_1, computes through β-reduction the result *ack(z,z_1)*.

Some problems arise in this way, such as providing adapted construction techniques, understanding the relationship between proofs and programs, and analyzing the efficiency of the programs.

3 Motivations

Underlying this study, there are two major aims:

1) Show the suitability and the power of AF2 (in particular the *second-order quantification*), for extracting programs from constructive proofs. Indeed, AF2 naturally contains induction through the data type definitions (for instance, the previous definition of integers includes the *iteration* scheme). Moreover, the universal quantification over predicate variables provides a uniform way to perform different forms of induction.

2) Investigate the mechanization of the proof construction in order to envisage automatic program synthesis through successful strategies of proof search which yield efficient programs. It relies on good induction choices and tactics ordering.

4 Proof and Program Development

Proofs are constructed in a top-down fashion and the proof search is guided by and controlled with the help of *tactics* which perform inference rules and can be composed with *tacticals* [2].
They can be achieved using *proof plans* (sequences of tactics linked by tacticals) [1], which describe the high-level structure of some classes of proofs. At the present time, the system provides four principal proof plans corresponding to different ways to obtain proofs (namely the *constructor* proof plan for non-inductive proofs, the *function* proof plan for those functions specified as the restriction of another one, and two inductive proof plans *p-induction* and *f-induction* to guide inductive proof search). These proof plans allow us to obtain proofs and programs for various examples such as the insertion sort, the Ackermann function or the Fibonacci one. They work independently of the data types used. However, the choice of a proof plan is a key issue for the success of the proof search and even for the efficiency of the program obtained.
We can have a *meta-level generic proof plan* which includes all of the previous proof plans and turns the interactive theorem prover into an automatic one. The choice of an adequate proof plan is supplied by an heuristic procedure based on the initial program specification analysis. The goal of this procedure is to select a good induction scheme (induction object and induction rule) with two main objectives. The first is to be able to actually build a proof and therefore a program meeting

the given specification. Secondly, in case several induction schemes are applicable, it chooses the one which yields the most efficient program. Induction choices (by means of the second-order specialization rule application) are essential for the success of the proof search. Indeed, for the *predecessor* function over integers (following the previous definition), the proof search will fail with *iteration* whenever it succeeds with *recursion*.
However, since inductive proofs in the second-order intuitionistic logic are an undecidable area, the execution of the obtained proof plan is not guaranteed to succeed, but we can handle a large class of specification forms.

5 Implementation Details

The system SKIL is under implementation in Quintus-Prolog and runs on Sun4 workstations under the Unix operating system. The tactic language is Prolog, i.e., to each inference rule and to each tactic corresponds a Prolog predicate to execute it. Prolog unification and backtracking in tactics have proved useful in the automatization of proof search.
On the other hand, the integrated object-oriented programming package *Protalk* provides us with a nice way to represent, store and manipulate information within the system. In particular, it handles the tactics, proof plans, data type definitions, function specifications, as well as the proofs themselves.

6 Conclusion

SKIL is a system to experiment the mechanization of the construction of correct and efficient programs from proofs. Compared to *ProPre* [4], another implementation of AF2, we focus on the mechanization of the proof development and the induction choices. It treats directly recursive data types and automatically obtains programs with the help of their *recursors* synthesized for each data type. It includes an abstract machine executing efficiently λ-terms. Such works have potentially a broad application field in areas where deriving correct programs from specifications is becoming an absolute necessity.
A further work will lead to the extension of the existing proof plans and in particular the generic one, to deal with other induction rules (as of now, it provides the *iteration, recursion* and *course-by-value* induction rules) such as the *structural induction*.

References

1. A. Bundy. The use of explicit plans to guide inductive proofs. In *9th Conference on Automated DEduction*, pages 111–120. LNAI 310, Springer-Verlag, 1988.
2. D. Galmiche and O. Hermann. Automated proof and program development. In C.M. Rattray & R.G. Clark, editor, *The Unified Computation Laboratory*, pages 397–410. Oxford University Press, 1992.
3. J.L. Krivine and M. Parigot. Programming with proofs. *J. Inform. Process. Cybernet.*, 3:149–167, 1990.

4. P. Manoury, M. Parigot, and M. Simonot. ProPre: a programming language with proofs. In A. Voronkov, editor, *Logic Programming and Automated Reasoning*, pages 484–486. LNAI 624, 1992.