Computer Programming as Mathematics in a Programming Language and Proof System CL

Ján Komara and Paul J. Voda

Institute of Informatics, Comenius University Bratislava Slovakia.
E-mail: {komara,voda}@fmph.uniba.sk

CL (C*lausal* L*anguage*) is a computer programming language with mathematical syntax and a proof system based on Peano arithmetic which we have repeatedly used in the teaching of three (first and second year) undergraduate courses covering respectively *declarative programming*, *program verification*, and *program and abstract data specification*.

CL functions are over natural numbers, and yet CL has a look and feel of a modern functional language (higher-order functions are for the time being not covered). The coding of data structures into natural numbers is done via a pairing function which effectively identifies the domain of S-expressions of LISP with natural numbers. Recursion schemas available for the definition of CL functions are extremely programmer-friendly in that that they permit arbitrarily nested recursion where a previously defined measure of arguments goes down. By the well-known theorem of Tait on nested ordinal recursion (restricted in CL to ω) this does not lead outside of primitive recursive functions. Thus the Tait's theorem characterizes the CL programming language as being able to define exactly the unary primitive recursive functions (the effect of n-ary functions is achieved via pairing).

CL comes with its own proof system (intelligent proof checker) for proving properties of CL-defined functions such as the demonstration that they satisfy previously stated specifications. The proof system is also used for *proof obligations* where the user convinces the system that his recursively defined functions are properly introduced (they decrease arguments in certain measures).

The proof system is based on signed tableaux of Smullyan whose F-signed formulas are interpreted as goals to be proved and T-signed ones as assumptions. This permits a natural deduction style as used in mathematical practice and the proofs are easily described in English. It is amazing that CL seems to be the first system with such an obvious interpretation of signed tableaux.

The strength of the CL-proof system is characterized as a certain fragment of Peano Arithmetic. By the incompleteness result of Gödel, every formal system containing addition and multiplication admits only a fragment of recursive functions determined by its *proof* strength. Thus it seemed natural to us to choose that fragment of Peano arithmetic whose provably recursive functions are precisely the primitive recursive functions. This is the $I\Sigma_1$*-arithmetic* where induction axioms are restricted to Σ_1-formulas. Primitive recursive functions have very natural closure properties and certainly contain all feasibly computable functions.

H. de Swart (Ed.): TABLEAUX'98, LNAI 1397, pp. 42–43, 1998.

Because CL uses strong recursion schemas for the definition of functions, its proof system requires a rich variety of induction schemas for proving their properties. The induction schemas are automatically derived from CL predicates characterizing data structures (such as lists, trees, tables) and amount to the shell principles of Boyer-Moore's system. Because CL has quantifiers, the induction schemas are extremely simply given in the form of Π_2-*rules* which by the well-known theorem of Parsons are reducible to Σ_1-induction axioms.

The domain of natural numbers is so well-known that the students have no problem understanding the meaning (semantics) of functions of CL and have a good intuition about their properties. This should be contrasted with similar systems with more complex and less intuitive domains (for instance PVS which is based on typed functionals). Our experience is that the students seem not only to understand but also enjoy CL.