HORN CLAUSE COMPUTABILITY

STEN-ÅKE TÄRNLUND

Abstract.

It is proved that a Turing computable function \( f \) is computable in binary Horn clauses, which are a subset of first order logic. Moreover, it is proved that the binary Horn clauses do not need more than one function symbol. The proofs comprise computable relations that can be run efficiently as logic programs on a computer.

Key words and phrases: Turing computable functions, predicate logic, resolution logic, Horn clauses, logic programming, programming language.

CR Categories: 5.21, 5.26, 5.27.

Introduction.

We prove that a Turing computable function \( f \) is computable in binary Horn clauses. In addition, we prove that the binary Horn clauses do not need more than one function symbol. The theorems follow if we can prove that there exists a Turing machine which satisfies the restrictions of the theorems. Our specification of this machine differs in two ways from the axiomatic definition in first order logic by Turing [13] in the late thirties. Turing employed a standard form (non-clausal form) whereas we, obviously, are restricted to binary Horn clauses. Second, our clauses comprise a computable relation that can be run as a program on a computer by a theorem prover simulating a universal Turing machine (see appendix). The computable relation is trivially derived from an axiomatic definition of a universal Turing machine in standard form. In contrast, it is not trivial to derive from Turing's definition a computable relation that can be run as a program by a theorem prover.

A computable relation in Horn clauses is viewed as a program in a procedural interpretation of a Horn clause. Taking this view predicate logic becomes a programming language (Hayes [5], Kowalski [7]). Our first theorem ensures that any programming language based on the set of Horn clauses has sufficient computational power to compute any computable function.

Horn clauses used with simple control information, for example, ordering of literals for computational efficiency comprise a very high level programming language which has indeed been demonstrated by PROLOG (Roussel [12]). Some examples of PROLOG applications are: natural language question answering (Colmerauer et al. [3]), symbolic integration (Kanoui [6]), data bases (Tärnlund [15]) and compiler writing (Warren [16]). Warren's PROLOG-compiler is, in fact, written in PROLOG.

2. Well-formed formulas.

The well-formed formulas are the only meaningful expressions of predicate logic. They are defined in this section and will be used to axiomatize a universal Turing machine relationship in section 3.

An expression is a term if and only if it is a variable or individual constant or a function string having terms as arguments.

To distinguish between variables and individual constants we will adopt the convention that small letters denote variables and capital letters individual constants. Thus \( q, t, x \) are variables and \( H, L \) and \( R \) are individual constants. Examples of function strings are \( l(x',x), r(z,z') \) and \( \text{tape}(u,v,w) \).

We get an atomic formula if and only if we apply a predicate string to zero or more terms. An expression is a well-formed formula of predicate logic if and only if it is an atomic formula, or if \( A \) and \( B \) are well-formed formulas and \( y \) a variable, then \( \sim A, A \rightarrow B \) and \( (y)A \) are well-formed formulas which mean "not \( A \)", "\( B \) if \( A \)" and "for all \( y \) \( A \)".

We will, of course, use the connectives conjunction, disjunction, equivalence and the existential quantifier. They are introduced by the following definitions:

\[
\begin{align*}
A \land B & \quad \text{for} \quad \sim(A \rightarrow \sim B) \\
A \lor B & \quad \text{for} \quad (A \rightarrow B) \land (A \rightarrow \sim B) \\
A \leftrightarrow B & \quad \text{for} \quad (A \rightarrow B) \land (B \rightarrow A) \\
y \forall A & \quad \text{for} \quad \sim((x)\sim A).
\end{align*}
\]

Besides the introduced forms of predicate logic we will make use of the clausal form. The clausal form is interesting since it is a canonical form for the resolution principle (Robinson [11]). As we will see, this inference system can be given a computational interpretation.

Let us now introduce the clausal form. An expression is a literal if and only if it is an atomic formula or a negated atomic formula. Atomic formulas and negated atomic formulas are also called positive and negative literals.