1. INTRODUCTION

Proving the termination of algorithms is one of the challenges in program verification. Termination proofs can be quite complicated, e.g. proving the termination of McCarthy's 91-function

```plaintext
function f91(x: nat): nat <=
    if x > 100
    then x - 10
    else f91(f91(x + 11))
fi
```
or Takeuchi's function

```plaintext
function tak(x, y, z: nat): nat <=
    if x ~ y
    then y
    else tak(tak(x - 1, y, z), tak(y - 1, z, x), tak(z - 1, x, y))
fi
```
require a sophisticated argumentation, cf. (Manna, 1974; Moore, 1979; Feferman, 1991; Knuth, 1991), and for other algorithms as e.g.

```plaintext
function col(x: nat): nat <=
    if x ~ 1
    then x
    else if even(x)
        then col(x/2)
        else col(3x + 1)
    fi
fi
```
(attributed to Collatz) termination is an open problem for more than 50 years (Gardner, 1983). On the other hand, proof procedures exist which verify the termination of "standard" procedures for arithmetic functions, as e.g.
function gcd(x, y : nat) : nat ⇐
  if y = 0 then x else if x ≥ y then gcd(x - y, y) else if x = 0 then y else gcd(x, y - x) fi fi

for sorting, or for operations on trees and graphs etc. without any human advice (Walther, 1994b; Giesl, 1995; Giesl et al., 1998). Hence termination can be proved in a uniform way in some cases at least. ¹

In this paper we illustrate which problems must be solved for proving the termination of an algorithm. We formulate some sufficient criteria and demonstrate their success and failure by examples. We also define classes of algorithms by simple syntactic requirements, whose termination usually is easier to verify compared to the general case.

Informally, the termination of an algorithm means that each computation involving the algorithm halts after finitely many steps. A computation of a machine M is modelled by a function eval which maps expressions of a program into values of the machine. The function eval is called the interpreter of M and is formally given as a partial function eval : EXPR → VAL, where VAL is the set of all values of M and EXPR denotes the set of expressions of a program P, consisting of calls of procedures defined by P as well as calls of base operations provided by the machine M.

We need some syntax to represent values, expressions and programs, and we use standard notions of predicate logic, cf. e.g. (Gallier, 1986), and term rewriting, cf. e.g. (Avenhaus, 1995; Baader and Nipkow, 1998), for that purpose. In particular, we write \( \mathcal{T}(\Sigma, \mathcal{V})_s \) for the set of many-sorted terms of sort s over some S-sorted signature \( \Sigma \) for function symbols, a set \( \mathcal{V} \) of variable symbols and a set \( S \) of sort symbols, where \( S = \{ \text{nat, bool} \} \) throughout the paper. \( \mathcal{T}(\Sigma)_s \) abbreviates the set of all ground terms of sort s. We sometimes write terms as strings, i.e. \( f s_1 \ldots s_n \) is written instead of \( f(s_1, \ldots, s_n) \), and we use \( \mathcal{T}(\Sigma, \mathcal{V})_w \) etc. as a shorthand notation for the cartesian product \( \mathcal{T}(\Sigma, \mathcal{V})_{s_1} \times \ldots \times \mathcal{T}(\Sigma, \mathcal{V})_{s_k} \), where \( w = s_1 \ldots s_k \in S^+ \).

For sake of simplicity, we assume a machine M which operates on natural numbers and boolean values only.² Natural numbers are represented by the nullary base operation 0 (for zero) and the unary base operation succ denoting the successor function. So natural numbers are formally given by the set \( \mathcal{T}(\{0,\text{succ}\})_{\text{nat}} = \{0, \text{succ}(0), \text{succ}(\text{succ}(0)), \ldots \} \) of ground terms over 0 and succ and we use \{true, false\} as the set of boolean values. Subsequently, we write \( \mathcal{C}_{\text{nat}} \) as an abbreviation for \( \mathcal{T}(\{0,\text{succ}\})_{\text{nat}} \), \( \mathcal{C}_{\text{bool}} \) as an abbreviation for \{true, false\}, and we use \( \mathcal{C} := \mathcal{C}_{\text{nat}} \cup \mathcal{C}_{\text{bool}} \) instead of \( \text{VAL} \) to denote the set of all values on which M operates.