Enumerator

Wieger Wesselink

April 26, 2024

This document specifies an algorithm for enumeration. Given an expression φ of type T and a list of data variables v, the algorithm will iteratively report expressions $[\varphi_0, \varphi_1, \dots]$ that can be obtained from φ by assigning constant values to the variables in v.

Let R be a rewriter on expressions of type T, let r be a rewriter on data expressions, and let σ a substitution on data variables that is applied during rewriting with R. Furthermore let P be a queue of pairs $\langle v, \varphi \rangle$, with v a non-empty list of variables and φ an expression. The function *report_solution* is a user supplied callback function. Whenever the callback function returns true, the while loop is interrupted. The predicate function $reject$ is used to discard an expression, so that it does not end up in the queue P. The predicate function *accept* is used to accept an expression as a solution, even though it may still have a non-empty list of variables. By default the $reject$ and *accept* functions always return false. The $reject$ function is not just a cosmetic detail. The termination of the enumeration may depend on it. Enumeration is often used to find solutions of boolean predicates. Then we typically reject the expression $false$ and accept the expression *true* or vice versa.

The is_finite case in the algorithm applies to finite function sorts and finite sets. We assume that all elements of such sorts can be obtained using the function values. We assume that for each sort s a non-empty set of constructor functions constructors(s) is defined.

ENUMERATE$(P, R, r, \sigma, \mathit{report_solution}, \mathit{reject}, \mathit{accept})$
while $P \neq \emptyset$ **do**
  let $\langle v, \varphi \rangle = \text{head}(P)$ **with** $v = [v_1, \ldots, v_n]$
  if $v = []$ **then**
    $\varphi' := R(\varphi, \sigma)$
    if $\mathit{reject}(\varphi')$ **then skip**
    else if $\mathit{report\_solution}(\varphi')$ **then break**
  else if $\mathit{reject}(\varphi)$ **then**
    skip
  else if $\text{is\_finite}(\text{sort}(v_1))$ **then**
    for $e \in \text{values}(\text{sort}(v_1))$ **do**
      $\varphi' := R(\varphi, \sigma[v_1 := e])$
      if $\mathit{reject}(\varphi')$ **then**
        skip
      else if $\text{tail}(v) = [] \vee \mathit{accept}(\varphi')$ **then**
        if $\mathit{report\_solution}(\varphi)$ **then break**
      else
        $P := P +\!\!+ [\langle \text{tail}(v), \varphi' \rangle]$
  else
    for $c \in \text{constructors}(\text{sort}(v_1))$ **do**
      let $c : D_1 \times \ldots \times D_m \to \text{sort}(v_1)$
      choose $y_1, \ldots, y_m$ **such** that $y_i \notin \{v_1, \ldots, v_n\} \cup FV(\varphi)$, for $i = 1, \cdots, m$
      $\varphi' := R(\varphi, \sigma[v_1 := r(c(y_1, \ldots, y_m))])$
      if $\mathit{reject}(\varphi')$ **then**
        skip
      else if $\mathit{accept}(\varphi') \vee (\text{tail}(v) = [] \wedge (\varphi = \varphi' \vee [y_1, \ldots, y_m] = []))$ **then**
        if $\mathit{report\_solution}(\varphi)$ **then break**
      else
        if $\varphi = \varphi'$ **then** $P := P +\!\!+ [\langle \text{tail}(v), \varphi' \rangle]$
        else $P := P +\!\!+ [\langle \text{tail}(v) +\!\!+ [y_1, \ldots, y_m], \varphi' \rangle]$
  $P := \text{tail}(P)$

Remark 1 *The algorithm works both for data expressions and PBES expressions.*

Remark 2 *In the case of data expressions, R and r may coincide.*

Remark 3 *The algorithm can be extended such that it also returns the assignments corresponding to a solution.*

Remark 4 *In some applications of the enumerator solutions with a non-empty list of variables are unwanted. In that case the $\varphi = \varphi'$ cases in the algorithm need to be removed. A boolean setting accept_solutions_with_variables is introduced to control this.*