# Enumerator

Wieger Wesselink

January 19, 2019

This document specifies an algorithm for enumeration. Let $\varphi$ be an expression of type $T$, let $v = FV(\varphi)$ be the list of free variables occurring in $\varphi$, and let *accept* be a predicate function on expressions of type $T$. We define a *solution* of $\varphi$ as a ground term consisting of constructors only, that is obtained by assigning values to the variables in $v$ such that $\mathit{accept}(\varphi)$ evaluates to true. The enumeration algorithm iteratively computes solutions of the expression $\varphi$.

Let $R$ be a rewriter on expressions of type $T$, let $r$ be a rewriter on data expressions, and let $\sigma$ a substitution on data variables that is applied during rewriting with $R$. We assume that for each sort $S$ a set of constructor functions constructors$(S)$ is defined, such that constructors(sort$(d)) \neq \emptyset$ for all $d \in v$. A precondition of the algorithm is that for all $v_i \in v$ we have $\sigma(v_i) = v_i$.

$\textsc{Enumerate}(v, \varphi, \mathit{accept}, R, r, \sigma)$
$P := [\langle v, R(\varphi, \sigma)\rangle]$
$\Omega := \emptyset$
**while** $P \neq \emptyset$ **do**
    **let** $\langle v, \varphi\rangle = \mathrm{head}(P)$
    $P := \mathrm{tail}(P)$
    **let** $v = [v_1, \ldots, v_n]$
    **if** $\mathit{accept}(\varphi)$ **then**
        **if** $v = []$ **then**
            $\Omega := \Omega \cup \{\varphi\}$
        **else**
            **if** $\mathrm{constructors}(\mathrm{sort}(v_1)) \neq \emptyset$ **then**
                **for** $c \in \mathrm{constructors}(\mathrm{sort}(v_1))$ **do**
                    **let** $c : D_1 \times \ldots \times D_m \to \mathrm{sort}(v_1)$
                    **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** $\varphi' = \varphi$ **then**
                        $P := P +\!\!+ [\langle [v_2, \ldots, v_n], \varphi'\rangle]$
                    **else**
                        $P := P +\!\!+ [\langle [v_2, \ldots, v_n, y_1, \ldots, y_m], \varphi'\rangle]$
            **else**
                **error**
**return** $\Omega$

where $\epsilon$ is the empty substitution.

**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 easily extended such that it also returns the assignments corresponding a solution.*

**Remark 4** *The most common use case is to take an expression $\varphi$ of type Bool, and to choose accept$(\varphi) \equiv \varphi \neq false$. Then it can for example be used to compute all assignments to $FV(\varphi)$ that cause a condition $\varphi$ to be evaluated to true.*

The enumeration can be extended to finite sets and functions by adding

$$
\begin{array}{l}
\textbf{else if } \mathrm{sort}(v_1) = Set(E) \text{ with } E \text{ finite } \textbf{then} \\
\quad \textbf{for } e \in \mathrm{subsets}(E) \textbf{ do} \\
\quad\quad P := P \mathbin{+\!\!+} [\langle [v_2, \ldots, v_n], R(\varphi, \sigma[v_1 := e]) \rangle] \\
\textbf{else if } \ \mathrm{sort}(v_1) = D_1 \times \ldots \times D_m \to D \ \ \textbf{then} \\
\quad \textbf{for } f \in \mathrm{functions}(\mathrm{sort}(v_1)) \textbf{ do} \\
\quad\quad P := P \mathbin{+\!\!+} [\langle [v_2, \ldots, v_n], R(\varphi, \sigma[v_1 := f]) \rangle]
\end{array}
$$