TOMÁŠ KROUPA, TOMÁŠ VALLA, Constructing many-valued logical functions with small influence of their variables.  
Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic, Prague, Czech Republic.
E-mail: kroupa@utia.cas.cz.
Faculty of Information Technology, Czech Technical University in Prague, Czech Republic.
E-mail: tomas.valla@fit.cvut.cz.

§1. Introduction. The Boolean functions with small influence of their inputs are used in the collective coin flipping algorithms [2]. In this contribution we replace the random bit generator with a random generator over a finite set and we show the existence of finitely-valued Lukasiewicz formulas with small influence of their variables.

§2. Lukasiewicz logic. We repeat basic definitions and results concerning finite-valued Lukasiewicz logic and its Lindenbaum algebra [3, 1]. We consider only finitely-many propositional variables $A_1,\ldots, A_n$. Formulas $\varphi, \psi, \ldots$ are then constructed from these variables and the truth-constant $\bar{0}$ using the following basic connectives: negation $\neg$ and strong disjunction $\oplus$. For any $k \in \mathbb{N}$ the semantics for connectives of $(k+1)$-valued Lukasiewicz logic is given by the corresponding operations of the finite $MV$-chain, which is just the set of rational numbers $L_k = \{0, \frac{1}{k}, \ldots, \frac{k-1}{k}, 1\}$ endowed with constant zero 0 and the operations of negation $\neg$ and strong disjunction $\oplus$ defined as $\neg a = 1 - a$ and $a \oplus b = \min(1, a + b)$, respectively, for each $a, b \in L_k$. The structure $(L_k, \oplus, \neg, 0)$ then becomes an $MV$-algebra. The operations $\odot, \land, \lor, \to$ are introduced in the standard way.

In the sequel we consider the expansion of the $(k+1)$-valued Lukasiewicz logic with the truth constants from the chain $L_k$. The language of $(k+1)$-valued Lukasiewicz logic with truth constants results from the language of $(k+1)$-valued Lukasiewicz logic by adding the truth constant $\bar{r}$ for each $r \in L_k$. Every truth constant is a formula. Formulas are built from propositional variables $A_1,\ldots, A_n$ and truth constants using the connectives $\oplus$ and $\neg$ as well as other defined connectives of Lukasiewicz logic. Let $F^n_k$ be the Lindenbaum algebra of $(k+1)$-valued Lukasiewicz logic with truth constants over $n$ variables. By [1] we may identify $F^n_k$ with the product $L_k^{\otimes n}$ whose elements are all the functions $f: L^n_k \to L_k$.

§3. Influence of Boolean variables. Boolean functions have a natural interpretation in game theory. A Boolean function is called a simple game. For every $y = (y_1,\ldots, y_{i-1}, y_{i+1},\ldots, y_n) \in L_{n-1}^k$, we denote by $f_y$ the function $L_k \to L_k$ defined by $f_y(x) = f(y_1,\ldots, y_{i-1}, x, y_{i+1},\ldots, y_n)$, where $x \in L_k$.

The following notations will be used throughout the paper. Let $f \in F^n_k$ and $i \in \{1,\ldots, n\}$. For every $y = (y_1,\ldots, y_{i-1}, y_{i+1},\ldots, y_n) \in L_{n-1}^k$, we denote by $f_y$ the function $L_k \to L_k$ defined by $f_y(x) = f(y_1,\ldots, y_{i-1}, x, y_{i+1},\ldots, y_n)$, where $x \in L_k$.

The influence $\beta_i(f)$ of variable $x_i$ on a monotone Boolean function $f \in F^n_k$ is defined by

\[ \beta_i(f) = \min \left\{ \alpha \mid \exists y \in L_{n-1}^k : f_y(x_i) \neq f_y \right\} \]
as the probability that \( f_y^{-1} \) remains non-constant when \( y \in \{0,1\}^{n-1} \) is selected at random: \( \beta_i(f) = \sum_{y \in \{0,1\}^{n-1}} \frac{f_y^{-1}(y) - f_y^{-1}(0)}{2^{n-1}}. \) The number \( \beta_i(f) \) is also called the Banzhaf index of player \( i \) in a coalition game \( f \). We also define the Banzhaf index \( \beta(f) \) of \( f \) as \( \beta(f) = \max \{ \beta_1(f), \ldots, \beta_n(f) \}. \) The Banzhaf index measures the influence of players.

A natural motivation for investigating the players’ influence comes from the collective random bit generators. Suppose there are \( n \) computers equipped with random generators. The task is to generate one random bit identical for all the machines. Simultaneously, each machine produces a uniform random bit and announces it to other machines. Each of them then has to perform a computation based on those inputs to produce the identical uniform random bit. The question is to which extent is a given random generator resistant towards possible third party attacks and corruption of one machine. The goal of the design of Boolean functions with low variables’ influence is to minimize the chance of the attacker to manipulate the result.

Consider the Boolean function \( f(x_1, \ldots , x_n) = x_k \) called the dictatorship of player \( k \). The influence of dictator \( k \) is 1 and 0 for other players: \( \beta(f) = 1. \) The Boolean majority function \( m \in F_3^n \) is defined as follows. Let \( n \) be odd and let \( m(x_1, \ldots , x_n) = 1 \) if there is a set \( S \subseteq \{1, \ldots , n \} \), \( |S| > n/2 \), such that \( x_i = 1 \) for \( i \in S \), and 0 otherwise. It follows that \( \beta(m) = \Theta(1/\sqrt{n}). \) On the other hand, it was shown \([5, 4]\) that for any Boolean function the average influence of a variable is at least \( \Omega(1/n) \). Surprisingly, there exists a Boolean function \( L \) performing better than the majority functions. In the next theorem we identify the vertices of \( \{0,1\}^n \) with the subsets of \( \{1, \ldots , n \} \).

**Theorem 1** (Ben-Or and Linial [2]). There exists a construction of the function \( L \in F_3^n \) such that \( |L^{-1}(0)| = |L^{-1}(1)| = 2^{n-1} \) and \( \beta(L) = O\left(\frac{\log n}{n}\right)\).

The rough idea how the function \( L \) is constructed is as follows. Let \( b \) be the unique solution of the equation \( (2^b - 1)^{1/b} = 2^{1-1/n} \). Decompose the set \( \{1, \ldots , n\} \) into \( n/b \) blocks of size \( b \) and consider the set \( J \) of those subsets of \( \{1, \ldots , n\} \) which contain no block. Let \( L \) be defined such that \( L(A) = 0 \) if \( A \in J \) and \( L(A) = 1 \) otherwise.

### 4. Influence of variables in many-valued logics

We will propose a natural generalization of Boolean Banzhaf index. Let \( f \in F_h^n, k \geq 1, \) and \( i \in \{1, \ldots , n\}. \) The influence of variable \( x_i \) on \( f \) is \( \gamma_i(f) = (k+1)^{k-1} \sum_{y \in L_h^{k-1}} \left( \max_{x \in L_h} f_y^{-1}(x) - \min_{x \in L_h} f_y^{-1}(x) \right). \)

It can be shown that \( \gamma_i(f) \) is a faithful generalization of the Banzhaf index. A natural next step is to design functions with low variable influence in the setting of the finitely-valued Lukasiewicz logic with truth constants. Our setting is the case of random generators producing a number from a finite set. Note that this setting naturally allows designs that may ask the input generators repeatedly.

### 4.1. \( 2^k \)-valued logic

Let us consider the set \( L_{h-1} \) for \( h = 2^k \) for some nonnegative integer \( k \). In the sequel, we will naturally identify the elements \( S_h = \{0,1, \ldots , h-1\} \) with those in \( L_{h-1} \). Note that we may encode each \( x \in \{0,1, \ldots , h-1\} \) by a \( k \)-element Boolean vector \( (x^1, x^2, \ldots , x^k) \) representing the binary number \( x \), with \( x^1 \) being the highest bit and \( x^k \) the lowest. Under the identification of \( S_h \) with \( L_{h-1} \), we may analogously use the Banzhaf index \( \gamma_i(f) = h^{2-k} \sum_{y \in S_h^{k-1}} (\max_{x \in S_h} f_y^{-1}(x) - \min_{x \in S_h} f_y^{-1}(x)) \). Let us define the function \( f : S_h^n \rightarrow S_h \) as

\[
(1) \quad f(x_1, \ldots , x_n) = \left( L(x^1_1, x^2_1, \ldots , x^k_1), L(x^1_2, \ldots , x^k_2), \ldots , L(x^1_n, \ldots , x^k_n) \right),
\]

where the value \( f(x_1, \ldots , x_n) \) is the binary representation of a number in \( S_h \).

We shall prove that \( f \) has a small variable influence.
Theorem 2. For $i = 1, \ldots, n$, $\gamma_i'(f) = O\left(\frac{\log n}{n}\right)$.

Proof. The Banzhaf index of the function $L' = L(x_1^i, \ldots, x_n^i)$ is $\beta_i(L') = O(\log n/n)$ by [2]. Observe that in the resulting vector of $f$ the value of the highest bit $L(x_1^i, x_2^i, \ldots, x_n^i)$ has the same effect on the size of the output value as the sum of all other lower bit orders, and the same holds for the influence of each lower bit. We may thus bound

$$\gamma_i'(f) \leq (2^{k-1} + 2^{k-2} + \cdots + 1) \cdot \beta(L') \leq 2^k \cdot \beta(L') = O\left(\frac{\log n}{n}\right).$$

4.2. General many-valued logic. Let us now consider the set $S_h$ with $h > 2$ and let $\ell$ be the smallest integer such that $2^\ell \geq h$. Let us denote by $G_2$ a random generator producing one uniform random bit. We construct the generator $G_{2\rightarrow h}$ that uses $G_2$ as the input and produces a value from $S_h$ with uniform distribution:

1. Produce a number $N$ by reading random bits $b_1, \ldots, b_\ell$ from the $G_2$.
2. If $N < h$ then return the number $N$.
3. Otherwise, repeat the whole process.

Lemma 3. The generator $G_{2\rightarrow h}$ produces a result with a uniform distribution over $S_h$. The expected number of random bits read from $G_2$ is $\ell 2^\ell / h = \Theta(\log h)$.

Proof. W.l.o.g., let $p$ denote the probability that 0 is on the output of $G_{2\rightarrow h}$. Then

$$p = \frac{1}{2^\ell} + \frac{2^\ell - h}{2^\ell} p$$

since with probability $1/2^\ell$ the result is produced immediately and with probability $(2^\ell - h)/2^\ell$ the process is repeated independently on the previous round. Solving the equation yields $p = 1/h$.

Denote by $E$ the expected number of random bits needed to produce the result. Similar equation $E = \ell + \frac{2^\ell - h}{2^\ell} E$ holds as $\ell$ bits are used always and with probability $(2^\ell - h)/2^\ell$ the whole memoryless process repeats. The solution gives $E = \ell 2^\ell / h$. Finally, note that $\ell \approx \log_2 h$ and $h \leq 2^\ell < 2h$.

Let us denote $G_h$ a random generator producing uniform $S_h$-valued output. Using analogous technique, we design a generator $G_{h\rightarrow 2}$ which reads uniformly distributed random $S_h$-valued input and produces a uniformly distributed random bit. The process is as follows. If $h$ is even, one random input is read and its parity is returned. If $h$ is odd, the following procedure is used.

1. Read one random number $N \in S_h$ from $G_h$.
2. If $N \neq h - 1$, return the parity of $N$.
3. Otherwise, repeat the whole process.

Lemma 4. The generator $G_{h\rightarrow 2}$ produces a uniform random bit. If $h$ is even, $G_{h\rightarrow 2}$ reads 1 random input value. If $h$ is odd, the expected number of random values read is $h/(h - 1)$.

Proof. Let $p$ denote the probability that $G_{h\rightarrow 2}$ produces, w.l.o.g., 0. Then $p = (h - 1)/h + 1/h$ as with probability $(h - 1)/(2h)$ the result is produced immediately and with probability $1/h$ the process is repeated independently on the previous round. We get $p = 1/2$.

Denote by $E$ the expected number of random $S_h$-values needed to produce the result. We have that $E = 1 + \frac{1}{h} E$ holds as 1 value is used always and with probability $1/h$ the memoryless process repeats, which gives the solution $E = h/(h - 1)$.
4.3. Function with low influence of variables. We describe the generator $G$ which is given $n$ uniform $S_h$-valued random generators and produces the $S_h$-valued output with a low influence of the input generators. Let us denote the input random $S_h$-value generators by $g_1, \ldots, g_n$.

The generator is constructed as $G = G_{2 \to h} \left( L(G_{h \to 2}(g_1), G_{h \to 2}(g_2), \ldots, G_{h \to 2}(g_n)) \right)$, where $L$ is the function from Theorem 1. In another words, the generator $G_{2 \to h}$ repeatedly asks for Boolean bits from the function $L$, which in turn asks the $n$ generators $G_{h \to 2}$ connected to the inputs of $L$, which in turn ask the input generators $g_1, \ldots, g_n$.

**Lemma 5.** The probability that $G$ produces $v \in S_h$ is $1/h$.

**Proof.** By Lemma 4, each generator $G_{h \to 2}(g_i)$ produces uniform random bit. By Theorem 1, the function $L$ then produces a uniform random bit. Finally, the generator $G_{2 \to h}$ produces uniform $S_h$-valued output.

Let $\ell$ be the smallest integer such that $2^\ell \geq h$.

**Lemma 6.** For the generator $G$, the expectation of the total number of random values produced by the generators $g_1, \ldots, g_n$ in order to obtain one output of $G$ is $n \cdot (2^\ell / (h-1))$.

**Proof.** Recall that by Lemma 4 the expectation of random values needed by the generator $G_{h \to 2}$ to produce single output is $h/(h-1)$. As the generators $g_1, \ldots, g_n$ are independent, linearity of expectation yields that to produce one output bit of the function $L$, $nh/(h-1)$ input values are needed in the expectation. Since each execution of $L$ is independent on the previous runs, by Lemma 3 we obtain the total expectation $(t2^\ell / nh)/(h-1) = n(t2^\ell / (h-1))$.

**Lemma 7.** For each $i = 1, \ldots, n$, we have $\gamma_i(G) = O((2^\ell \log n)/n)$.

**Proof.** Observe that the function $L' = L(G_{h \to 2}(g_1), \ldots, G_{h \to 2}(g_n))$ behaves exactly as the function $f$ defined by (1). Theorem 2 yields $\gamma_i(L') = O((2^\ell \log n)/n)$. During the step 1 of the generator $G_{2 \to h}$, the function $L$ is called $\ell$ times, which produces a number $N$ in the range $0, \ldots, 2^\ell - 1$. Less than one half of the possible values of $N$ is rejected and step 1 is repeated independently on the result of the previous iteration.

We may identify $G$ with a unique function $\tilde{G} : L_n^o \rightarrow L_{h-1}$. Note that $\gamma_i(\tilde{G}) = \gamma_i(G)/(h-1)$. Since $\ell \approx \log_2 h$, using Lemma 6 and Lemma 7 we may conclude with the following corollary.

**Corollary 8.** The generator $\tilde{G}$ needs in total $\Theta(n \log h)$ input random values in expectation, and $\max_{i=1, \ldots, n} \gamma_i(\tilde{G}) = O(\log n/n)$.


