International Journal of Pure and Applied Mathematics
Volume 60 No. 2 2010, 149-168

# ON THE STRUCTURE OF BOOLEAN SATISFIABILITY

Silvano Di Zenzo
IBM Rome Scientific Center
University of Rome "La Sapienza"
181, C. Colombo St., Rome, 00147, ITALY
e-mail: silvano.dizenzo@acm.org

**Abstract:** We define various notions of structural independence of a decision problem. We then show that $SAT$ exhibits each of these properties. Using this result, we show that $SAT$ has no collective certificates of membership of the kind that we call "wizards." As a consequence, all the programs $P$ that solve $SAT$ have the same "kernel."

**AMS Subject Classification:** 68Q15
**Key Words:** NP problems, certificates of membership, structural independence, witness, wizard, logogram, program kernel, irreducible kernel

## 1. Introduction

We develop an application to $SAT$, to be positioned in current stream of interest in the structure of Boolean satisfiability (Kirousis and Kolaitis [12], Allender et al [2], and Gopalan et al [9], among others). We use *strings* in a technical sense, taken from computability (Odifreddi [14]).

An *encoded decision problem* over an alphabet $\Sigma$ is a pair $(E, F)$ where $E$=words that encode instances of the problem, $F$=words to be accepted. On input $x$, a decision program $P$ for problem $(E, F)$ either *accepts* $x$ (if $x$ is in $F$) or *rejects* $x$ (if $x$ is in $E - F$) or else *discards* $x$ (for $x$ outside $E$).

Our fundamental construct is a set of strings $Log_E(F)$ called *logogram* of $F$ relative to $E$ that conveys structural information on $E$, $F$, and how $F$ is embedded in $E$. We mostly use the reduced version $|Log_E(F)|$, consisting of those strings in $Log_E(F)$ that do not include other strings in same set. The *kernel* $Ker(P)$ of a program $P$ that solves $(E, F)$ is the set of those

Received: February 15, 2010 

strings in $|Log_E(F)|$ that are actually used by $P$ in making decisions. There are relationships between the composition in terms of strings of the kernel $Ker(P)$ of a program $P$ solving problem $(E, F)$ and the complexity of $P$.

Our application to $SAT$ uses a property of structural independence of a decision problem that we call "strong internal independence." Think of a computation in which the result of any computation step does not change the results that are left possible for subsequent steps. Internal independence is defined in terms of a relation of *entanglement* between sets of strings relative to reference set $E$.

Our results are the following. We show that $SAT$ exhibits the strong internal independence property: Intuitively, no "entanglement at distance" between strings belonging to the logogram of $SAT$ is possible. Besides, we show that $SAT$ cannot have, in its reduced logogram, certain collective certificates that we call *wizards*. The wizards are strings that, if included in an input word, indicate that that input has solutions: However, a wizard does not include any specification of the particular solutions that satisfy that word. By way of consequence, all the decision programs $P$ that solve $SAT$ have same kernel.

## 2. Decision Strings

We first recall notions regarding the certificates of membership in NP theory. As next step, we illustrate possible use of strings to represent certificates. We conclude the section by reviewing various basic properties of strings.

Let $G \subseteq \Sigma^* \times \Sigma^*$ so that $G$ is a relation on words over $\Sigma$. Let $Dom(G)$ and $Cod(G)$ be first and second projection of $G$. A relation $G$ which is both polynomial-time decidable and polynomially balanced is an NP relation. $L$ is in NP if and only if there exists an NP relation $G$ such that $L = Dom(G)$. We interchange problems with languages, thus $(E, F) \in$ NP and $F \in$ NP amount to same.

Let $(E, F)$ be an NP problem. Then there exists a sequence $y_1$, $y_2$,.. of words (over some appropriate alphabet) called *solutions* or else *certificates of membership* for problem $(E, F)$. For any instance $x \in E$ we have that $x$ can possibly be satisfied by some of the $y_i s$. We also have "unsatisfiable" instances. What "satisfaction" means operationally pertains to problem $(E, F)$.

Cardinality function $\alpha(n)$ of an NP problem: We may arrange notations so that all solutions that can possibly satisfy an $x$ of size $n$ are between $y_1$ and $y_{\alpha(n)}$. Associated with solutions $y_1$, $y_2$,.. there is a decomposition of the target

set $F$ into subsets $F_i$ called *solution regions*, where $F_i$ is the set of those $x's$ that are satisfied by $y_i$. Solution regions satisfy the obvious relation $F = \cup_i F_i$.

**Decision Tests.** In this paper the ordinary certificates of membership of an NP problem are associated with certain *decision tests*, also called *decision strings*, that exist in any NP problem. The decision tests consist of *strings*, defined to be functions $N \to \Sigma$ with finite domain ($N$=positive integers). In loose words, a string $g$ being *included* (or *subsumed*) in a word $x$ is that which remains by canceling zero or more letters in $x$, while leaving blanks in the places of letters. The introduction of the decision tests allows us to define certain more general certificates of membership that we call *wizards*. The ordinary solutions $y_1$, $y_2$,.. continue to be the certificates of membership associated with an NP problem as in the standard NP theory.

We assume that satisfiability, being a property exhibited by certain words, is accompanied by characteristic *signs*, that we think as distinctive marks, or signatures, being somehow inscribed within the word $x$ under study. A detailed discussion would indicate strings as proper formalization of such notions as "mark" or "signature." Thus, we assume that signs consist of strings interspersed in $x$. Since strings represent words in shorthand, we refer to sets of strings as *logograms*.

**Properties of the Strings.** We define $\Sigma_\infty$ to be the set of all strings over $\Sigma$. We look at a string $g \in \Sigma_\infty$ as a prescription that a word $x$ over $\Sigma$ may or may not satisfy. If $Dom(g)$ is an initial segment of $N$ then $g$ is an ordinary word: Thus, words are certain special strings. The length (or size) $|g|$ is a number associated with $g$, being at least as large as the greatest number in $Dom(g)$.

$\Sigma_\infty$ is partially ordered. Given $f, g \in \Sigma_\infty$, $g$ is an *extension* of $f$, written $f \leq g$, as soon as $Dom(f) \subseteq Dom(g)$ and $g$ takes same values as $f$ in $Dom(f)$. The expression "$f \leq g$" is often verbalized as "$f$ is included in $g$" or else "$f$ is subsumed in $g$". If $f \leq g$ and $g \leq f$ then $f = g$. If $f \leq g$ but not $g \leq f$, write $f < g$ and say $g$ is a proper *extension* of $f$, or else $f$ is a proper *restriction* of $g$. The empty partial function $N \to \Sigma$ , noted $\perp$, is the *void* string, and $Dom(\perp) = \emptyset$. Any $f$ in $\Sigma_\infty$ is an extension of $\perp$, thus $(\Sigma_\infty, \leq)$ has a least element $\perp$.

Two strings $f$ and $g$ are *compatible* as soon as $f(x) = g(x)$ for any $x$ in $Dom(f) \cap Dom(g)$. If $f, g$ are disjoint, which is to say $Dom(f) \cap Dom(g) = \emptyset$, then $f$ and $g$ are certainly compatible. The *meet* $f \wedge g$ of any pair $f, g$ is the restriction of $f$ (or $g$) to that portion of the intersection $Dom(f) \cap Dom(g)$ where $f$ and $g$ agree. The *join* of two compatible strings $f, g$, noted $f + g$, is

the least string which is an extension of both $f$ and $g$. Thus $f, g \leq f + g$ and $Dom(f + g) = Dom(f) \cup Dom(g)$. Equipped with meet and join, $\Sigma_\infty$ is an upward directed complete meet-semilattice, see Gierz et al [8].

To be able to iterate the operation of join we define the join of two sets of strings. Thus we define the join of the nonempty sets of strings $H, K$ to be the set of strings formed by joining each string of $H$ with each string of $K$ and aggregating all the joins formed in this way in one set. Thus, the join of $H$ and $K$ is the set of strings $H + K = \{a + b : a \in H, b \in K\}$.

## 3. Expansion and Logogram

The cylinders defined below are as in computability (the formalism is slightly different). The logogram is a newcomer in computer science. Entanglement is a key concept to deal with the internal structure of computational problems.

**Cylinders.** Given $H \subseteq \Sigma_\infty$ we define

$$Exp(H) = \{x \in \Sigma^* : \exists a \in H \ \ (x \geq a)\}. \tag{1}$$

Thus, $Exp(H)$=set of all words that include strings from $H$. Call $Exp(H)$ the *absolute expansion*, equivalently, the *absolute cylinder* associated with $H$. Note that $Exp(H)$ is the union of the elementary cylinders $Exp(g)$ for $g \in H$.

Given any recursive set of words $E$, we write $\Sigma_\infty(E)$ for the set of all strings that happen to be included in words of $E$, thus

$$\Sigma_\infty(E) = \{g \in \Sigma_\infty : Exp(g) \cap E \neq \emptyset\}. \tag{2}$$

$\Sigma_\infty(E)$ is the set of those strings $g$ in $\Sigma_\infty$ whose associated cylinder $Exp(g)$ intersects $E$. We think of $E$ as the set of words over $\Sigma$ that encode instances of some fixed reference computational problem $\Pi$. (Whenever we talk of a reference set $E$ there is implicit reference to some fixed abstract decision problem $\Pi$ as well as to a program $P$ solving $\Pi$.) For $H \subseteq \Sigma_\infty(E)$ we write

$$Exp_E(H) = E^H = \{x \in E : \exists a \in H \ \ (x \geq a)\} = E \cap Exp(H). \tag{3}$$

Thus, $E^H$ is the set of those words in $E$ that contain strings from $H$. $E^H$ is the *expansion* of $H$ relative to base $E$. We actually regard $E^H$ as a relativized cylinder, equivalently, as being a cylinder relative to a reference set $E$. Note that for $E = \Sigma^*$ we regain the absolute expansion of set $H$. Given $H \subseteq \Sigma_\infty(E)$, correspondence $Exp_E : H \to E^H$ exhibits properties:

$$E^H \cup E^K = E^{H \cup K}, \quad E^H \cap E^K = E^{H+K}. \tag{4}$$

Thus, unions and intersections of sets that are cylinders relative to reference

set $E$ are cylinders relative to reference set $E$. For any $H, K \in \Sigma_\infty(E)$,

$$H \subseteq K \Rightarrow E^H \subseteq E^K, \tag{5}$$

$$Exp_E(Exp_E(H)) = Exp_E(E^H) = E^H. \tag{6}$$

Having defined the cylinders, we are now able to make a few additional remarks on the certificates of membership in NP theory. Let $(E, F)$ be an NP problem. If an instance $x$ is satisfied by a solution $y_i$, then, according to NP theory, there exists at least one certificate of the membership of word $x$ in region $F_i$. These certificates are accompanied by certain decisive strings interspersed in those words of the reference set $E$ that happen to belong to $F$. Thus, we may rely upon the following circumstance: If instance $x$ is satisfied by solution $y_i$, then $x$ subsumes at least one decisive string $g$ such that the relativized cylinder $Exp_E(g)$ is fully included in the target set $F$. These decision strings are the elements of the logogram of a decision problem, that we define in the next subsection.

**Logograms.** We now introduce the *logogram* of a set of words $A$ relative to a reference set $E$. Given $A \subseteq E$, we define

$$Log_E(A) = \{g \in \Sigma_\infty(E) : \forall x \in E(x \geq g \Rightarrow x \in E^A)\}. \tag{7}$$

Since ordinary words are strings, $A$ can be regarded as a set of strings, hence $E^A$ is defined. $E^A$ is the relative cylindrification of $A$ in $E$, and this relative cylindrification is in turn the set of all words in $E$ that are prefixed by words from set $A$.

If $A$ is a cylinder in $E$, i.e., if $A = E^H$ for some $H \subseteq \Sigma_\infty(E)$, then $E^A = A$ by equation (6). If $A$ happens to be a cylinder in $E$, then equation (7) can be rewritten with $E^A$ replaced with $A$. Also note that, for $E = \Sigma^*$ equation (7) gives the *absolute logogram*.

The most interesting property of the correspondence $Log_E : E \to \Sigma_\infty(E)$ is the following. Given any pair of sets $A, B \subseteq E$,

$$Log_E(A \cup B) \supseteq Log_E(A) \cup Log_E(B). \tag{8}$$

Let us understand this inclusion. $Log_E(A \cup B)$ is the set of all strings that, for $x$ in $E$, are able to trigger event $x \in E^{A \cup B} = E^A \cup E^B$. A string that triggers $x \in E^A$ certainly belongs to $Log_E(A \cup B)$. Analogously, a string that triggers $x \in E^B$ certainly belongs to $Log_E(A \cup B)$. Thus, $Log_E(A) \cup Log_E(B)$ certainly is a subset of $Log_E(A \cup B)$. However, there can be strings $f$ whose inclusion in a word $x \in E$ is a sufficient condition for event $x \in E^A \cup E^B$ but not for $x \in E^A$ or $x \in E^B$. Thus, in the general case $Log_E(A \cup B)$ is not the same set as $Log_E(A) \cup Log_E(B)$.

**Entanglement.** The presence of certain strings in a word may entail that of certain others. Given $H, K \subseteq \Sigma_\infty(E)$, we write $K \sqsupseteq^E H$ if the following happens: Every word in $E$ which includes strings from $K$ also includes strings from $H$. For $E = \Sigma^*$ we write $\sqsupseteq$ for $\sqsupseteq^E$. (Think of strings in $K$ as spies, or else symptoms, for presence in an input string $x$ of strings from $H$.) If $H \sqsupseteq^E K$ and $K \sqsupseteq^E H$ then we write $H \equiv^E K$ and say that $H, K$ are *isoexpansive* relative to $E$. Note that, given any $H, K \subseteq \Sigma_\infty(E)$,

$$H \subseteq K \Rightarrow H \sqsupseteq K \Rightarrow H \sqsupseteq^E K. \tag{9}$$

We ask: Is there any easy piece of algebra linking expansion, logogram, entanglement? Then we construct a pair $(\alpha, \beta)$ of correspondences between sets of strings and sets of words: $\alpha$ carries any set of strings $H \subseteq \Sigma_\infty(E)$ into a corresponding set of words $H^\alpha \subseteq E$ and $\beta$ carries any set of words $A \subseteq E$ into a set of strings $A^\beta \subseteq \Sigma_\infty(E)$. These mappings have the properties:

$$H \sqsubseteq^E K \Rightarrow H^\alpha \supseteq K^\alpha, \tag{10}$$

$$A \subseteq B \Rightarrow A^\beta \sqsupseteq^E B^\beta, \tag{11}$$

$$H \sqsubseteq^E H^{\alpha\beta}, \quad A \subseteq A^{\beta\alpha}. \tag{12}$$

The connection is formally defined through the explicit expressions:

$$H^\alpha = E^H, \tag{13}$$

$$A^\beta = Log_E(A). \tag{14}$$

We emphasize that $A$ is any subset of $E$. Thus, given any subset $A$ of the reference set $E$ the function $A^\beta = Log_E(A)$ is defined. However, not all subsets $A$ of $E$ happen to be the conjugate set $H^\alpha$ of some set $H \subseteq \Sigma_\infty(E)$. If that happens, we say that $A$ is closed. Note that $A$ closed implies $E^A = A$.

**Theorem 1.** *$(\alpha, \beta)$ is a Galois connection.*

*Proof.* We must derive equations (10)-(12) from equations (13)-(14).

(I) Let $H, K \subseteq \Sigma_\infty(E)$ be given, and assume $H \sqsubseteq^E K$. Let $x \in K^\alpha$. By equation (13), $x \in E^K$. Then there is $g \in K$ such that $x \geq g$. Since $K \sqsupseteq^E H$, there exists $f \in H$ such that $x \geq f$. Then $x \in E^H$. But $E^H = H^\alpha$ by equation (13). Then $x \in H^\alpha$. Equation (10) is proved.

(II) Now we prove equation (11). Let $A, B \subseteq E$, assume $A \subseteq B$. It is left for us to prove $A^\beta \sqsupseteq^E B^\beta$. For, we must prove that if a word $x \in E$ includes a string $g$ from $A^\beta$ then $x$ also includes a string $f$ from $B^\beta$.

Assume $g \in A^\beta$ and, besides, assume that $x$ is a word from $E$ that subsumes string $g$, written $x \geq g$. Then, by virtue of equation (14), $g \in Log_E(A)$. Besides, by equation (7), for any word $y \in E$ we have $y \geq g \Rightarrow y \in E^A$. We conclude

that $x \in E^A$. But $A \subseteq B$, hence, by equation (5), $E^A \subseteq E^B$. Then, we also have $x \in E^B$.

Now, by equation (7), $Log_E(B)$ is the set of all the strings $f \in \Sigma_\infty(E)$ whose inclusion in a word from $E$ ensures that that word belongs to $E^B$. Besides, we have seen that the inclusion of string $g$ in a word $y$ from $E$ yields $y \in E^B$.

Then, string $g$ is a member of the logogram $Log_E(B)$.

By equation (14), $Log_E(B) = B^\beta$, hence $g \in B^\beta$.

We have shown that $A^\beta \subseteq B^\beta$. By equation (9), this implies equation (11).

(III) Next, we prove the first of equations (12).

Let $g \in H^{\alpha\beta}$. By equation (13), $H^\alpha = E^H$, hence, by virtue of equation (14), $H^{\alpha\beta} = (E^H)^\beta = Log_E(E^H)$. Thus, $g \in Log_E(E^H)$.

Assume that $x$ is any word in $E$ such that $x \geq g$.

Since $g$ is a member of $Log_E(E^H)$, and, besides, $x \geq g$, then equation (7) yields $x \in Exp_E(E^H)$.

Then, by one application of equation (6), we obtain $x \in E^H$. That means that there exists $f \in H$ such that $x \geq f$. We have shown the first of equations (12).

(IV) Next, we prove the second of equations (12). It follows from equations (13), (14) that $A^{\beta\alpha} = E^{Log_E(A)}$. On other hand one has $A \subseteq E^{Log_E(A)}$ for any $A \subseteq E$. Indeed, $A \subseteq E^A$ and $E^A = E^{Log_E(A)}$ from definitions, taking into account equation (6). We proved the second of equations (12). □

**Theorem 2.** *The following equations hold:*

$$H \subseteq H^{\alpha\beta}, \quad A \subseteq A^{\beta\alpha} \tag{15}$$

$$H^\alpha = H^{\alpha\beta\alpha}, \quad A^\beta = A^{\beta\alpha\beta} \tag{16}$$

*Proof.* From theory of Galois connection, see Birkhoff [4]. □

**Theorem 3.** *The map $H \to H^{\alpha\beta}$ is a closure operation in $\Sigma_\infty(E)$, and $A \to A^{\beta\alpha}$ is a closure operation in $E$.*

*Proof.* From theory of Galois connection, see Birkhoff [4]. □

The set of strings $Log_E(F)$ associated with a decision problem $(E, F)$ is our fundamental construct in this paper. It conveys information on the internal structure of problem $(E, F)$. We will mostly use the reduced version $|Log_E(F)|$, consisting of those strings in $Log_E(F)$ which do not include other strings from same set.

It follows from Theorem 3 that a set of words $A \subseteq E$ is closed as soon as

$A = A^{\beta\alpha}$. Note that only for a closed $A$ do we have that, for any $x$ in $E$, $x \in A$ if and only if there exists a $g \in Log_E(A)$ such that $x \geq g$. If and only if $A$ is a closed subset of the reference set $E$ we define the reduced logogram $|Log_E(A)|$. Regarding the reduced logogram $|Log_E(A)|$ of a closed set $A \subseteq E$, we explicitly note that, for any $x$ in $E$, $x \in A$ if and only if there is $g \in |Log_E(A)|$ such that $x \geq g$.

In this paper we consider encoded problems of the form $(E, F)$ where $E$ and $F$ are cylinder sets. We assume that the target set $F$ is decomposed according with $\cup_n F_n = F$ where the $F_i$s are solution regions. Also the solution regions are assumed to be cylinder sets. The rationale for these assumptions is that a major instance of an NP decision problem, namely $SAT$, exhibits all these properties, as will be seen when we develop our application to that problem. (We also remember that $SAT$ is known to be a cylinder set in all reasonable encoding schemes, see Balcazar et al [3].)

## 4. The Kernel of a Decision Program

We begin with a few remarks on the nature of the strings that happen to occur in the reduced logogram $|Log_E(F)|$.

As mentioned above, for any NP decision problem $(E, F)$ we assume $F$ to be a cylinder set. The strings in $|Log_E(F)|$ are certificates of membership for $F$ relative to $E$: For words in $E$, to include one or more strings from $|Log_E(F)|$ is necessary and sufficient for membership in $F$. In principle, we cannot exclude that $|Log_E(F)|$ may contain strings that behave as collective witnesses, also called wizards. In that case a program $P$ solving $(E, F)$ might do calculations that are functionally equivalent to testing input $x$ for possible inclusions of wizards.

Let $P$ solve problem $(E, F)$. The computations that $P$ performs are functionally equivalent to sequences of tests done on input $x$. This is part of current views of computation in logic, see Fagin et al [7], [6], and Halpern [10], and of semantics, see Scott [15] and Larsen and Winskel [13]. (The term "test" is used in logic: Terms such as "token" or else "piece of information" are most often used in studies of semantics.) Note that all current theories of semantics of computation are consistent with our developments as soon as we identify tokens with strings. In this view what $P$ eventually performs is a search of the input $x$ for possible inclusion of strings from the logogram $|Log_E(F)|$. That yields a view of computations as consisting of sequences of tests *in disguise*.

Let program $P$ solve problem $(E, F)$. The tests in $|Log_E(F)|$ are those that $P$ can use: They are so to speak at disposal for a program $P$. Which of these tests are actually used by $P$ is a different story. We define the *kernel* of program $P$, noted $Ker(P)$, to be the set of the strings from $|Log_E(F)|$ that $P$ actually uses for making decisions. The strings in $Ker(P)$ are uniquely identified by the algorithm that $P$ implements. The composition of $Ker(P)$ in terms of strings can also be determined through experiments with the executable of $P$.

A concept of great relevance for sequel is that of a *complete* subset of the reduced logogram $|Log_E(F)|$ of decision problem $(E, F)$: We define a set $H \subseteq |Log_E(F)|$ to be complete for problem $(E, F)$ as soon as, for any $x \in E$, one has $x \in F \Leftrightarrow \exists f \in H(f \leq x)$.

The proofs of following two theorems are not difficult and are omitted.

**Theorem 4.** *A necessary condition for $P$ to correctly solve $(E, F)$ is $Ker(P)$ complete for $(E, F)$.*

Let $H \subseteq |Log_E(F)|$ be a complete set of strings for $(E, F)$. We define $H$ to be *irreducible* for $(E, F)$ as soon as no proper subset $K \subset H$ happens to be complete for $(E, F)$.

**Theorem 5.** *Let $|Log_E(F)|$ be irreducible and programs $P, Q$ both solve $(E, F)$. Then $Ker(P) = Ker(Q)$.*

## 5. Independence of Decision Problems

In this section we use notions of dependence/independence of computations that were defined in Di Zenzo et al [5]. We first introduce a notion of pairwise independence of strings relative to a reference set $E$. As next step, we define a notion of internal independence of set $E$. Next we define notions of internal and strong internal independence of a decision problem.

**Mutual Independence of Strings.** Let $f, g$ be any two strings in $\Sigma_\infty(E)$ where $E$ is any infinite recursive set of words over alphabet $\Sigma$. According to definitions, $f$ entangles $g$ relative to $E$ as soon as, for all $x \in E$, $x \geq f \Rightarrow x \geq g$. We agreed that $f \sqsupseteq^E g$ means that $f$ entangles $g$ relative to $E$.

Observe that $f$ fails to entangle $g$ relative to $E$ if and only if there exists $x \in E$ such that $x$ contains $f$ and does not contain $g$. If $f \not\sqsupseteq^E g$ and $g \not\sqsupseteq^E f$ then $f$ and $g$ are said *mutually independent* relative to $E$; $f$ and $g$ are *mutually dependent* relative to $E$ when they fail to be mutually independent relative

to $E$. If $f, g$ are incompatible, then certainly $f, g$ are mutually independent relative to any $E$.

**Independence of a Decision Problem.** Our next step is to define the internal independence of a recursive set $E$. We define $E$ to be *internally independent* as soon as, given any $f, g \in \Sigma_\infty(E)$ one has $f \sqsupseteq^E g$ if and only if $f$ is part of $g$, that is to say, if and only if $f \leq g$.

Now we are ready to introduce the simple internal independence of a decision problem $(E, F)$. We call $(E, F)$ *internally independent* as soon as the strings in $|Log_E(F)|$ are mutually independent taken two by two.

**Theorem 6.** *If $E$ is internally independent then any decision problem $(E, F)$ based on $E$ as reference set exhibits the simple internal independence property.*

*Proof.* Let $E$ be any infinite recursive set exhibiting the internal independence property. Let $(E, F)$ be any decision problem based on $E$ as reference set. Let $f, g$ be any two strings in the reduced logogram $|Log_E(F)|$ of the problem.

(I) Assume $f, g$ incompatible. Since $f \in \Sigma_\infty(E)$, we have $E \cap Exp(f) \neq \emptyset$. Let $x \in E \cap Exp(f)$. Then $x$ is in $E$, $x$ includes $f$ and does not include $g$. Analogously, one can find a $y \in E$ which includes $g$ and does not include $f$. Thus, $f, g$ are mutually independent in $E$.

(II) Assume $f, g$ compatible. By the minimality property of the reduced logogram $|Log_E(F)|$ it cannot be that $f \geq g$. By the internal independence of the reference set $E$ one has $f \sqsupseteq^E g$ if and only if $f \geq g$. Then, it also cannot be the case that $f \sqsupseteq^E g$. As a consequence, there exists $x \in E$ which includes $f$ and does not include $g$. Analogously, there exists $y \in E$ which includes $g$ and does not include $f$. Thus, again we have that $f, g$ are mutually independent.

We conclude that problem $(E, F)$ exhibits the simple internal independence property. □

**Strong Independence of a Decision Problem.** Let us now come to the strong internal independence of a decision problem. We know that $(E, F)$ is internally independent as soon as the strings in its reduced logogram $|Log_E(F)|$ are mutually independent taken two by two.

The simple internal independence of a decision problem $(E, F)$ certainly is a form of internal independence of a decision problem, but we may indeed ask for more independence: We may ask for independence of the elements of the reduced kernel $|Log_E(F)|$ taken $m$ by $m$ all $m$. The following notion of internal independence of a decision problem captures this extreme form of

internal independence of a problem.

We shall say that the decision problem $(E, F)$ exhibits the property of *strong internal independence* if, for any choice of $s$ distinct strings $f_1, .., f_s$ in $|Log_E(F)|$, the following is true: For every $i$ between 1 and $s$ there exists a word $x_i \in E$ such that $x_i$ contains $f_i$ and fails to contain any of the remaining strings in $\{f_1, .., f_s\}$. It is left for the reader to show that strong internal independence of a decision problem implies simple internal independence.

## 6. Witnesses and Wizards

In principle, we cannot exclude that the reduced logogram $|Log_E(F)|$ of a problem $(E, F)$ may contain strings that behave as collective witnesses, also called wizards. There are problems where $|Log_E(F)|$ has wizards, e.g. *COMPOSITES* (Note, incidentally, that *COMPOSITES* is in $P$, Agrawal et al [1]).

If there are wizards in the logogram, a program $P$ solving $(E, F)$ might do calculations that are functionally equivalent to testing input $x$ for possible inclusion of wizards: We expect that that would make the task of deciding about the input easier.

Assume that we are considering an input word $x$ of size $n$. Let $g$ be a string in the reduced logogram $|Log_E(F)|$ of an NP problem $(E, F)$ so that $Exp_E(g)$ is a subset of the target set $F$. We assume that the size of string $g$ is less or equal to $n$, which is written $|g| \leq n$, thus $g$ actually belongs in $|Log_E(F^n)|$.

Let us remember that, given any NP problem $(E, F)$, there is a decomposition of the target set $F$ into subsets $F_i$ called solution regions, where $F_i$ is the set of those words $x$ in $E$ that are satisfied by solution $y_i$.

That given, we say that $g$ is a witness as soon as its associated relativized cylinder $Exp_E(g)$ is fully included in at least one of the $F_i$s. We say that $g$ is a proper witness as soon as its associated relativized cylinder $Exp_E(g)$ is fully included in exactly one of the $F_i$s. Equivalently, $g$ is a single-valued witness as soon as $g$ is an encoded sign of satisfiability that points toward a unique solution $y_k$. If $Exp_E(g)$ is included in the intersection of two or more of the $F_i$s then $g$ is a multiple-valued witness (also called a pseudowizard).

We say that $g$ is a wizard as soon as $Exp_E(g)$ is included in $F$ but fails to be fully included in a solution region. Thus, when $g$ is a wizard, its inclusion in a word $x$ belonging to the reference set $E$ is a guarantee that $x$ is satisfiable while nothing can be said about what particular solutions satisfy $x$.

**Completeness of the Set of the Witnesses.** From equation (8) we have

$$Log_E A_1 \cup .. \cup Log_E A_m \subseteq Log_E(A_1 \cup .. \cup A_m) \tag{17}$$

for closed $A_1, .., A_m \subseteq E$. Now replace $m$ with $\alpha(n)$ and $A_i$ with $F_i$ :

$$Log_E F_1 \cup .. \cup Log_E F_{\alpha(n)} \subseteq Log_E(F_1 \cup .. \cup F_{\alpha(n)}). \tag{18}$$

The strings in $Log_E F_1, .., Log_E F_{\alpha(n)}$ are witnesses. The strings in

$$Log_E(F_1 \cup .. \cup F_{\alpha(n)}) - Log_E F_1 \cup .. \cup Log_E F_{\alpha(n)}, \tag{19}$$

if there are any of them, are exactly the strings that we called wizards a few lines above. We call them "wizards" since they are so to speak able to witness that an input $x$ shall be in someone of the $F_i s$ but could not say which. The possible existence of this type of strings in the reduced logogram $|Log_E(F)|$ of a decision problem $(E, F)$ can be demonstrated by examples. Wizards have been found to exist in the reduced logograms of following problems: To decide if a symmetric loopfree graph is connected, and, as already mentioned above, to decide if a given positive integer is composite.

In a situation in which the target set $F$ is decomposed according to $\cup_n F_n = F$, the witnesses are always there in the reduced logogram of set $F$ relative to $E$. On the contrary, the wizards may be missing. It pertains to the structure of the computational problem at hand whether the target set $F$ has wizards. We conclude this section proving a theorem:

**Theorem 7.** *Let $(E, F)$ be an NP problem with $F_1$,.., $F_{\alpha(n)}$ as solution regions for the instances of size $n$. The set $\cup_{i=1}^{\alpha(n)} |Log_E(F_i)|$ of all the witnesses that occur in the reduced logogram $|Log_E(F)|$ is complete for problem $(E, F)$.*

*Proof.* We will confine to inputs of size $n$.

Since the solution regions $F_i$ have been taken to be cylinders, we may introduce a reduced logogram $|Log_E(F_i)|$ associated with each problem $(E, F_i)$ for every $i$, $1 \leq i \leq \alpha(n)$.

It is clear that, for any integer $i$, all of the strings in $|Log_E(F_i)|$ are witnesses of problem $(E, F)$. It is also obvious that there are no other witnesses for $(E, F)$, hence the set of all the witnesses of problem $(E, F)$ is $witSet_E(F) = \cup_{i=1}^{\alpha(n)} |Log_E(F_i)|$.

All that granted, we must prove that the set $witSet_E(F)$ is a complete subset of the full reduced logogram $|Log_E(F)|$ of problem $(E, F)$. This is to say that it is left for us to prove that, given any $x \in F$, there exists at least one string $g$ from $witSet_E(F)$ such that $g \leq x$.

Since $x$ is in $F$, and $F = \cup_j F_j$, there exists $i$ such that $x \in F_i$.

Now we know that if an instance $x$ is satisfied by a solution $y_i$ then $x$ subsumes at least one string $g$ such that the relativized cylinder $Exp_E(g)$ associated with $g$ is fully included in $F_i$. Clearly, $g \in |Log_E F_i|$. Then we also have $g \in |Log_E(F_1)| \cup .. \cup |Log_E(F_{\alpha(n)})| = witSet_E(F)$. □

## 7. Application to Boolean Formulas

Much recent work on $SAT$ and problems related to $SAT$ is focused on the search for new structural properties of Boolean satisfiability. This application is to be positioned in this stream of renewed interest in the structure of $SAT$. Significant papers in this research direction include Kirousis, Kolaitis [12], Allender et al [2], Gopalan et al [9]. In the rest of the paper we develop an application to Boolean formulas that relies upon the property that we called strong internal independence.

Given any Boolean formula $\gamma$, its complexity $|\gamma|$ is the number of distinct Boolean variables that have occurrences in $\gamma$. We take $x_1$, $x_2$,.. to be the variables that occur in the formulas that we consider. If a formula contains exactly $n$ distinct variables, we assume that these are $x_1, .., x_n$.

A Boolean formula in conjunctive normal form is a conjunction $A \wedge B \wedge .. \wedge C$ of clauses $A, B, .., C$. A clause in turn is a disjunction $u \vee v \vee .. \vee w$ of literals $u, v, .., w$, where a literal is either a Boolean variable or the negation of a Boolean variable. A clause is simple if it is not the case that one and the same variable occurs in it more than once. Thus, $x \vee y$ is simple while $x \vee x$ and $x \vee (-x)$ are not. We shall identify Boolean formulas in CNF with sets of simple clauses.

We now introduce the encoding scheme that we adopt for formulas in CNF format. The scheme converts Boolean formulas in CNF into words over the problem alphabet $\Sigma = \{0, 1, 2\}$. The set of words which encode Boolean formulas is denoted $E$ consistently with previous notations. The satisfiable subset of $E$ is denoted $F$.

We represent a clause by a sequence of $n$ symbols from $\Sigma$ that serve as variable indicators (one for each of the variables $x_1, .., x_n$ taken in this order). A variable indicator takes a value of 0 to denote absence of the variable, a value of 1 to denote presence without minus, a value of 2 to denote presence with minus. Thus, for $n = 3$, 012 means $x_2 \vee -x_3$. A whole formula is encoded as a sequence of clauses: In all, a word of length $nm$ ($n$ number of variables, $m$ number of clauses). In our scheme the encoded formula is preceded by a prefix of the form 0..010..01 consisting of $n$ 0s followed by a 1 followed by $m$ 0s

followed by a 1. A program $P$ that solves $(E, F)$ learns the values of $n$ and $m$ from this prefix.

The property of $SAT$ that is known as paddability occurs under this scheme as the fact that all characters beyond prefix plus formula are to be disregarded. Under this encoding scheme an input word consists of three segments: a prefix ($n+m+2$ characters), a formula ($nm$ characters), and a padding stream (any characters, any length). In presence of such an input word, a program $P$ takes the prefix and the formula, and ignores all that is beyond the formula.

When we speak of an encoded problem instance we understand prefix plus formula. The collection of all the encoded problem instances will be called *CNF*. Very evidently, *CNF* is a prefix-free language over the problem alphabet $\Sigma = \{0, 1, 2\}$.

In our encoding scheme, the set of all Boolean formulas in conjunctive normal form is thus encoded as a prefix-free language $CNF$ over the alphabet $\Sigma = \{0, 1, 2\}$. $SAT$ is the subset of $CNF$ consisting of the satisfiable formulas. The actual reference set $E$ is the absolute expansion of the prefix-free language *CNF*, thus $E = Exp(CNF)$. Analogously, the actual target set is $F = Exp(SAT)$.

We stress that $E$, $F$, and each of the solution regions $F_i$ all are absolute cylinders as required by the theory developed above.

Only the words over the alphabet $\Sigma = \{0, 1, 2\}$ which happen to begin with a syntactically correct encoded instance from $CNF$ are taken as relevant inputs by a decision program $P$ solving $SAT$. On a correct input, program $P$ takes under consideration the encoded problem instance and disregards everything that may follow.

Essentially, our application consists in investigating whether $|Log_E(F)|$ might possibly contain strings not already in some of the $|Log_E(F_i)|$.

Before we discuss the propositions that we were able to derive, let us spend a few words on the logogram of $SAT$. A string in $|Log_E(F^{nm})|$ is a prescription that a word in $F^{nm}$ may or may not be conformant with ($F^{nm}$ is the $n, m$ echelon of $F$). We may represent a string in $|Log_E(F^{nm})|$ as a word of length $nm$ over $\{\flat\} \cup \Sigma$. Example for $n = m = 3$: String $\flat\flat 11\flat 2\flat 2\flat$ prescribes that first clause shall include $x_3$, second shall include $x_1$ and $-x_3$, third shall include $-x_2$. Note that strings in $|Log_E(F^{nm})|$ only prescribe either 1 or 2 as values (by the minimality property of reduced logogram).

**Theorem 8.** *Problem $(E, F)$ where $E = Exp(CNF)$ and $F = Exp(SAT)$ exhibits the strong internal independence property.*

*Proof.* We consider $s$ distinct strings $f_1, .., f_s$ in $|Log_E(F^{nm})|$. Thus, regarded as a partial function, each $f_i$ will assign only values 1 or 2. We must prove that for each $i = 1, .., s$ there exists a word $x_i \in E$ such that $x_i$ includes $f_i$ and does not include any of the remaining strings $f_1, .., f_s$.

Let $i$ be any one of the index values $1, .., s$. Then $Dom(f_i) \subseteq \{1, .., nm\}$ and, for all $h \in Dom(f_i)$, we either have $f_i(h) = 1$ or $f_i(h) = 2$.

Let $x_i$ be that word of length $nm$ over $\Sigma = \{0, 1, 2\}$ such that for all $h \in Dom(f_i)$ it holds that $x_{ih} = f_i(h)$ while for $h$ not in $Dom(f_i)$ one has $x_{ih} = 0$. Then certainly $x_i$ subsumes $f_i$.

Let $f_j$ be any one of the strings $f_1, .., f_s$ being different from $f_i$. Thus, $f_j \neq f_i$. We must prove that $x_i$ does not include $f_j$.

(I) Assume $Dom(f_j) = Dom(f_i)$. Since $f_i$ and $f_j$ are different, there exists an argument value $k \in Dom(f_i)$ such that $f_i(k) \neq f_j(k)$.

But $x_{ik} = f_i(k)$, then $x_{ik} \neq f_j(k)$. Then $x_i$ does not include $f_j$.

(II) Assume $Dom(f_j) \neq Dom(f_i)$. Then either there is $a \in Dom(f_j)$ such that $a \notin Dom(f_i)$ or there exists $b \in Dom(f_i)$ such that $b \notin Dom(f_j)$.

Assume that $a$ exists. Then $x_i$ does not include $f_j$ since $x_{ia} = 0$ while $f_j(a) \neq 0$, hence $x_{ia} \neq f_j(a)$. Analogously, $x_i$ does not include $f_j$ in case $b$ exists. □

**Theorem 9.** *The reduced logogram $|Log_E(F)|$ where $E = Exp(CNF)$ and $F = Exp(SAT)$ does not contain wizards.*

*Proof.* We consider formulas with $n$ variables and $m$ clauses.

It is left for us to prove:

$$Log_E F_1 \cup .. \cup Log_E F_{\alpha(n)} = Log_E(F), \tag{20}$$

where $\alpha(n)$ is the cardinality function and $F_i$ is the range of the value assignment $y_i$ (set of formulas in $F$ that are satisfied by $y_i$). Since equation (18) holds, we just have to prove that the right-hand side of equation (20) is included in the left-hand side. We actually will prove:

$$|Log_E(F)| \subseteq |Log_E F_1| \cup .. \cup |Log_E F_{\alpha(n)}| \tag{21}$$

which is clearly equivalent. (In this proof we reason echelon by echelon: It should be clear that in equations (20), (21), and similar formulas in this proof, we actually address the $n, m$ echelon of the corresponding logograms.)

We write $K$ as short notation for $|Log_E(F)|$, thus $K$ stands for the reduced logogram of target set $F$ relative to base $E$. For every integer $i = 1, .., \alpha(n)$, we write $K_i = |Log_E(F_i)|$. Besides, we write $witset = K_1 \cup .. \cup K_{\alpha(n)}$, so that

*witset* is the set of all witnesses associated with the $n, m$ echelon of the reduced logogram $K$ (portion of $K$ associated with formulas with $n$ variables and $m$ clauses).

We must prove $K \subseteq witset$.

Let $h \in K$. Since $K$ is a subset of $\Sigma_\infty(E)$, we have $h \in \Sigma_\infty(E)$. Then, by equation (2), there exists an $x \in E$ with $n$ variables and $m$ clauses such that $x \geq h$. On the other side, if $x$ is a word in $E$ and $x$ subsumes $h$ as a substring, then $x \in F$, and then, since *witset* is complete for $F$ relative to $E$ by Theorem 7, there exists a string $k \in witset$ such that $x \geq k$.

We set $h \rightarrow k$ to mean that (i) $k$ is a member of *witset*, (ii) there exists $x \in E$ such that both $h \leq x$ and $k \leq x$. When $h \rightarrow k$ we say that $k$ is a witness related to $h$ (note that $h \rightarrow k$ implies that $h$ and $k$ are compatible).

Besides, we introduce the set $witset(h) = \{k \in witset : h \rightarrow k\}$ of those witnesses that are related to string $h$. The string set $witset(h)$ may be called the witset associated with $h$. By way of contradiction, we assume that string $h$ does not belong to $witset(h)$.

We then have that the elements in the set $\{h\} \cup witset(h)$ are all distinct.

Now we observe that all the strings in $\{h\} \cup witset(h)$ are members of the reduced logogram $|Log_E(F)|$. On another side, by Theorem 8, $SAT$ exhibits the strong internal independence property.

As consequence, in correspondence to each string $f \in \{h\} \cup witset(h)$ there exists a word $x \in E$ such that $f \leq x$ and for no $g \in \{h\} \cup witset(h)$ being distinct from $f$ one has $g \leq x$.

Let $x \in E$ be such that $x \geq h$ and for no $g \in witset(h)$ one has the inclusion $g \leq x$. Word $x$ is in $F$ since $x$ includes $h$ which is an element of $|Log_E(F)|$. Besides, $x$ does not contain any element from $witset(h)$. But that in turn means that $x$ does not contain any strings from *witset*. (Should $x$ include a string $k$ from *witset* that would mean that both $x \geq h, x \geq k$ would hold, hence string $k$ would be related with $h$, which in turn would imply $k \in witset(h)$.)

This is absurd, since *witset* is complete for $SAT$ relative to $CNF$ by Theorem 7. We conclude that $h$ is a member of $witset(h)$, and hence is in *witset*. Since $h$ was chosen to be any member of $K$, we conclude that $K$ is a subset of *witset*. Since we already know that *witset* is a subset of $K$, we conclude that $K = witset = K_1 \cup .. \cup K_{\alpha(n)}$. Thus, $SAT$ has no wizards. □

**Theorem 10.** *The reduced logogram* $|Log_E(F)|$ *where* $E = Exp(CNF)$ *and* $F = Exp(SAT)$ *is irreducible.*

*Proof.* Let $g \in |Log_E(F)|$. In this proof we consider formulas with $n$ variables and $m$ clauses. (In this proof we still reason echelon by echelon: When we speak of the logogram $|Log_E(F)|$ it goes without saying that we actually refer to the $n, m$ echelon of the reduced logogram.)

By Theorem 9 we know that $g$ must be a witness.

Thus, $g$ is a string carrying the explicit specification of one or more value assignments (all of which satisfy the formula that hosts $g$).

Since $g$ is a member of the reduced logogram $|Log_E(F)|$, then $g$ is a minimal decision string (no proper restriction of $g$, when included in an instance $x$, is sufficient condition for the event $x \in F$ to occur).

String $g$ prescribes the presence of a well defined literal to each of the clauses of a formula.

The above remarks make it a straightforward task to specify the general shape that string $g$ shall exhibit.

First of all, $Dom(g)$ shall have to be a set of exactly $m$ numbers taken from $\{1, .., nm\}$. The first of these numbers is to be taken from the first block $\{1, .., n\}$ (where the first clause is allocated), the second is to be from the second block $\{n+1, .., 2n\}$,.., the $m$-th is from the $m$-th block $\{n(m-1)+1, .., nm\}$ (where the last clause is allocated). Thus, there are $nm$ possible determinations for $Dom(g)$. We know that regarded as a prescription, $g$ can only prescribe the two values 1 and 2. (To help intuition, string $g$ can be thought of as a sequence of flats $\flat\flat..\flat$ of length $nm$ in which some of the flats (as many as $m$) have been replaced with $1s$ or $2s$.)

With any $g$ that satisfies the above requirements we associate a formula $\gamma(g)$ as follows. We note that, regarded as a prescription, $g$ prescribes the presence of exactly one literal in each clause of a formula $x$ consisting of $m$ clauses: We then state that the $i$-th clause of $\gamma(g)$ shall consist of exactly the single literal that $g$ prescribes to the $i$-th clause of $x$.

Evidently, $\gamma(g)$ is satisfiable and $g \leq \gamma(g)$. We claim that $\gamma(g)$ does not include members of $|Log_E(F)|$ other than $g$.

Indeed, the strings in $|Log_E(F)|$ never prescribe 0 as value, and $g$ is the largest string being included in $\gamma(g)$ which does not prescribe 0 as value. Thus, the only strings that do not prescribe 0 as value and happen to be included in $\gamma(g)$ are exactly string $g$ itself and the proper restrictions of string $g$. Since $g$ is minimal, all of its proper restrictions are not members of $|Log_E(F)|$. Thus $g$ is the only string being included in word $\gamma(g)$ to be found in $|Log_E(F)|$.

Hence, $|Log_E(F)| - \{g\}$ is not complete for $F$ relative to $E$. □

## 8. Conclusions

We advocated strings (with special meaning for the term) as a fundamental notion for studies of computation. So to speak, strings are needed to express the notions of internal and strong internal independence of a decision problem that underly our theory of decision problems. We were led to formulate strings to become able to derive the very basic notion of internal independence of a decision problem. Strings seem to be useful since they are absolutely elementary. Note that they are already at work in complexity: The "restrictions" that are used in the study of circuit complexity are strings, see Hemaspaandra and Ogihara [11].

Strings are not made of consecutive letters. A string can be interspersed in a word: By canceling zero or more letters in a word $x$, and by leaving blanks in places of letters, we get a string $f$ which is a substring of the original word $x$. In a string, one has information associated with spaces between letters (and hence with possible multiple periodicity with which letters may occur). As soon as we have the strings, we are able to define the kernel $Ker(P)$ of a decision program $P$, a set of strings which capture structural features of both program $P$ and the decision problem $(E, F)$ that $P$ solves.

$Ker(P)$ is a subset of the reduced logogram $|Log_E(F)|$ of target set $F$ in base $E$. The reduced logogram consists of substrings of the words in $F$ which exhibit the following property: If a word in $E$ includes one of these substrings then it belongs to $F$. We may think of the strings in $|Log_E(F)|$ as kind of genes of the words in $F$. (In early notes the logogram was the *jinnee* or *genie* of problem $(E, F)$.) The idea clearly comes from biology, where it is known that certain occurrences at given intervals of certain letters within DNA sequences convey structural information, and yield observable characters in the macroscopic development of the structures.

Our application to $SAT$ uses a structural property of that problem that seems to have escaped attention so far. We called it "strong internal independence." Theorem 8 shows that $SAT$ exhibits the strong internal independence property. Theorem 9 shows that, by that property, $SAT$ cannot have collective certificates in its reduced logogram. As a consequence, all the programs that solve $SAT$ have same kernel (Theorem 10).

The theory presented in this paper has applications to various fields of computer science. In these conclusions we mention the following one.

DNA strands convey information about how an individual is expected to be as it grows. Where and how is that information recorded? The above

theory gives a precise answer to this question: The information is conveyed by substrings (it is associated with presence or absence of certain substrings). Thus, to understand what strings happen to be associated to, say, hare-lip in the humans, we shall have to put at work the appropriate algorithms (likely of a randomized character) that will give us convenient approximations of the logogram of $F$ to base $E$ where $E$=humans, $F$=humans exhibiting hare-lip. It may be that hare-lip is not of utmost interest, but our intent here is to just call attention to an application of major importance.

## Acknowledgments

I am grateful to Professors Fabrizio Luccio (computer scientist), Umberto Mosco, Claudio Procesi (mathematicians) for their advice and encouragement.

## References

[1] M. Agrawal, N. Kayal, N. Saxena, Primes is in $P$, *Annals of Mathematics*, **160** (2004), 781-793.

[2] E. Allender, M. Bauland, N. Immerman, H. Schnoor, H. Vollmer, The complexity of satisfiability problems: Refining Schaefer's Theorem, *Journal of Computer and System Sciences*, **75** (2009), 245-254.

[3] J. Balcazar, J. Diaz, J. Gabarro, *Structural Complexity, II*, Springer (1990).

[4] G. Birkhoff, *Lattice Theory*, Volume 25, AMS.

[5] S. Di Zenzo, P. Bottoni, P. Mussio, A notion of information related to computation, *Information Processing Letters*, **64** (1997), 207-215.

[6] R. Fagin, J.Y. Halpern, Y. Moses, M.Y. Vardi, *Reasoning about Knowledge*, The MIT Press (2003).

[7] R. Fagin, J.H. Halpern, M.Y. Vardi, A model-theoretic analysis of knowledge, *Journal of the ACM*, **38** (1991), 382-428.

[8] G. Gierz, K. Hofmann, K. Keimel, J.D. Lawson, M.W. Mislove, D. Scott, *Continuous Lattices and Domains*, Cambridge University Press.

[9] P. Gopalan, P.G. Kolaitis, E.N. Maneva, C.H. Papadimitriou, The connectivity of Boolean satisfiability: Computational and structural dichotomies, *SIAM J. Comput.*, **38**, No. 6 (2009), 2330-2355.

[10] J.Y. Halpern, *Reasoning about Uncertainty*, The MIT Press (2005).

[11] L.A. Hemaspaandra, M. Ogihara, *The Complexity Theory Companion*, Springer (2002).

[12] K. Kirousis, P. Kolaitis, The complexity of minimal satisfiability problems, *Information and Computation*, **187** (2003), 20-39.

[13] K.G. Larsen, G. Winskel, Using information systems to solve recursive domain equations, *Information and Computation*, **91** (1991), 232-258.

[14] P. Odifreddi, *Classical Recursion Theory*, North-Holland (1989).

[15] D. Scott, *Domains for Denotational Semantics*, ICALP82, Lecture notes in Computer Science, **140**, Springer (1982).