Int. J. Appl. Math. Comput. Sci., 2014, Vol. 24, No. 2, 283–297
DOI: 10.2478/amcs-2014-0021

SELECTION OF SEARCH STRATEGIES FOR SOLVING 3–SAT PROBLEMS

ANDRZEJ PUŁKA

Institute of Electronics
Silesian University of Technology, ul. Akademicka 16, 44-100 Gliwice, Poland
e-mail: Andrzej.Pulka@polsl.pl

The paper concerns the problem of Boolean satisfiability checking, which is recognized as one of the most important issues in the field of modern digital electronic system verification and design. The paper analyzes different strategies and scenarios of the proving process, and presents a modified and extended version of the author's FUDASAT algorithm. The original FUDASAT methodology is an intuitive approach that employs a commonsense reasoning methodology. The main objective of the work is to investigate the SAT-solving process and try to formulate a set of rules controlling the reasoning process of the FUDASAT inference engine. In comparison with the author's previous works, the paper introduces new mechanisms: hypergraph analysis, multiple variable assignments and search space pruning algorithms. The approach considers only 3-SAT class functions, although a generalization of the method is discussed as well. The presented approach has been tested on various benchmarks and compared with the original pure FUDASAT algorithm as well as with other algorithms known from the literature. Finally, the benefits of the proposed SAT solving technique are summarized.

Keywords: SAT solving, formal verification, CNF, Boolean satisfiability.

1. Introduction

The problem of Boolean SATisfiability (SAT) has been defined since the beginning of the existence of logical circuits and now is one of the most studied issues in the field of combinatorial search and minimization as well as artificial intelligence. Davis and Putnam (1960) proposed the first formal approach to SAT problems. They developed an algorithm for checking the validity of first-order logic formulas using a resolution-based decision procedure for propositional logic. In general, the original DP (the acronym for the authors' initials) algorithm consists of three main steps: transformation of the original formula into a special form (in particular, CNF) and elimination of all quantifiers, generation of all propositional ground instances (one by one), and checking the satisfiability of all instances.

The DP algorithm appeared not to be very effective and a modified version of the first approach, known as the DPLL (Davis–Putnam–Logemann–Loveland) algorithm (Davis *et al.*, 1962) became, in fact, a first complete, backtracking-based algorithm for deciding about the satisfiability of propositional logic formulae expressed in CNF, i.e., for solving the CNF-SAT problem. Since then many mathematicians as well as engineers from all over the world proposed various techniques in the field of SAT (Suyama *et al.*, 1999; Marques-Silva and Sakallah, 1999; Han *et al.*, 2010; Moskewicz *et al.*, 2001; Hu *et al.*, 2008; Yin *et al.*, 2012). SAT solvers are commonly used for testing in automated test patterns generators, and many modern Electronic Design Automation (EDA) tools contains appropriate SAT packages.

The work presented in the paper proposes some modifications to existing approaches; in particular, it introduces a new original SAT-solving algorithm supported with optimal selection of variables and backtrack search controlling mechanisms based on the technique borrowed from AI, i.e., Fuzzy Default Logic (FDL). After a short discussion of related works in Section 2, the author points out the parallelism between the non-monotonic reasoning and Boolean satisfiability solving (Section 3). The FDL mechanism is briefly recalled in Section 4. Section 5 presents the basic stages of the original FUDASAT algorithm (Pułka, 2011), which is the starting point for the discussion on the searching strategies. Section 6 is the main part of the paper. It analyzes the searching strategies and introduces the methodology of pruning the searching space.

Section 7 summarizes the paper with concluding remarks. All presented methodologies are exemplified on various functions (problems) given in Appendices.

2. Boolean satisfiability problem formulation

One can express the problem of Boolean function satisfiability as the verification and/or the mathematical task that answers the question *if there exists (or does not exist) an assignment for a given function's variables for which this function is satisfied (i.e., this function gives truth)*.

2.1. Standard specification of SAT problems. The most convenient form of the description of the SAT problems is the standard Conjunctive Normal Form (CNF), i.e., a conjunction of clauses C_i, where each clause is given as a disjunction of literals. Each literal comprises the elementary logical unit of a given Boolean function F (problem), being merely an instance of a variable or its complement. Formally, for a given Boolean function $F(x_1, \ldots, x_N)$ of variables $X = \{x_1, \neg x_1, \ldots, x_N, \neg x_N\}$ (where the symbol $\neg x_i$ denotes the complemented variable), the CNF form of the function F is

$$F(x_1, \neg x_1, \ldots, x_N, \neg x_N) = C_1 \wedge C_2 \wedge \cdots \wedge C_M. \quad (1)$$

This form (CNF) of problem representation is very clear, because in order to prove the satisfiability of the function we have to satisfy all clauses included within the formula (function). Very often SAT problems are restricted to instances where all clauses have the same length k. Such problems are denoted by *k-SAT*:

$$F^{k-\mathrm{SAT}}(x_1, \neg x_1, \ldots, x_N, \neg x_N) = C_1 \wedge \cdots \wedge C_M, \quad (2)$$

where every clause C_i is built of k variables (literals):

$$C_i = x_{i1} \vee x_{i2} \vee \cdots \vee \neg x_{ik-1} \vee \neg x_{ik}, \quad (3)$$

and

$$\{x_{i1}, \ldots \neg x_{ik-1}, \neg x_{ik}\} \subset \{x_1, \neg x_1, \ldots, x_N, \neg x_N\}. \quad (4)$$

It could be proved (and many experiments in the field confirm it) that the smallest value of k for which the k-SAT problem is NP-complete is three. This fact justifies why 2-SAT and 3-SAT tasks are the most thoroughly investigated problems in the field.

The former search algorithms for SAT problems could be divided into static and dynamic ones. The former are based on the predetermined (fixed) order of the variables, while in the latter the order of variables dynamically changes during the runtime. Moreover, static algorithms are usually simple, but ineffective and slower than dynamic search algorithms, which usually require heuristic approaches.

Generally, we can distinguish several problems occurring during the solving process: the appropriate order of variables selection, conflict analysis and registration process as well as the backtrack search procedure in the case of a conflict. Very popular are approaches to variable selection based on the MOM (Maximum number of Occurrences in the Minimum length clauses) philosophy (Suyama *et al.*, 1999). This methodology is intuitive, because we can simply explain that variables constituting the shortest clauses are the most constrained literals of the entire formula (function). There exist at least several versions of the MOM idea (Suyama *et al.*, 1999), but many works (Marques-Silva and Sakallah, 1999; Tille *et al.*, 2010; Han *et al.*, 2010) show that there is no one universal methodology of the variables' order selection. The Chaff technique (Moskewicz *et al.*, 2001) uses a very interesting decision heuristic, the VSIDS (Variable State Independent Decaying Sum), which ranks variables by literal counts and periodically divides all counts by an empirical constant, and during the backtrack process (forced by the conflict) another unassigned variable is taken from the stack.

The GRASP technique (Marques-Silva and Sakallah, 1999) proposes an implication graph, in which every variable has a node (similar to the technique based on binary decision diagrams (Aloul *et al.*, 2002)). The implication graph keeps track of the values assigned to variables and the dependencies of these assignments, i.e., we can easy explain why a given variable has been forced to true or false. The GRASP algorithm also records conflict clauses and uses non-chronological backtracking. Recently, Yin *et al.* (2012) presented some interesting ideas improving the effectiveness of the deductions trees by examination of the resolution process during elimination of variables. As a result, implication graphs could be improved.

3. Background of the approach

The author, in his previous works concerning SAT problems (Pułka, 2011), pointed out some common properties of nonmonotonic reasoning and Boolean satisfiability problems. Let us briefly recall them.

(i) *Satisfiability*: looking for an assignment that satisfies a given logical function (in SAT) can be compared to searching for the extension of a given non-monotonic theory (Brewka, 1991; Reiter, 1980).

(ii) *Heuristic*: both techniques are based on non-standard inference schemes (experience), and both problems can be classified as data mining and information reduction issues, i.e., searching for unrecognized relationships between elements hidden inside the database.

(iii) *Selection of variable ordering*: this property corresponds to the problem of the variable assignment

order in SAT solving and the preference operator proposed in Answer Set Programming (ASP) (Balduccini *et al.*, 2006) based on disjunction logic.

(iv) *Variable and conflict clauses recording*: the analog operation is also present in non-monotonic reasoning. Brewka (1991) discussed the problem of consistency and stability of extensions, which means that if we want to have a stable, inconsistent system, we have to store the assumed hypotheses. This property is sometimes called cumulativity.

(v) *Backtracking and conflict solving*: In non-monotonic logic, this process is called revision of beliefs. It occurs when it is necessary to remove some previously assumed hypotheses, which are inconsistent with a new piece of information delivered to the system.

The inference engine of the FUDASAT algorithm is based on Fuzzy Default Logic (FDL)—a mechanism that combines the cumulative version of Reiter's default logic (Reiter, 1980) with Zadeh's Generalized Theory of Uncertainty (GTU) (Zadeh, 2006). A detailed description of the FDL formalism is given by Pułka (2009).

FDL introduces Fuzzy Default Rules (FDRs) (modified original Reiter defaults (Reiter, 1980)) in the following form:

$$\frac{\alpha : \beta_1, \beta_2, \ldots, \beta_N}{\Phi^{\lambda}}, \tag{5}$$

where $\alpha, \beta_1, \beta_2, \ldots, \beta_N$ $(i = 1, \ldots, N)$ are wffs (well formulated formulas) in a given propositional language L, and Φ^{λ} is a Fuzzy Hypothesis (FH):

$$\Phi^{\lambda} = \left\{ \left[h_1^{\lambda}, Tw\left(h_1^{\lambda}\right) \right], \ldots, \left[h_m^{\lambda}, Tw\left(h_m^{\lambda}\right) \right] \right\}, \tag{6}$$

where h_i^{λ} $(i = 1, \ldots, m)$ are wffs in a given propositional language L, and $Tw\left(h_i^{\lambda}\right)$ denotes trustworthiness, i.e., one of the modality of generalized constraints in the Zadeh sense (Zadeh, 2006) (bivalent, probabilistic, fuzzy, veristic, etc.). For the simplest case, trustworthiness can be treated as a membership function or probability (Zadeh, 2008). In the application to the SAT technique, a hypothesis h_i^{λ} corresponds to a single variable of the Boolean function. Additionally, we assume that a prerequisite α (like in the work of Reiter (1980)) is represented by strong information (facts in the sense of Gelfond and Lifschitz (1988)), while the possible uncertainty or missing information is represented by justifications $\beta_1, \beta_2, \ldots, \beta_N$. Nonmonotonicity is present thanks to two assumptions: NaF (Negation as a Failure) and CWA (Closed World Assumption). This scheme reduces the problem of inference path propagation and tracing for trustworthiness. If we would like to have an FDR based fully on ignorance and/or vagueness, the prerequisite is an empty set ($\alpha \equiv \oslash$).

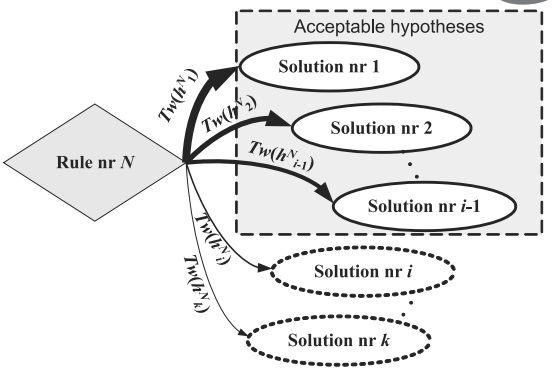

Fig. 1. Granular view of hypotheses generation from the FDR.

The interpretation of the FDR is very similar to the standard DL rule except for the form of the hypothesis (FH), which consists of different aspects (views, extensions) of the same problem, and each of these sub-hypothesis has its own Tw coefficient reflecting the significance of the given solution. This significance is usually subjective and can be modified. Elements of a given FH Φ^{λ}, i.e., $h_1^{\lambda},\ h_2^{\lambda}, \ldots, h_m^{\lambda}$, are mutually exclusive. At a first glance it looks like inconsistency, because we would like to derive different hypotheses about the same world, but we should remember that each of them has its own trustworthiness level, and moreover, it is the preprocessing phase before the final assessment of hypotheses. To eliminate some very weak hypotheses, we can add an additional cut-off mechanism, which preserves inferring a hypothesis with a small level of trustworthiness. Such a solution of the inference engine simplifies hypothesis derivation, and the problem of priorities and existence of various extensions of default theories (Reiter, 1980; Brewka, 1991) does not limit the application. During the inference process, it is possible to observe various options (debug the inference engine), and their number can be controlled by the cut-off level. A granular representation of the FDR reasoning procedure is presented in Fig. 1 (the width of the arrows corresponds to the values of trustworthiness). Selection of the final hypothesis requires introduction of two additional operations necessary for hypotheses assessment (Pułka, 2009), namely, the Credible Set (CS) and Hypotheses Reduction (HR).

The credible set H^{λ} of a given fuzzy hypothesis Φ^{λ} is a subset of Φ^{λ} consisting of elements h_j^{λ} which have appropriate (acceptable according to selected criteria) trustworthiness, i.e.,

$$H^{\lambda} \subseteq \Phi^{\lambda} \wedge \quad \forall_{h_j^1 \lambda \in H^1 \lambda} \quad Tw\left(h_j^{\lambda}\right) \geq cut\_off. \tag{7}$$

Intuitively, we can say that the CS corresponds to those hypotheses that can be considered during the further inferring process. Certainly, the presented

mechanisms of their selection may be more complicated or set up dynamically according to other constraints. The trustworthiness of hypotheses and its ordering corresponds to the ordering of literals in the head of the disjunction rules in answer set programming (literals l_0 to l_k from the expression (4)) and preference rules (Gelfond and Lifschitz, 1988; Balduccini *et al.*, 2006; Lukasiewicz and Straccia, 2008).

The assessment of hypotheses and selection of the final solution require a mechanism allowing comparison of hypotheses, so we need to analyze the credible sets and reduce all values of trustworthiness to only one for every hypothesis. This operation is called hypothesis reduction, i.e.,

$$\begin{aligned} HR\Big\{\bigcup_i H_i^{\lambda}\Big\} \\ = \Big\{\big[h_i^{\lambda}, Tw\big(h_i^{\lambda}\big)\big]\Big|\exists H_k^{\lambda_k} : h_i^{\lambda} \in H_k^{\lambda_k} \\ \wedge\, Tw\big(h_i^{\lambda}\big) = opt\big(Tw\big(h_i^{\sum \lambda_k}\big)\big)\Big\}, \end{aligned} \tag{8}$$

where $opt\big(Tw\big(h_i^{\sum \lambda_k}\big)\big)$ denotes the optimal value of trustworthiness for a given element (hypothesis) selected from all credible sets.

The problem of proper selection of the opt function is very interesting and it can be a field for user interaction to the inferring process. The optimal function can be flexible, i.e., it can have different meanings for different kinds of trustworthiness (bivalent, probabilistic, veristic, fuzzy). An optimal function can be also rigid (the same for every trustworthiness), which means that it corresponds to one of the following cases: maximum (optimistic approach), mean (no priorities), minimum (worst case analysis), max-min (fuzzy approach), etc. An example of FDR rule application and hypotheses (variables) assessment is presented in the last subsection of this section, while the details of the implementation as well as the appropriate algorithms can be found in our earlier work (Pułka, 2009).

Taking into account realistic problems, i.e., Boolean functions consisting of hundreds or even thousands of variables, we need to find more than one hypothesis (variable). That is why the entire process must be repeated, and we can show it as a multistage procedure.

Hypotheses can be generated directly from fuzzy default rules (as their hypotheses), and then we call them hypotheses of the zero level or we can deduce given information based on the hypothesis which is a result of a non-monotonic inference process (Reiter, 1980; Brewka, 1991). The latter also has to be classified as a non-monotonic hypothesis, because it is based not on a strong fact but on another non-monotonic hypothesis and can be invalidated later. So each hypothesis has to remember its predecessor. Because of this deduction structure, we can call the inference engine multilevel. After the revision of beliefs the system is ready to check the completeness of the basic extension and make any necessary supplements, to have a full set of the design information.

Consequently, the generated hypotheses form a kind of deductive chain, and a given assumed hypothesis 'remembers' its ancestor $Source_i$. The final assessment and selection of the best hypothesis as a final conclusion can be based on various schemes which depend on chosen demands: we can take a simple criterion of the trustworthiness value (like verity in veristic modality of generalized constraints), analyze the entire path (paths) from $Source_0$ to $Source_i$, and find the global trustworthiness (regarding it as probabilities or possibilities), or use fuzzy max(min) criteria (Zadeh, 2006). The following example shows that the assessment mechanism gives an additional ability to control the model (selected extension). The example concerns variable selection in a SAT-solving procedure.

Example 1. Let us assume that at the $\lambda - 1$ level the FUDASAT algorithm (described in the following section) has selected variable x_k, and the sequence of previous assignments (from the first level to the level of number $\lambda - 1$) indicates that the possibilities (values of trustworthiness) of the selections for variables x_1, x_2 and x_3 are $Tw(x_1) = 0.2$; $Tw(x_2) = 0.3$ and $Tw(x_3) = 0.19$, respectively. Moreover, from the other two FDR (static set), the system is able to infer the hypotheses (variables) with the following trustworthiness values: $Tw'(x_1) = 0.33$, $Tw"(x_1) = 0.22$, $Tw'(x_2) = 0.35$, $Tw"(x_2) = 0.16$ and $Tw'(x_3) = 0.4$, $Tw"(x_1) = 0.11$ (Fig. 2). So, we have three credible sets (Pułka, 2009) for variables x_1, x_2 and x_3: $CS(x_1) = 0.2, 0.33, 0.22$, $CS(x_2) = 0.3, 0.35, 0.16$ and $CS(x_3) = 0.19, 0.4, 0.11$.

The final choice depends on the assessment criteria. If the maximal possible value is taken into account, variable x_3 (with the highest value 0.4) is the best choice. In the case of the mean value we should take the variable x_2 (with the mean value 0.27), and in the worst case (max-min) analysis suggests the variable $x - 1$. It is also possible to neglect other rules and follow only the hypotheses generated by the dynamic set (here the best choice is x_2). The assessment of the hypothesis in generalized fuzzy default logic gives a lot of flexibility to

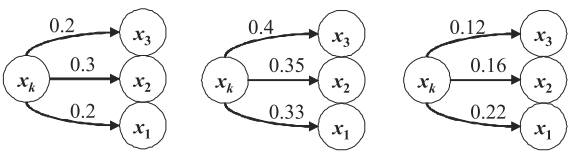

Fig. 2. Example of different possibilities of selecting the next variable.

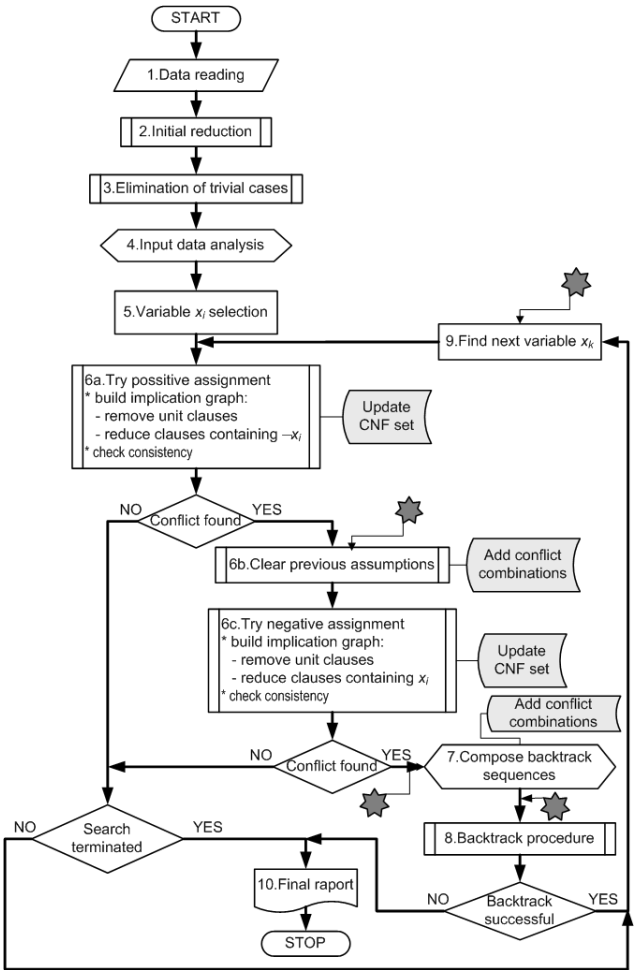

Fig. 3. Block diagram of the basic (FUDASAT) algorithm.

the user and is application dependent. Moreover, we can split the HR process into two steps: the first, responsible for the selection of the optimal value of trustworthiness for each hypothesis, and the next, making the final assessment and selecting only one ('the best') solution. Moreover, we can use different criteria for each step (level). ♦

4. Basic FUDASAT algorithm

The author proposed an approach based on fuzzy default logic (Pułka, 2011) and called it the FUDASAT algorithm. It uses the parallelism between non-monotonic reasoning and logical satisfiability problems, a set of sophisticated reasoning rules which aid the searching process, and this justifies the name of the algorithm (an acronym for FUzzy Default logic based Algorithm for SAT checking). The block diagram, depicted in Fig. 3, presents the first original version of the algorithm (Pułka, 2011). Because the basic FUDASAT has already been described in detail by Pułka (2011), its main steps are just briefly mentioned.

The first three stages can be called the preprocessing phase of the algorithm. They eliminate possible redundancy and check if a trivial case occurs. It is assumed that the input data has a CNF form, and usually the initial function consists of at least several clauses, so the exhaustive proof of satisfiability of the function is impossible at such early stages of the procedure. Stage 4 runs a set of procedures estimating and analyzing the clauses. These procedures take into account various criteria and priority functions (Suyama *et al.*, 1999). As a result, the system selects the variable assignment order (Stage 5). Based on the results of Stage 4, Stage 5 selects the order of variable analysis. This order can be neglected or canceled when a conflict is detected and the knowledge-base is supplied with extra forbidden combinations responsible for conflicts, so these variable selections (assignments) will be called hypotheses.

Stage 6 performs a given assignment, function reduction (via set of binary constraints propagation steps) and consistency check. In the case of inconsistency of the reduced function, i.e., the existence of two or more mutually exclusive assignments (for example, $x_i = 1$ and $x_i = 0$), the algorithm initiates procedures composing backtrack scenarios (Stage 7). This stage was only indicated and casually analyzed in the basic FUDASAT algorithm (Pułka, 2011). The problem is addressed in detail in the next section.

When the backtrack sequence is ready, the algorithm performs the procedure of the revision of beliefs (Stage 8) and, depending on the backtrack result, the satisfiability proof is continued or is reported (Stages 9 and 10) (Fig. 3).

The example presented in Appendix A (based on the benchmarks (DIMACS, 1993)) describes an example of simple backtrack scenario generation.

The first experiments conducted with the implementation of non-standard inference engines to Boolean satisfiability problems have showed their usefulness (Pułka, 2011), so the author has decided to continue them and extend the set of procedures responsible for effective conflict analysis. The next section presents these issues.

5. Experiments for developing search strategies

The searching strategy described in the example from the previous section is not always optimal. The basic FUDASAT algorithm has to be supplied with efficient tools allowing pruning the searching space. The series of experiments showed some regularities and relationships between the average number of clauses and variables. Figure 4 presents a diagram that summarizes results obtained for the family of benchmark functions consisting of 50 variables. It shows the relationship between normalized (to the peak value) evaluation times and

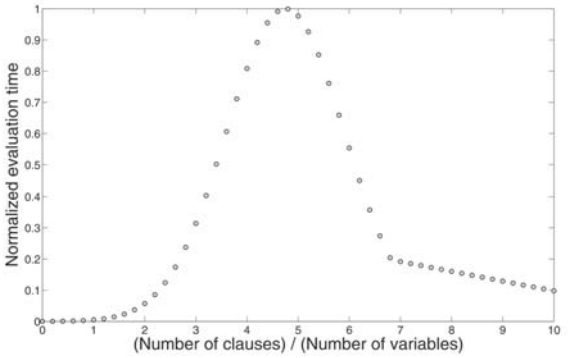

Fig. 4. Normalized evaluation times versus the number of clauses per variable.

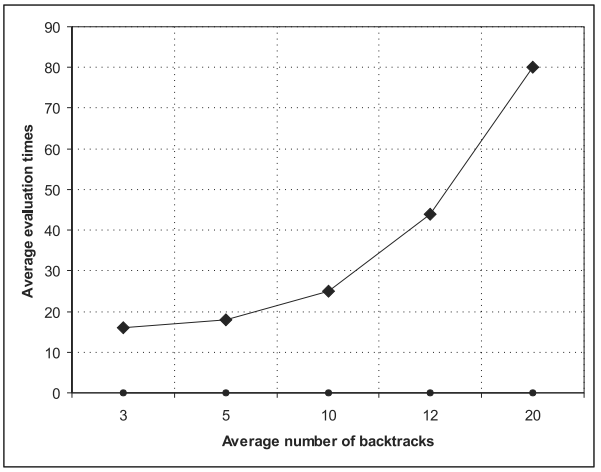

Fig. 5. Relationship between the average evaluation times and the number of backtracks obtained for a family of functions consisting of 50 variables.

the density of the function description expressed by a number of clauses per variable. A similar tendency could be observed for other cases and different benchmarks containing more variables as well as more clauses. Figure 5 shows that better results can be obtained by appropriate composing of the backtrack scenarios and minimizing their number. So the appropriate conflict analysis seems to be very important and in many cases decides about the success. The following point addresses this problem.

5.1. Conflict analysis. The mechanisms of conflict analysis introduced into the modified FUDASAT algorithm will be described on an example function borrowed from the DIMACS benchmarks (DIMACS, 1993).

Example 2. The initial function f consists of 160 clauses and has 60 variables. Each clause is built of 3 variables, so it is a classical 3-SAT class problem. Below are listed only those clauses which take part in the conflict presented on the hypergraph diagram (Fig. 6). The function $f = \bigcap_{i=1}^{160} C_i$ and some of its clauses are

$$\begin{aligned}
C_2 &= \neg x_1 \vee x_2 \vee x_3,\\
C_{17} &= \neg x_{13} \vee \neg x_{14} \vee \neg x_{15},\\
C_{21} &= \neg x_{16} \vee \neg x_{17} \vee \neg x_{18},\\
C_{50} &= \neg x_{37} \vee x_{38} \vee x_{39},\\
C_{56} &= x_{16} \vee \neg x_{37} \vee x_{40},\\
C_{57} &= \neg x_{15} \vee \neg x_{41} \vee \neg x_{42},\\
C_{63} &= x_{14} \vee x_{41} \vee \neg x_{43},\\
C_{66} &= \neg x_{10} \vee x_{18} \vee x_{44},\\
C_{69} &= \neg x_{13} \vee \neg x_{17} \vee \neg x_{44},\\
C_{148} &= x_3 \vee \neg x_{58} \vee x_{59},\\
C_{152} &= \neg x_{39} \vee x_{43} \vee x_{60},\\
C_{157} &= \neg x_{38} \vee \neg x_{59} \vee \neg x_{60},
\end{aligned}$$

During the first simulation, after the sequence of 22 assignments for 22 variables, we get

$$\begin{aligned}
&S\\
&= \{x_1; x_2; x_{24}; x_{19}; x_5; x_4; x_{21}; x_{30}; x_{25}; x_8; x_7; x_{27};\\
&\quad x_{15}; x_{13}; x_{17}; x_{10}; x_{11}; x_{33}; x_{31}; x_{37}; x_{35}; x_{38}\}.
\end{aligned}$$

The FUDASAT system has detected a conflict and, moreover, the conflict has also occurred for a negative assignment for the last variable x_{38} at level 22. If the system acted as described in Example A1, i.e., it added to the database the entire information about the sequence combination containing all 21 (without x_{38}) variables, The analysis would never stop or would take long time and consume many resources. This redundancy may unnecessarily complicate the initial function and from a practical point of view is useless. As a matter of fact, not every of those 22 assignments that preceded the conflict has an impact on the conflict situation.

Accordingly, the analysis of the conflict, which (if it occurs) gives a lot of information, should be done more carefully. The authors of the latest works in the field (Han *et al.*, 2010) have proposed a concept of the

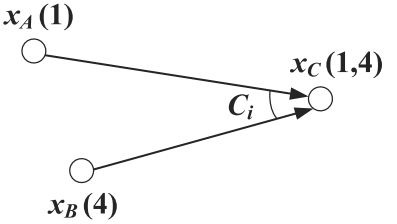

Fig. 6. Example of the implication graph for clause $C_i = \neg x_A \vee \neg x_B \vee x_C$.

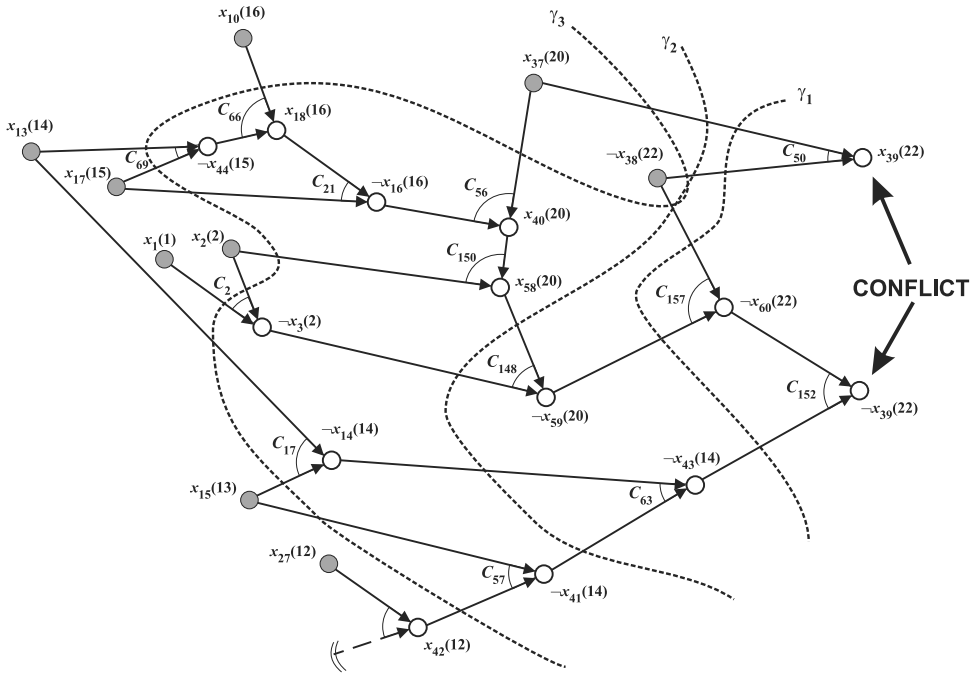

Fig. 7. Fragment of the implication hypergraph for a sequence of 22 assignments for the Example 2.

implication hypergraph. The hypergraph is a Directed Acyclic Graph (DAG) consisting of nodes and branches (directed arcs). The nodes show the assignments to variables, while branches present the transformations (reductions) of clauses. Figure 6 presents a fragment of the implication hypergraph. It explains the mechanism of graph construction. The assignments x_A = '1' and x_B = '1' on the level 1 and on the level 4, respectively, force (based on the clause $C_i = \neg x_A \vee \neg x_B \vee x_C$) the assignment x_C = '1' (binary propagation of constraints). Generally, in an implication hypergraph (Fig. 7), we can distinguish edge nodes and intermediate nodes. The former (gray shaded) represent the assignments of variables generated by the algorithm (system selections), while the latter reflect unit clauses as effects of binary constraint propagation.

A set of branches pointing to the same intermediate node is connected by the labeled arc corresponding to the clause number. Specially constructed cuts allow recognizing the source of the conflict. Figure 7 shows the implication hypergraph illustrating the above sequence of 22 assignments and their final effect—the conflict. The picture contains also three different (alternative) cuts drawn with dotted lines: γ_1, γ_2 and γ_3.

Each cut 'separates' the conflict nodes from the edge nodes. The arcs covered by a given cut show the assignments responsible for the conflict (they are the sources of the conflict). The conflict clauses describing the cuts can be obtained from linear resolution, which can be expressed by the following inference rule:

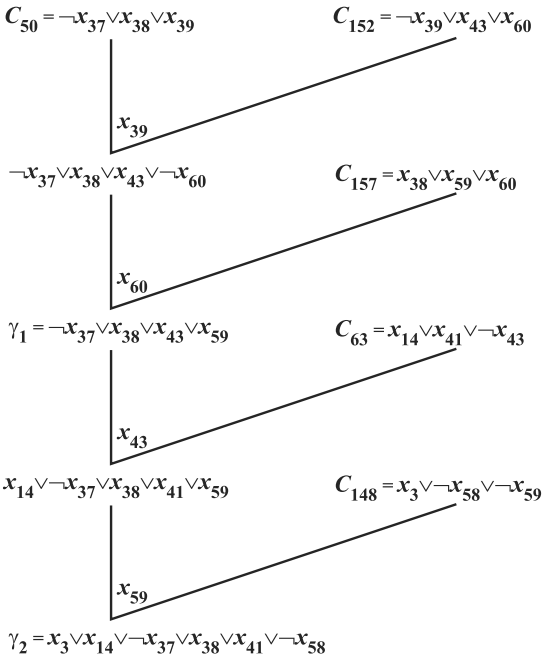

Fig. 8. Series of resolutions showing the process of generating cut clauses γ_1 and γ_2.

$$\frac{A \vee B \;\; \neg A \vee C}{B \vee C}. \tag{9}$$

The diagram depicted in Fig. 8 shows the series of resolutions demonstrating the process of obtaining cuts γ_1

and γ_2 from Fig. 7. Each operation of linear resolution is denoted (in the corner) by the name of the removed variable.

Consequently, the cuts from Fig. 7 can be described by the following clauses:

$$\begin{aligned}\gamma_1 &= \neg x_{37} \vee x_{38} \vee x_{43} \vee x_{39},\\ \gamma_2 &= x_3 \vee x_{14} \vee \neg x_{37} \vee x_{41} \vee \neg x_{58},\\ \gamma_3 &= \neg x_1 \vee \neg x_2 \vee \neg x_{10} \vee \neg x_{13} \vee \neg x_{15}\\ &\quad \vee \neg x_{17} \vee \neg x_{37} \vee x_{38} \vee \neg x_{42}.\end{aligned} \tag{10}$$

However, there is still an open question: Which of the clauses (10) should be added to the initial description? There is no universal answer to this question. Moreover, in such cases we can ask some other questions, like how deeply the implication hypergraph should be investigated? Should the initial function be extended by the conflict clause (eventually the redundancy)? If yes, what is the best length of the added conflict clause? How to construct an optimal backtrack sequence? Unfortunately, as in the analyzed example, there are no general solutions. The first iterations of the main loop of most algorithms (and also of FUDASAT) are based just on statistics and some general factors like the frequency of variables appearance, so we can say that this is a kind of semi-blind search. Conflicts that come out during the proving process not only extend the evaluation times, but they usually give a lot of valuable information about the analyzed function.

Following many experiments with the previous version of the FUDASAT algorithm has led us to some interesting observations and regularities. When referring to the results presented above (Fig. 4), it is clear that an increase in the number of clauses not always favors reducing the evaluation times. This redundancy may be helpful with cutting the searching space and avoiding repeatable iterations, i.e., passing through the same paths. The answer to the question concerning the length of the conflict clause is closely related to the depth of hypergraph investigation. From the technical (precise) point of view, the answer is very imprecise: the depth of the search should be set reasonable. That is why we should use a very sophisticated inference engine that would be able to deal with vague information.

5.2. Introduction of dynamic rules. As mentioned above, the FUDASAT algorithm uses fuzzy default logic based on the FDR inference rules (5), which for a given level λ of analysis (number of iteration) has the form

$$\frac{Initial\_conditions : Set(\beta)}{\Phi^{\lambda}}, \tag{11}$$

where $Initial\_conditions$ corresponds to the initial prerequisites reflecting the current state and level and forbidden (conflict) combinations, $Set(\beta)$ denotes variables not assigned ($x_i \in Set(\beta)$) so far, while Φ^{λ} is a fuzzy hypothesis (6), where hypotheses h_i^{λ} are replaced by Boolean variables x_i. Example 1 from Section 5 introduces a new type of FDR called dynamic, i.e., values of the possibilities within a given fuzzy hypothesis can be dynamically updated during the analysis, after a conflict analysis. In other words, we can say that the FUDASAT system can contain a static set of FDRs, the so-called general-purpose inference rules and dynamic FDRs that keep data concerning the current state of the analysis. In the simplest case, these possibilities may represent the reverse of the frequencies of the choices that previously led to conflicts, i.e., if a given variable had already been selected before and this choice was recognized as a source of the conflict. And finally, as has been shown, FDL inference engine allows the final judging of the hypotheses by using different evaluation schemes.

The results of conflict analysis have a strong impact on the construction of backtrack scenarios. Some other authors (Marques-Silva and Sakallah, 1999; Moskewicz *et al.*, 2001; Han *et al.*, 2010) propose radical solutions in the case of conflict detection: returning to the beginning (first iterations) and starting again with different initial conditions. However, the author's experiments show that such a solution is not optimal and in many cases we lose *'good'* assignments, which do not necessarily appear to be a direct source of the conflict. Three solutions of backtrack scenarios have been implemented within the FUDASAT inference engine: the short or immediate backtrack, the direct backtrack, and the long backtrack.

Example 2 presents all kinds of these backtrack scenarios. The short backtrack is used if the system finds a conflict for the first positive assignment at a given level, then it tries negative assignments (complement of the variable) and denotes this level as the conflict level

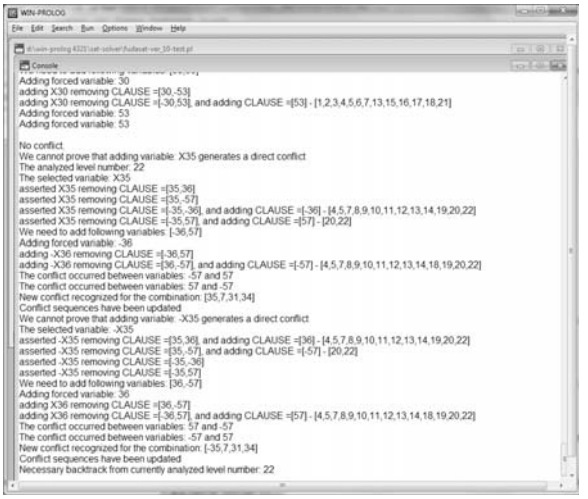

Fig. 9. Example of the FUDASAT program run in PROLOG.

of the current run. In the example it takes place in level 22 (the assignment of the variable x_{38}). If this change keeps the analysis in the conflict, the algorithm builds short backtrack sequences corresponding to the conflict clauses (cuts). Usually, we have only one sequence, although sometimes it is possible to record more. For the hypergraph from Fig. 7, we have three possible backtrack scenarios (numbers in brackets denote the return levels): 22, 20, 14 and 22, 20, 14, 2, and 22, 16, 15, 13, 12, 2, 1. The long backtrack corresponds to the return to the highest level of the current analysis. Therefore, the algorithm sometimes needs to be restarted. The latter radical step is used when the system tries to investigate one of the previously searched paths and/or the backtrack procedure has given no positive results. The best (reasonable) choice for the case discussed within the example seems to be γ_2 with the second backtrack sequence. However, not always is the choice so obvious and intuitive.

5.3. Techniques of pruning the searching space. This section addresses problems concerning search optimization methods. Two algorithms allowing reducing the number of iterations as well shrinking the searching space will be discussed. Beforehand, the term cliques should be defined.

Definition 1. A *clique* CQ_i is a set of all different clauses $\{V_1; V_2; V_3\}$ containing identical variables, which identify the clique. It could be denoted as $CQ_i \equiv C_i^{\{V_1;V_2;V_3\}}$. However, we usually use the shorter form CQ_i. Formally,

$$\begin{aligned} CQ_i \equiv C_i^{\{V_1;V_2;V_3\}} &\equiv \left\{C_i^1, C_i^2, \ldots, C_i^N\right\} \\ &\Leftrightarrow \left\{\forall_{C_i^k, C_i^m \in CQ_i} C_i^k \neq C_i^m\right\} \\ &\wedge \left\{ \begin{array}{c} C_i^k = V_1^k \vee V_2^k \vee V_3^k \\ C_i^m = V_1^m \vee V_2^m \vee V_3^m \end{array} \right\} \\ &\wedge \left\{ \begin{array}{c} V_1^k, V_1^m \in \{x_1; \neg x_1\} \\ V_2^k, V_2^m \in \{x_2; \neg x_2\} \\ V_3^k, V_3^m \in \{x_3; \neg x_3\} \end{array} \right\}. \end{aligned} \tag{12}$$

For example, the clauses $C_1 = x_4 \vee \neg x_5 \vee x_8$ and $C_1 = \neg x_4 \vee x_5 \vee \neg x_8$ belong to the same clique, while the clauses $C_4 = x_2 \vee \neg x_5 \vee \neg x_6$ and $C_7 = x_4 \vee x_5 \vee \neg x_8$ are members of different cliques.

In fact, every clause represents a forbidden combination, namely, a given clause $C_k = x_2 \vee \neg x_4 \vee \neg x_8$ corresponds to the forbidden combination $X_k = \neg x_2 \wedge x_4 \wedge x_8$ (the sequence of assignments x_2 = '0', x_4 = '1' and x_8 = '1'). Based on this information, it is possible to force the assignments to multiple variables at the same moment (in the same iteration), reducing the number of binary constraint propagations (Arangú and Salido, 2011).

There are two algorithms Pruning the Search Space (PSS), namely, PSS1 and PSS2, described in Appendix B, while the examples given in Appendix C show benefits of the approach: the former addresses the problem of search space pruning, while the latter concerns function decomposition, which is a very common technique used in various practical applications of Boolean functions (Opara and Kania, 2010; Wyrwoł and Hrynkiewicz, 2013).

6. Concluding remarks

The experiments carried out with hundreds of benchmarks allow modifying the set of inference rules (FDRs) in order to obtain a more effective tool. This certifies the openness of the FUDASAT system (especially its FDL-based engine). The stars added to the block diagram of the basic FUDASAT algorithm show those stages of the methodology where the suggested modifications can speed up the proving process. The most important is the problem of conflict analysis, which delivers very important information about the analyzed function. The new elements that should be introduced to the original algorithm include the analysis of implication hypergraphs, introduction of two types of FDR inference rules: static and dynamic (with dynamically changed trustworthiness), flexible building backtrack strategies based on conflict analysis and introduction of tools pruning the search space with the idea of multiple variable assignment. The modification of fuzzy hypotheses allows using information coming from conflicts and we can call the system a self-learning tool. The approach presented in our previous works (Pułka, 2011) uses a simple and rather intuitive investigation of the inference chain to produce the backtrack trace (see the example in Appendix A), while the experiments with implication hypergraphs (suggested by Han *et al.* (2010)) have shown that in many cases it is possible to construct the backtrack trace with a controlled depth.

The mechanisms allowing pruning the search space (algorithms PSS1 and PSS2) are relatively simple and allow cutting the problem and reducing its amount to reasonable sizes that can be analyzed in detail with exhausted investigation of all states. For example, problems consisting of 50 variables (over 10^{15} different combinations) can be reduced to tasks requiring the analysis of about 10^6 states. The algorithms PPS1 and PSS2 are introduced to the modified FUDASAT analyzer and constitute a new tool that allows building the optimal strategy replacing a single variable assignment with a multiple variable assignment and reducing binary constraint propagation. The FDL-based inference engine always decides weather or not to add a conflict clause. It is also responsible for cutting the search space.

Detailed analysis of some experiments shows that in many cases the conflict can become apparent much

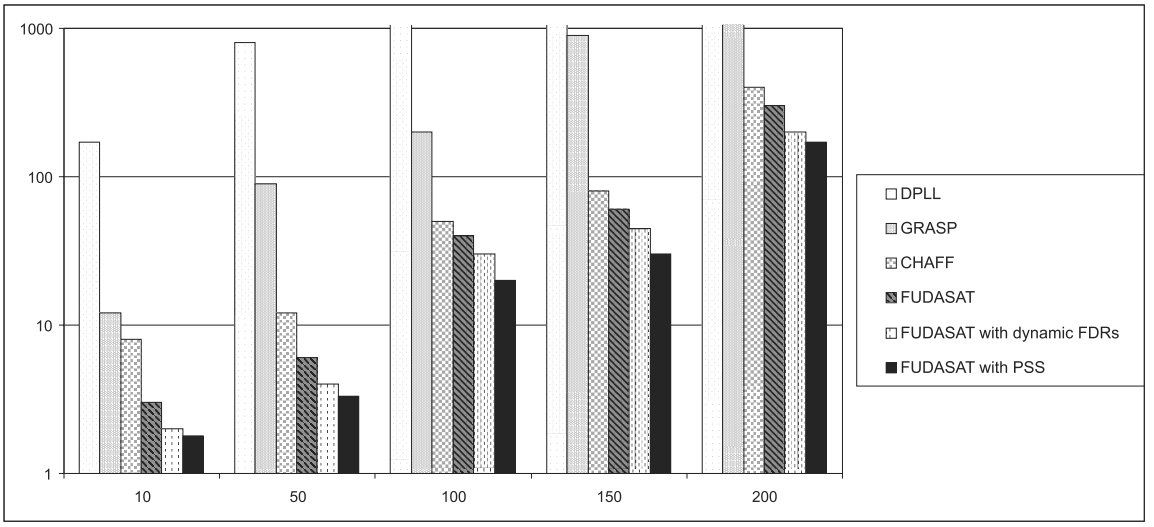

Fig. 10. Evaluation times of different algorithms related to the number of variables (average values for selected benchmarks (DIMACS, 1993)).

earlier, i.e., at the previous levels of the search process. Example A1 in Appendix A shows that, in fact, at the level 9' we can prove unsatisfiability of the reduced function, because it contains all 4 maxterms (Yin *et al.*, 2012) of variables x_4 and x_8. In other words, there is no sense to go to the level 12 to discover the conflict. To eliminate this weakness of the system, we have to introduce an appropriate procedure checking the maxterm covering. The modified version of the FUDASAT algorithm contains such rules for 2-variable clauses.

7. Summary

The approach presented in the paper introduces some modifications to a FUDASAT methodology that combines non-monotonic reasoning with satisfiability analysis. Thanks to a series of experiments and investigations, it was possible to eliminate some weaknesses of the original FUDASAT algorithm and optimize the inference process.

The obtained results, in comparison with other approaches (GRASP, CHAFF), have already proved the efficiency of the methodology (Pułka, 2011), and the suggested modifications show optimization of the original FUDASAT approach (Fig. 10). The evaluation times for unsatisfied problems are usually several times bigger. Theoretically, the proof that a given formula is unsatisfied requires exhaustive search. However, in practice it is sufficient to find a small subset of variables which do not satisfy a given logical function. The experiments showed that the best efficiency could be obtained with the mixed static-dynamic variable selection scheme.

Very interesting seems the idea of mixing a single variable assignment with multiple variable assignments, i.e., if a number of combinations given by 2-SAT clauses is too big, it is possible to go back to single variable assignments (the standard FUDASAT mode). This problem is currently being investigated by the author.

And, finally, we have the problem of the generalization of the presented methodology to all classes of SAT problems. The presented approach focuses on 3-SAT problems, which are representative of NP-hard SAT problems. However, without any fundamental changes, it could be applied to all classes of SAT problems. The main part of the system (its shell) consists of universal rules handling clauses of any length. Only the control part of PSS algorithms should be modified and extended to be able to analyze clauses of different lengths. The system has to be supplied with the mechanisms controlling clauses reduction and the clique cardinality assessment needs to be based on more complicated factors. Moreover, after the first conflict analysis, the function may not represent 3-SAT problems, if the added conflict clause consists of more literals. Thus, in this sense the methodology is universal.

References

Aloul, F., Mneimneh, M. and Sakallah, K. (2002). ZBDD-based backtrack search SAT solver, *Proceedings of the International Workshop on Logic Synthesis, Lake Tahoe, CA, USA*, pp. 131–136.

Arangú, M. and Salido, M.A. (2011). A fine-grained arc-consistency algorithm for non-normalized constraint satisfaction problems, *International Journal of Applied Mathematics and Computer Science* **21**(4): 733–744, DOI: 10.2478/v10006-011-0058-2.

Balduccini, M., Gelfond, M. and Nogueira, M. (2006). Answer set based design of knowledge systems, *Annals of Mathematics and Artificial Intelligence* **47**(1–2): 183–219.

Brewka, G. (1991). Cumulative default logic: In defense of nonmonotonic inference rules, *Artificial Intelligence* **50**(2): 183–205.

Davis, M., Logemann, G. and Loveland, D. (1962). A machine program for theorem proving, *Communications of the ACM* **5**(7): 394–397.

Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory, *Journal of the ACM* **7**(3): 201–215.

DIMACS (1993). CNF benchmarks database, `ftp://dimacs.rutgers.edu/pub/challenge/sat/benchmarks/cnf/`.

Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming, *in* R.A. Kowalski and K.A. Bowen (Eds.), *Proceedings of the International Logic Programming Conference and Symposium*, MIT Press, Cambridge, MA, pp. 1070–1080.

Han, H., Somenzi, F. and Jin, H. (2010). Making deduction more effective in SAT solvers, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **29**(8): 1271–1284.

Hu, Y., Shih, V., Majumdar, R. and He, L. (2008). Exploiting symmetries to speed up SAT-based Boolean matching for logic synthesis of FPGAs, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **27**(10): 1751–1760.

Lukasiewicz, T. and Straccia, U. (2008). Tightly coupled fuzzy description logic programs under the answer set semantics for the semantic web, *International Journal on Semantic Web and Information Systems* **4**(3): 68–89.

Marques-Silva, J. and Sakallah, K. (1999). GRASP: A search algorithm for propositional satisfiability, *IEEE Transactions on Computers* **48**(5): 506–521.

Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L. and Malik, S. (2001). Chaff: Engineering an efficient SAT solver, *Proceedings of the Design Automation Conference, Las Vegas, NV, USA*, pp. 530–535.

Opara, A. and Kania, D. (2010). Decomposition-based logic synthesis for PAL-based CPLDs, *International Journal of Applied Mathematics and Computer Science* **20**(2): 367–384, DOI: 10.2478/v10006-010-0027-1.

Pułka, A. (2009). Decision supporting system based on fuzzy default reasoning, *Proceedings of the IEEE Human Systems Interaction Conference, HSI'09, Catania, Italy*, pp. 32–39.

Pułka, A. (2011). An effective SAT-solving mechanism with backtrack controlled by FDL, *Proceedings of the 18th International Conference on Mixed Design of Integrated Circuits and Systems (MIXDES), Gliwice, Poland*, pp. 252–257.

Reiter, R. (1980). A logic for default reasoning, *Artificial Intelligence* **13**(1): 81–132.

Suyama, T., Yokoo, M. and Nagoya, A. (1999). Solving satisfiability problems on FPGAs using experimental unit propagation heuristic, parallel and distributed processing, *in* J. Rolim (Ed.), *Parallel and Distributed Processing*, Lecture Notes in Computer Science, Vol. 1586, Springer-Verlag, Berlin, pp. 709–711.

Tille, D., Eggersgluss, S. and Drechsler, R. (2010). Incremental solving techniques for SAT-based ATPG, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **29**(7): 1125–1130 .

Wyrwoł, B. and Hrynkiewicz, E. (2013). Decomposition of the fuzzy inference system for implementation in the FPGA structure, *International Journal of Applied Mathematics and Computer Science* **23**(2): 473–483, DOI: 10.2478/amcs-2013-0036.

Yin, L., He, F., Hung, W., Song, X. and Gu, M. (2012). Maxterm covering for satisfiability, *IEEE Transactions on Computers* **61**(3): 420–426.

Zadeh, L.A. (2006). Generalized theory of uncertainty (GTU)—principal concepts and ideas, *Computational Statistics and Data Analysis* **51**(1): 15–46.

Zadeh, L.A. (2008). Is there a need for fuzzy logic?, *Information Sciences: An International Journal* **178**(13): 2751–2779.

Andrzej Pułka received the M.Sc. Ph.D. and D.Sc. degrees in electronics from Silesian Technical University, Gliwice, Poland, in 1989, 1997 and 2013, respectively. Currently he is an assistant professor in the Institute of Electronics, Silesian University of Technology, Gliwice, Poland. He is the author and a co-author of about 70 scientific papers including journal articles, book chapters and conference papers. His research interests cover automated design of digital and mixed signal circuits in FPGAs, modeling and simulation of electronic embedded systems, precision time machines (PRET), AI and commonsense reasoning modeling, and genome pattern recognition. He is a senior IEEE member and a member of the Electronics Commission of the Polish Academy of Sciences.

Appendix A

The following example presents the philosophy of backtrack scenario construction in the basic FUDASAT algorithm.

Example A1. The example is based on a 3-SAT problem coming from benchmarks (DIMACS, 1993). Only the fragment of the inference chain and a subset of clauses will be analyzed. The interesting subset S consists of the following clauses:

$$C_1 = x_4 \vee \neg x_7 \vee x_8, \qquad C_2 = x_4 \vee x_7 \vee x_8,$$
$$C_3 = x_1 \vee x_6 \vee \neg x_9, \qquad C_4 = \neg x_1 \vee \neg x_3 \vee \neg x_9,$$
$$C_5 = x_3 \vee \neg x_5 \vee \neg x_9, \qquad C_6 = x_5 \vee \neg x_7 \vee \neg x_9,$$
$$C_7 = \neg x_1 \vee \neg x_4 \vee \neg x_8, \qquad C_8 = \neg x_4 \vee x_8 \vee x_9,$$
$$C_9 = x_4 \vee \neg x_8 \vee x_9.$$

The analysis is reduced only to some interesting levels of the inference chain—the fragment which is important to observe the sources of conflicts. Let us assume that the following variables are asserted: x_7 at Level 3, x_7 at Level 7, x_9 at Level 9 and x_4 at Level 12. The following transformations take place:

- **At Level 3:**
  C_2 is absorbed by x_7, $C_1 \rightarrow C_1^3 = x_4 \vee x_8$,
  $C_6 \rightarrow C_6^3 = x_5 \vee \neg x_9$, $C_7 \rightarrow C_7^3 = \neg x_4 \vee \neg x_8$.

- **At Level 5:**
  C_3 is absorbed by x_1, $C_4 \rightarrow C_4^5 = \neg\, x_3 \vee \neg x_9$.

- **At Level 9:**
  C_8 and C_9 are absorbed by x_9, $C_4^5 \rightarrow C_4^{5,9} = \neg x_3$ (the unit clause),
  $C_6^3 \rightarrow C_6^{3,9} = x_5$ (the unit clause),
  $C_5 \rightarrow C_5^9 = x_3 \vee \neg x_5$, so we have a conflict!
  The conflict combination $Conflict_1 = x_1 \wedge x_7 \wedge x_9$ is recorded.

- **At Level 9:**
  (for the variable assignment $x_9 = 0$ and all previous transformations for that level invalidated):
  Three clauses are absorbed by $\neg x_9$: C_4^5, C_5 and C_6^3 are absorbed by $\neg x_9$, $C_8 \rightarrow C_8^9 = \neg x_4 \vee x_8$, $C_9 \rightarrow C_9^9 = x_4 \vee \neg x_8$.

- **At Level 12:**
  C_1^3 and C_9^9 are absorbed by x_4, $C_7^3 \rightarrow C_7^{3,12} = \neg x_8$ (the unit clause),
  $C_8^9 \rightarrow C_8^{9,12} = x_{58}$ (the unit clause) and this causes a conflict!
  The conflict combination:
  $Conflict_2 = x_4 \wedge x_7 \wedge\neg\ x_9$ (since x_9 has been complemented at Level 9') is recorded.

- **At Level 12'**
  (for the variable assignment x_4 = '0' and all previous transformations for Level 12 invalidated):
  C_7^3 and C_8^9 are absorbed by $\neg x_4$; $C_8^9 \rightarrow C_8^{9,12} = x_8$ (the unit clause),
  $C_9^9 \rightarrow C_9^{9,12} = \neg x_8$, so we have a conflict!
  $Conflict_3 = \neg x_4 \wedge \neg x_9$ is recorded.

This sequence of transformations shows that we have found an unresolved conflict on Level 12, so a backtrack is required. The basic FUDASAT algorithm generates at Stage 7 possible backtrack scenarios: [$Level_9$, $Level_{12}$]; [$Level_3$, $Level_9$, $Level_{12}$], and the algorithm tries to find another assignment for the combination of variables x_4, x_7 and x_9 that are responsible for the conflict. ♦

Appendix B

PSS1 and PSS2 (Algorithms 1 and 2, respectively) are two algorithms for searching space pruning.

Algorithm 1. PSS1.

Step 1.1: findall cliques CQ_i of Boolean function f;

Step 1.2: Find the cliques CQ_{sel} of the greatest cardinality: $|CQ_{sel}| = \max(|CQ_1|, |CQ_2|, \ldots, |CQ_N|)$;

Step 1.3: **findall** possible (allowed) combinations $Comb_k(V_1'; V_2'; V_3')$ of variables $\{V_1, V_2, V_3\}$ that identify the selected clique $CQ_{sel}^{\{V_1,V_2,V_3\}}$, i.e.
$Comb_k(V_1', V_2', V_3') \Rightarrow \forall_{C_i \in CQ_{sel}} C_i(V_1'; V_2'; V_3') =$ '1';.

Step 1.4: Assign to the variables $\{V_1; V_2; V_3\}$ the first combination $Comb_1$.

Step 1.5: Remove all clauses belonging to the clique $CQ_{sel}^{\{V_1,V_2,V_3\}}$ and **findall** possible reduction of the rest clauses (see Step 6 of the FUDASAT algorithm).
In **case** of a conflict take next assignment for $\{V_1, V_2, V_3\}$ and **repeat** Step 1.5.

1: **if** there is no other combination **then**
2: **return *UNSATISFIED*** **stop**
3: **else**
4: **goto** Step 1.6
5: **end if**

Step 1.6: **SET** the $iteration\_level = 1$ and **run** Algorithm PSS2.

Step 1.7:

1: **if** Algorithm PSS2 returns ***DIVISION*** **then**
2: decompose the function and **restart** perform the proof for each sub-function independently;
3: **else if** Algorithm PSS2 returns ***SUCCESS*** **then**
4: **return *SATISFIED*** **stop**
5: **else if** Algorithm PSS2 returns ***FAIL*** **then**
6: take next assignment for $\{V_1; V_2; V_3\}$ and **repeat** Step 1.4.
7: **else**
8: there is no other combination, so **return *UNSATISFIED*** **stop**
9: **end if**

Algorithm 2. PSS2.

Step 2.1: findall clauses reduced to the class 2-SAT, i.e., clauses consisting of 2 variables;

1: **if** there is no such clauses **then**
2: **stop** and **return *DIVISION***
{The set of assigned variables and the set of unassigned variables are ***strongly disjunctive***}
3: **else**
4: **continue**
5: **end if**

Step 2.2: find set Γ consisting of all possible (allowed) combinations of variables building 2-SAT clauses

1: **if** $\Gamma = \emptyset$ **then**
2: **stop** and **return *FAIL***
{The set of assigned variables and the set of unassigned variables are ***strongly disjunctive***}
3: **else**
4: Take the first assignment from set Γ and **continue**
5: **end if**

Step 2.3: Remove all 2-SAT clauses and **findall** possible reduction of the rest clauses (see Step 6 of the FUDASAT algorithm) in case of a conflict take the next assignment from Γ and **repeat** Step 2.3.

1: **if** there is no other combination **then**
2: **goto** one iteration level up; i.e.
$iteration\_level = iteration\_level - 1$;
3: **else if** $iteration\_level = 1$ **then**
4: **return *FAIL***
5: **else if** the set of unassigned variables is empty **then**
6: **return *SUCCESS***
7: **else**
8: increment iteration level; i.e.
$iteration\_level = iteration\_level + 1$;
9: **end if**

Step 2.4: run Algorithm PSS2 for the remaining set of clauses.

Appendix C

Two examples showing the benefits of the algorithms pruning the search space.

Example C1. Let us assume the following set of clauses that describes a given Boolean function f:

$$
\begin{aligned}
C_1 &= \neg x_1 \vee \neg x_2 \vee \neg x_3,\\
C_2 &= \neg x_1 \vee x_2 \vee x_3,\\
C_3 &= x_1 \vee x_2 \vee \neg x_3,\\
C_4 &= x_1 \vee \neg x_2 \vee x_3,\\
C_5 &= x_1 \vee x_2 \vee x_3,\\
C_6 &= \neg x_{22} \vee \neg x_{23} \vee \neg x_{24},\\
C_7 &= \neg x_{22} \vee x_{23} \vee x_{24},\\
C_8 &= x_{22} \vee x_{23} \vee \neg x_{24},\\
C_9 &= x_{22} \vee \neg x_{23} \vee x_{24},\\
C_{10} &= \neg x_{16} \vee \neg x_{37} \vee \neg x_{40},\\
C_{11} &= \neg x_{16} \vee x_{37} \vee x_{40},\\
C_{12} &= x_{16} \vee x_{37} \vee \neg x_{40},\\
C_{13} &= x_{16} \vee \neg x_{37} \vee x_{40},\\
C_{14} &= \neg x_1 \vee \neg x_{24} \vee \neg x_{49},\\
C_{15} &= \neg x_1 \vee x_{24} \vee x_{49},\\
C_{16} &= x_1 \vee x_{24} \vee \neg x_{49},\\
C_{17} &= x_1 \vee \neg x_{24} \vee x_{49},\\
C_{18} &= \neg x_{19} \vee \neg x_{23} \vee \neg x_{49},\\
C_{19} &= \neg x_{19} \vee x_{23} \vee x_{49},\\
C_{20} &= x_{19} \vee x_{23} \vee \neg x_{49},\\
C_{21} &= x_{19} \vee \neg x_{23} \vee x_{49},\\
C_{22} &= x_3 \vee x_{58} \vee x_{59},\\
C_{23} &= \neg x_3 \vee \neg x_{58} \vee x_{59},\\
C_{24} &= \neg x_3 \vee x_{58} \vee \neg x_{59},\\
C_{25} &= x_3 \vee \neg x_{58} \vee \neg x_{59},\\
C_{26} &= x_2 \vee x_{40} \vee x_{58},\\
C_{27} &= \neg x_2 \vee \neg x_{40} \vee x_{58},\\
C_{28} &= \neg x_2 \vee x_{40} \vee \neg x_{58},\\
C_{29} &= x_2 \vee \neg x_{40} \vee \neg x_{58},\\
C_{30} &= x_{38} \vee x_{59} \vee x_{60},\\
C_{31} &= \neg x_{38} \vee \neg x_{59} \vee x_{60},\\
C_{32} &= \neg x_{38} \vee x_{59} \vee \neg x_{60},\\
C_{33} &= x_{38} \vee \neg x_{59} \vee \neg x_{60}.
\end{aligned}
$$

The first selected clique with the greatest cardinality (equals 5) $CQ_{sel} = CQ_1$ is identified by the variables $\{x_1, x_2, x_3\}$ and consists of the clauses C_1, C_2, C_3, C_4 and C_5. So the set of allowed assignments to variables x_1, x_2, x_3 consists of 3 combinations: ["011", "101", "110"]. All these combinations absorb (satisfy) every clause of the clique CQ_1. Taking the first one (Step 1.4 of Algorithm 1) we can run Algorithm 2. Step 2.1 of the algorithm removes clauses $C_{14}, C_{15}, C_{22}, C_{25}, C_{26}$ and C_{29}, while six clauses are reduced,

$$
\begin{aligned}
C_{16} \rightarrow C_{16}^1 &= x_{24} \vee \neg x_{49},\\
C_{17} \rightarrow C_{17}^1 &= \neg x_{24} \vee x_{49},\\
C_{23} \rightarrow C_{23}^1 &= \neg x_{58} \vee x_{59},\\
C_{24} \rightarrow C_{24}^1 &= x_{58} \vee \neg x_{59},\\
C_{27} \rightarrow C_{27}^1 &= \neg x_{40} \vee x_{58},\\
C_{28} \rightarrow C_{28}^1 &= x_{40} \vee \neg x_{58}.
\end{aligned}
$$

These clause creates 3 2-SAT cliques with the following allowed combinations: $\{x_{24}, x_{49}\}$ = [“00”, “11”]; $\{x_{58}, x_{59}\}$ = [“00”, “11”] and $\{x_{40}, x_{58}\}$ = [“00”, “11”].

Accordingly, we get the set

$$\Gamma = \{[\text{“00000”}], [\text{“01011”}], [\text{“10100”}], [\text{“11111”}]\}$$

or, in other words, there exist four possible assignments for variables $x_{24}, x_{40}, x_{49}, x_{58}$ and x_{59}, namely,

assign_2.1: $\{x_{24}, x_{40}, x_{49}, x_{58}, x_{59}\}$ = [“00000”],
assign_2.2: $\{x_{24}, x_{40}, x_{49}, x_{58}, x_{59}\}$ = [“01011”],
assign_2.3: $\{x_{24}, x_{40}, x_{49}, x_{58}, x_{59}\}$ = [“10100”],
assign_2.4: $\{x_{24}, x_{40}, x_{49}, x_{58}, x_{59}\}$ = [“11111”].

Thus, after one assignment we reduced the search space to $4 \times 4 = 16$ combinations, instead of investigating 2^8 different states (for 8 variables).

Executing the first assignment assign_2.1, we can eliminate all 2-SAT clauses (as for the other three assignments in this step) and also the following clauses: $C_6, C_8, C_{10}, C_{12}, C_{18}, C_{20}, C_{31}$ and C_{33}. Eight clauses are reduced to 2-SAT clauses, namely,

$$\begin{aligned}
C_7 &\rightarrow C_7^2 = \neg x_{22} \vee x_{23},\\
C_9 &\rightarrow C_9^2 = x_{22} \vee \neg x_{23},\\
C_{11} &\rightarrow C_{11}^2 = \neg x_{16} \vee x_{37},\\
C_{13} &\rightarrow C_{13}^2 = x_{16} \vee \neg x_{37},\\
C_{19} &\rightarrow C_{19}^2 = \neg x_{19} \vee x_{23},\\
C_{21} &\rightarrow C_{21}^2 = x_{19} \vee \neg x_{23},\\
C_{30} &\rightarrow C_{30}^2 = x_{38} \vee x_{60},\\
C_{32} &\rightarrow C_{32}^2 = \neg x_{38} \vee \neg x_{60}.
\end{aligned}$$

Finally, after these transformation the set Γ consists of eight possible combinations that correspond to the following assignments for seven variables:

assign_3.1:
$\{x_{16}, x_{19}, x_{22}, x_{23}, x_{37}, x_{38}, x_{60}\}$ = [“0000001”],
assign_3.2:
$\{x_{16}, x_{19}, x_{22}, x_{23}, x_{37}, x_{38}, x_{60}\}$ = [“0000010”],
assign_3.3:
$\{x_{16}, x_{19}, x_{22}, x_{23}, x_{37}, x_{38}, x_{60}\}$ = [“0111001”],
assign_3.4:
$\{x_{16}, x_{19}, x_{22}, x_{23}, x_{37}, x_{38}, x_{60}\}$ = [“0111010”],
assign_3.5:
$\{x_{16}, x_{19}, x_{22}, x_{23}, x_{37}, x_{38}, x_{60}\}$ = [“1000101”],
assign_3.6:
$\{x_{16}, x_{19}, x_{22}, x_{23}, x_{37}, x_{38}, x_{60}\}$ = [“1000110”],
assign_3.7:
$\{x_{16}, x_{19}, x_{22}, x_{23}, x_{37}, x_{38}, x_{60}\}$ = [“1111101”],
assign_3.8:
$\{x_{16}, x_{19}, x_{22}, x_{23}, x_{37}, x_{38}, x_{60}\}$ = [“1111110”].

If the set of 33 clauses is a full description of the function *f*, we have just proved that the function is satisfied, otherwise we have to continue the search. However, the search space has been pruned to $4 \times 4 \times 8 = 128$ different states, so for the total number of variables being 15, this gives the reduction by the factor $2^8 = 256$ for the exhausted search. ♦

Example C2. The Boolean function $f = \bigcup_{i=1}^{29} C_i$ consists of the following clauses:

$$\begin{aligned}
C_1 &= \neg x_1 \vee \neg x_2 \vee \neg x_3,\\
C_2 &= \neg x_1 \vee x_2 \vee x_3,\\
C_3 &= x_1 \vee x_2 \vee \neg x_3,\\
C_4 &= x_1 \vee \neg x_2 \vee x_3,\\
C_5 &= x_4 \vee x_{10} \vee x_{17},\\
C_6 &= \neg x_4 \vee \neg x_5 \vee \neg x_7,\\
C_7 &= \neg x_4 \vee x_5 \vee x_7,\\
C_8 &= x_4 \vee x_5 \vee \neg x_7,\\
C_9 &= x_4 \vee \neg x_5 \vee x_7,\\
C_{10} &= \neg x_{16} \vee \neg x_{17} \vee \neg x_{10},\\
C_{11} &= \neg x_{10} \vee x_{16} \vee x_{17},\\
C_{12} &= x_{10} \vee x_{16} \vee \neg x_{17},\\
C_{13} &= x_{10} \vee \neg x_{16} \vee x_{17},\\
C_{14} &= \neg x_1 \vee \neg x_6 \vee \neg x_9,\\
C_{15} &= \neg x_1 \vee x_6 \vee x_9,\\
C_{16} &= x_1 \vee x_6 \vee \neg x_9,\\
C_{17} &= x_1 \vee \neg x_6 \vee x_9,\\
C_{18} &= \neg x_7 \vee \neg x_8 \vee \neg x_{10},\\
C_{19} &= \neg x_7 \vee x_8 \vee x_{10},\\
C_{20} &= x_7 \vee x_8 \vee \neg x_{10},\\
C_{21} &= x_7 \vee \neg x_8 \vee x_{10},\\
C_{22} &= x_3 \vee x_9 \vee x_{11},\\
C_{23} &= \neg x_3 \vee \neg x_{11} \vee x_9,\\
C_{24} &= \neg x_3 \vee x_{11} \vee \neg x_9,\\
C_{25} &= x_3 \vee \neg x_6 \vee \neg x_9,\\
C_{26} &= x_2 \vee x_6 \vee x_{11},\\
C_{27} &= \neg x_2 \vee \neg x_6 \vee x_{11},\\
C_{28} &= \neg x_2 \vee x_6 \vee \neg x_{11},\\
C_{29} &= x_2 \vee \neg x_6 \vee \neg x_{11}.
\end{aligned}$$

The first selected clique with the greatest cardinality (equals 4) CQ_1 is identified by variables $\{x_1, x_2, x_3\}$ and consists of the clauses C_1, C_2, C_3 and C_4. Consequently, the set of allowed assignments to variables x_1, x_2, x_3 consists of four combinations: [“011”, “101”, “110”,“000”]. All these combination absorb (satisfy) every clause of the clique CQ_1. Taking

the first one (Step 1.4 of Algorithm 1), we can run Algorithm 2. Step 2.1 of the algorithm removes the clauses: $C_{14}, C_{15}, C_{22}, C_{25}, C_{26}$ and C_{29}, while six clauses are reduced:

$$\begin{aligned}
C_{16} &\rightarrow C_{16}^{1} = x_9 \vee \neg x_{11},\\
C_{17} &\rightarrow C_{17}^{1} = \neg x_6 \vee x_9,\\
C_{23} &\rightarrow C_{23}^{1} = \neg x_9 \vee x_{11},\\
C_{24} &\rightarrow C_{24}^{1} = x_{11} \vee \neg x_9,\\
C_{27} &\rightarrow C_{27}^{1} = \neg x_6 \vee x_{11},\\
C_{28} &\rightarrow C_{28}^{1} = x_6 \vee \neg x_{11}.
\end{aligned}$$

These clauses from three 2-SAT cliques with the following allowed combinations: $\{x_6, x_9\}$ = [“00”, “11”]; $\{x_9, x_{11}\}$ = [“00”, “11”] and $\{x_6, x_{11}\}$ = [“00”, “11”]. Because all clauses have common elements $\Gamma = \{[\text{“000”}], [\text{“111”}]\}$ (only two combinations satisfy 2-SAT clauses), two assignments are possible:

assign_2.1: $\{x_6, x_9, x_{11}\}$ = [“000”],
assign_2.2: $\{x_6, x_9, x_{11}\}$ = [“111”].

Any of the above two assignments produce no further reductions, so we have found that a set of clauses $C_1, C_2, C_3, C_4, C_{14}, C_{15}, C_{16}, C_{17}, C_{22}, C_{23}, C_{24}, C_{25}, C_{26}, C_{27}, C_{28}$ and C_{29} constitutes the independent sub-function f_1, which is a part of the function strongly disjunctive from the remaining part of the original function f. In other words, the set of the variables $\{x_1, x_2, x_3, x_6, x_9, x_{11}\}$ can be analyzed separately. As a side effect of the search space pruning algorithm, function decomposition has been achieved. ♦

Received: 13 January 2013
Revised: 4 October 2013