Serdica J. Computing **1** (2007), 87–100

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

APOLLO 13 RISK ASSESSMENT REVISITED

István Bukovics

Abstract. Fault tree methodology is the most widespread risk assessment tool by which one is able to predict – in principle – the outcome of an event whenever it is reduced to simpler ones by the logic operations conjunction and disjunction according to the basics of Boolean algebra. The object of this work is to present an algorithm by which, using the corresponding computer code, one is able to predict – in practice – the outcome of an event whenever its fault tree is given in the usual form.

Introduction. Fault tree methodology is the most widespread risk assessment tool by which one is able to predict – in principle – the outcome of an event whenever it is reduced to simpler ones by the logic operations conjunction ("AND") and disjunction ("OR"). Since the last century's sixties a number of monographs, textbooks, bibliographies and standards are available dealing with fault tree methodology [1].

In the following it is supposed that the reader is familiar with the basics including the most important Boolean algebraic notions. Still, they are summarized for the sake of convenience below.

ACM Computing Classification System (1998): F.4.1.
Key words: Fault tree, risk assessment, prime event, conjunction and disjunction.

The object of this work is to present an algorithm by which, using the corresponding computer code, one is able to predict – in practice – the outcome of an event whenever its fault tree is given in the usual form. The name of the algorithm is the acronym "FLORIAN" (Failure Logic Oriented Risk Imminence Assessment Normatives) given in honor of the firemen's patron St. Florian. The FLORIAN algorithm will be applied to the Apollo 13 fault tree and it will be shown how to predict the outcome of the top event

"Fuel cell power not available on main service module buses".

One of the most notable fault trees is that of the Apollo 13 accident elaborated by the NASA in 1970 June [2].

It is supposed that the reader can access the Internet sites where further references can be found. The Apollo 13 fault tree is quite sizable having almost 200 "base events" and more then 300 "composite events". See the definitions in a later part of the present article.

These events constitute an indirect Boolean function of about 200 variables.

The mathematical problem is to determine the "roots" of this function. The root is the set of all base events whose non-occurrence results in the non-occurrence of the top event. It is out of the question to determine **all the roots** of such a huge Boolean function. However, it will be shown that it is always possible – in practice (using an up to date PC) – to determine the "best" root in a natural sense.

The problem of finding the root of a (not necessarily indirect) Boolean function is in close relationship with the **Boolean satisfiability problem (SAT)**[3].

By definition it is: given a Boolean expression written using only *AND*, *OR*, *NOT*, variables, and parentheses, is there some assignment of *TRUE* and *FALSE* values to the variables that will make the entire expression true? Now our "Boolean root seeking problem" sounds like this: given a Boolean expression written using only *AND*, *OR*, variables, and parentheses, is there some assignment of *FALSE* values to the variables that will make the entire expression false?

SAT-problems are intensively investigated in several branches of science (see e.g. [4]).

Notions, notations and conventions.

Events [5]. Events are denoted by $A, B, C, \ldots$ or sometimes if necessary, by double capitals as e.g. TE

If A occurs if an only if B, then it is written that $A = B$.

It is well-known that if $A = B$ and $B = C$ then $B = A$ and $A = C$.

The event that occurs if and only if both B and C occur is called the **conjunction** (or the meet) of B and C and is written as $B \wedge C$.

The event that occurs if and only if either B or C occurs is called the **disjunction** (or the union) of B and C and is written as $B \vee C$.

The operation signs $\wedge$ and $\vee$ are loosely speaking also called conjunction and disjunction respectively.

The basic properties of disjunction and conjunction are as follows.

For all A, B, C

$$A \wedge (B \wedge C) = (A \wedge B) \wedge C \quad \text{and} \quad A \vee (B \vee C) = (A \vee B) \vee C,$$

$$A \wedge B = B \wedge A \quad \text{and} \quad A \vee B = B \vee A,$$

$$A \wedge (B \vee A) = A \quad \text{and} \quad A \vee (B \wedge A) = A,$$

$$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C) \quad \text{and} \quad A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C).$$

The event that always occurs is denoted by I.

The event that never occurs is denoted by O.

Always:

$$A \wedge I = A \quad \text{and} \quad A \vee O = A.$$

The equations above are the axioms (or the basic rules) of Boolean algebra.

If $A = A \wedge B$ then it is written that $A \subseteq B$ or $B \supseteq A$ and it is said that A **implies** B or that B **is implied by** A.

It follows from the above that for all A, B, C:

$$\text{if} \quad A \subseteq B \quad \text{and} \quad B \subseteq C \quad \text{then} \quad A \subseteq B$$

and

$$\text{if} \quad A \subseteq B \quad \text{and} \quad B \subseteq A \quad \text{then} \quad A = B.$$

If A implies B and $A \neq B$ then we write: $A \subset B$ or, equivalently, $B \supset A$.

If $A \subset B$ and there is no such C for which $A \subset C$ and $C \subset B$, then we say that B is a/the **parent** of A.

Fault tree events. The Fault Tree in question is denoted by **FT**.

Events belonging to the **FT** are indexed by $e, f, g, \ldots \in \{1, 2, \ldots, nEvt\}$, where $nEvt$ is the total number of events occurring in the **FT**.

The event of index e is denoted by Evt(e). Instead of speaking of the event Evt(e) we speak for short of "the event e".

The index of the TopEvent of the **FT** is 1.

If Evt(e) is the case then it is said that the **logic value** of Evt(e) is *True*, denoted by: Evt(e).LogVal = "T". Also, we say that Evt(e) is **active** (or loosely speaking "e is active").

If Evt(e) is not the case then it is said that the logic value of Evt(e) is *False*, denoted by: Evt(e).LogVal = "F". Also, we say that Evt(e) is **passive** (or loosely speaking "e is passive").

If the logic value "F" (*False*) is assigned to an active event then we say "e is **passivated**".

In a fault tree each event expects that the topevent has a unique parent. The index of the **parent** of Evt(e) is denoted by Evt(e).Parent. If f = Evt(e).Parent then we say "f is the parent of e".

Each event Evt(e) has a unique **name**. It is denoted by Evt(e).Name.

To be able to treat common cause events properly, we must differentiate between an event's name and content.

For all e, Evt(e).Name consists of three parts: the **Prefix**, the **PostFix** and the **namecontent** of Evt(e).

The Prefix of Evt(e) is denoted by Evt(e).PreFix, e. g. "3.2.1.1.8.4". It is a unique string of integers to identify the event and its parent simultaneously.

Thus the parent of 3.2.1.1.8.4 is 3.2.1.1.8; the fourth child's prefix of the event with prefix 3.2.1.1.8 is 3.2.1.1.8.4.

The Postfix of Evt(e) is denoted by Evt(e).PostFix, e. g. "(V)". It represents the logic type of e in computer code.

The Content of (the name of) Evt(e) is denoted by Evt(e).Content, e.g. "2-3" or "1-1 Generated power not delivered to buses."

Here only the short form (1-1, 1-2, ...) will be used as it stands in the original NASA fault tree. While the name of the event is – by construction – unique, it may happen that the content is not. If two events e and f have equal content and different parents g and h respectively, then we say that g and h are common cause events.

If Evt(e) is the parent of events Evt(f), Evt(g), Evt(h),... then Evt(f), Evt(g), Evt(h),... are called the **children** of Evt(e). Loosely speaking they are called the "Kids" of e. The set of all children of Evt(e) is denoted by Evt(e).Children.

If an event has a child then it is called a **composite** event.

It follows that:

A composite conjunctive event is passive if and only if it has a passive child.

A composite disjunctive event is active if and only if it has an active child.

A prime event is **forlorn** (or "do not care") if

(1) it is passive and its parent is active disjunctive;

(2) it is active and its parent is passive conjunctive.

If an event has no children then it is called a **PrimeEvent** (or BaseEvent).

If $Evt(e)$ is a PrimeEvent then, for the sake of brevity, we say "e is **prime**".

If a child of Evt(e) is prime then it is called, for short, a "**PrimeKid**".

For each active primeevent e one can speak of the **renovation cost** of p, notated Evt(e).RenCost. It is the cost that one has to pay to passivate e.

Similarly, Evt(e).RenTime is the notation of the time that is necessary to passivate the active primeevent e.

The collection name for Evt(e).RenTime and Evt(e).RenCost is the **Franklin property** of e.

The **best PrimeKid of an active event f with respect to a Franklin property** is the event e such that f = Evt(e).Parent and Evt(e).RenCost or Evt(e).RenTime is minimal among all the children of f (i.e. the siblings of e) respectively according to f's given Franklin property.

In the present paper Franklin properties are randomly assigned to each prime event in the interval [1, 99].

The **level** of event Evt(e) is defined recursively and denoted by Evt(e).Level.

Evt(1).Level is 0 (i. e. the TopEvent's level of the **FT** is by definition 0).

If Evt(e).Level = L for $e = 1, 2, \ldots, nEvt$; $L = 1, 2, \ldots$ and e is the parent of Evt(f) for any $f \neq e$, then Evt(f).Level = $Evt(e)$.Level + 1.

The maximal value of Evt(e).Level ($e = 1, 2, \ldots, nEvt$) is denoted by $Lmax$.

Fault tree and event functions.

Event functions and Boolean functions are synonyms in our context.

Our main concern is indirect Boolean function without negation. To every fault tree there belongs an indirect Boolean function without negation.

Consider an excerpt from the Apollo 13 fault tree:

The indirect Boolean function TE (Top Event) corresponding to the fault tree fragment of Fig. 1 is as follows:

TE = $A \vee B \vee C$ is the topevent of **FT.**

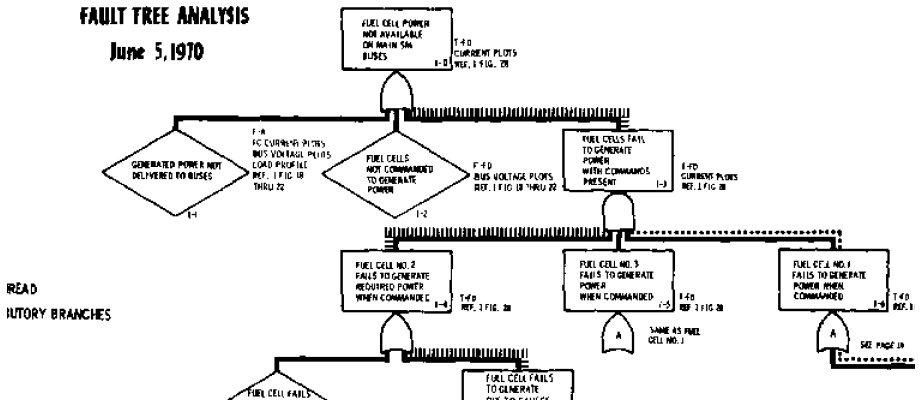

Fig. 1. Excerpt from the Apollo 13 fault tree [6]

Its namecontent: "1-0 Fuel cell power not available on service module buses."

A is a primitive event.

Its name: "1-1 Generated power not delivered to buses."

B is a primitive event.

Its namecontent: "1-2 Fuel cells not commanded to generate power."

$C = D \wedge E \wedge F$ is a conjunctive event.

Its namecontent: "1-3 Fuel cells fail to generate power with commands to present."

D is a disjunctive event not expanded here further in the **FT** fragment.

Its namecontent: "1-4 Fuel cell No 2 fails to generate required power when commanded."

E is a disjunctive event not expanded here further in the **FT** fragment.

Its namecontent: "1-5 Fuel cell No 1 fails to generate required power when commanded."

F is a disjunctive event not expanded here further in the **FT** fragment.

Its namecontent: "1-6 Fuel cell No 3 fails to generate required power when commanded."

Thus the explicit form of TE of **FT** is:

$$\begin{aligned} \mathrm{TE} &= A \vee B \vee C \\ &= A \vee B \vee (D \wedge E \wedge F). \end{aligned}$$

This is the disjunctive normal form [4, p. 33] (DNF) of TE. The conjunctive normal form [4, p. 33] (CNF) is:

$$TE = (A \vee D) \wedge (A \vee E) \wedge (A \vee F).$$

From which it is easily seen that TE has three independent roots:

$$\{A, D\}, \quad \{A, E\}, \quad \{A, F\}.$$

When CNF has a huge amount of factors (as in the case of the **FT**) then it is hard to determine CNF from the explicit form of the Boolean function of the **FT**.

Although E and F are considered originally identical, we treat them as different because of their content being different referring to Fuel cell No. 2 and No. 1, respectively.

The standard graphical representation of the fault tree seems to be quite out of date. Its drawback is that it is hard to overview. Due to the availability of Microsoft Windows® type explorers it is more natural to represent fault trees that way.

There are a number of firms dealing with risk assessment using fault tree methodology.

Recently Item Software® has added to their portfolio the new QRAS (Quantitative Risk Assessment) System package which has been funded by NASA since 1997.

QRAS is a comprehensive Microsoft Windows®-based software tool for conducting Probabilistic Risk Assessment. (`http://www.itemsoft.com`)

In this paper the original Apollo 13 fault tree is rewritten in Microsoft Windows® Explorer fashion. The computer code was developed at PROFES® LTD using the Profes + 4 software package. (`www.profes.hu`)

The Main Algorithm (FLORIAN).

Step 1: Set Level to 0 (`L = 0`).

Step 2: `L = L + 1`.

Step 3: If `L > Lmax` then exit.

Step 4: Set `e = 0`.

Step 5: Let `e = e + 1`.

Step 6: If `e > nEvt` then goto Step 2.

Step 8: If `e` is not prime then goto Step 5.

Step 9: If `Evt(e).Level` = `L` and `Evt(e).Parent` is active and disjunctive then passivate all of its active PrimeKids.

Step 10: If `Evt(e).Level` = `L` and `Evt(e).Parent` is active and conjunctive then passivate the `Evt(e)`'s best active PrimeKid with respect to the given Franklin property.

Step 11: Compute the logic value of the topevent. If it is "*True*" then exit.

Step 12: Goto 5.

Beside the main algorithm a number of sub-algorithms are developed at PROFES® to display the results properly on Windows-based personal computers.

Representation of Apollo 13 fault tree in Microsoft Windows® explorer fashion. In Fig. 2 below only the "short content" of the original Apollo 13 fault tree gate-texts are used as 1-1, 1-2, ... The original text can be reproduced from the cited Internet site.

The results.

The state representation. Within Profes, risk systems are characterized by the system states. By definition, the state of the system is the ordered set of all the primeevents' states. The state of an event e is "*True*" ("T" for short) if e is the case and "*False*" ("F" for short) otherwise. Diagrammatically *True/False* state of a primeevent is represented by an arrow directed upward/downward.

Composite events' state are represented by a distinctive shape of the arrow.

The suggested system prevention response to any failures

It is a mere incident that items in row 2 and 3 of column 2 are equal.

In the case of other fault trees the situation is different.

The state page. To each state there belongs a table showing the connection between prime event indices and their respective actvity and content. Fig. 6 below shows a fragment of case #3 in Fig. 5.

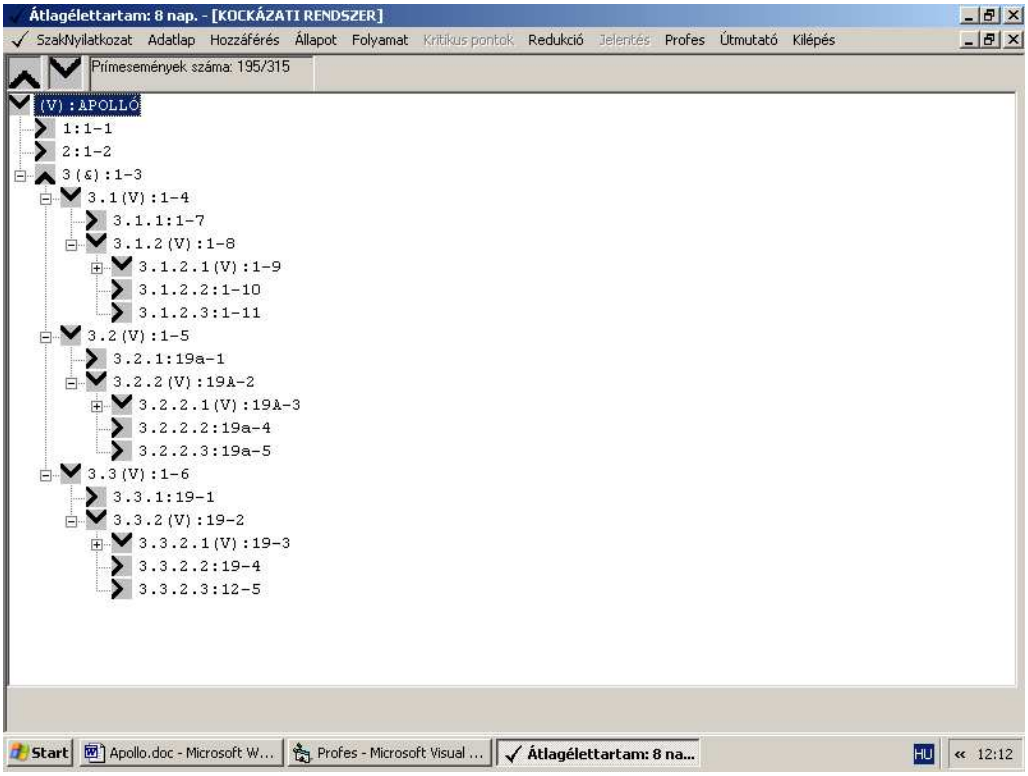

Fig. 2. The Fault Tree in the original Apollo 13 fault tree is rewritten in Microsoft Windows® Explorer fashion using the Profes + 4 software package.
Expanded in 3-level depth.
The sign "$>$" refers to primeevents
(With the permission of PROFES® `www.profes.hu`)

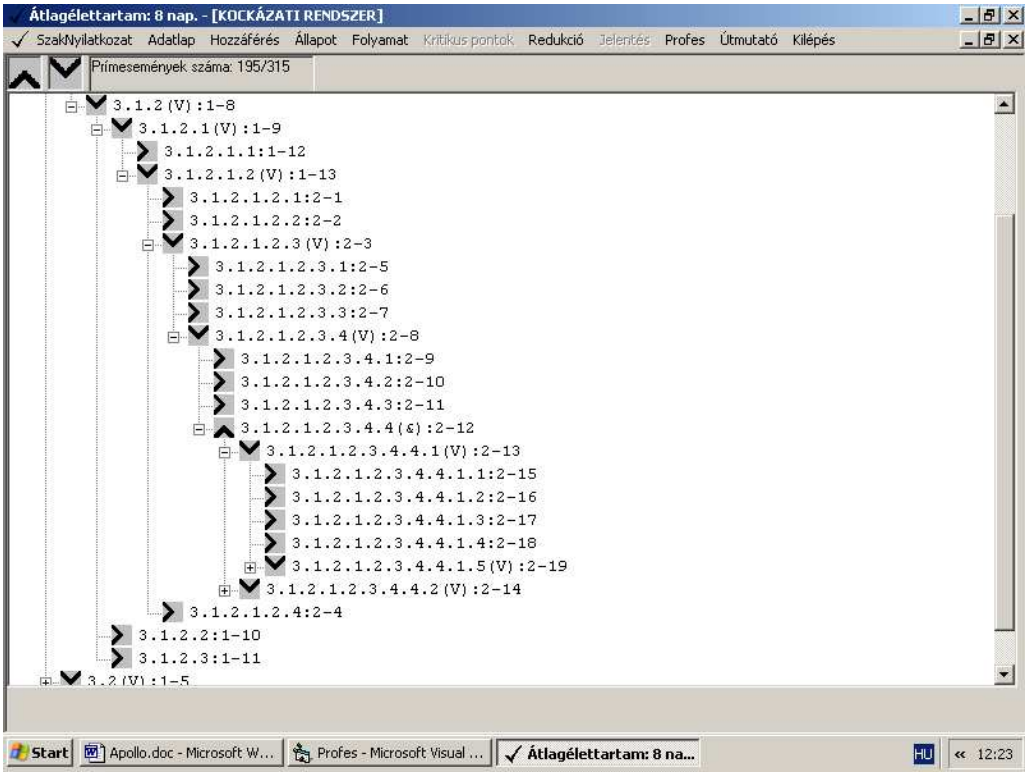

Fig. 3. The Fault Tree in the original Apollo 13 fault tree is rewritten in Microsoft Windows® Explorer fashion using the Profes + 4 software package.
Event 1-8 expanded in 6-level depth.
The sign "$>$" refers to primeevents
(With the permission of PROFES® `www.profes.hu`)

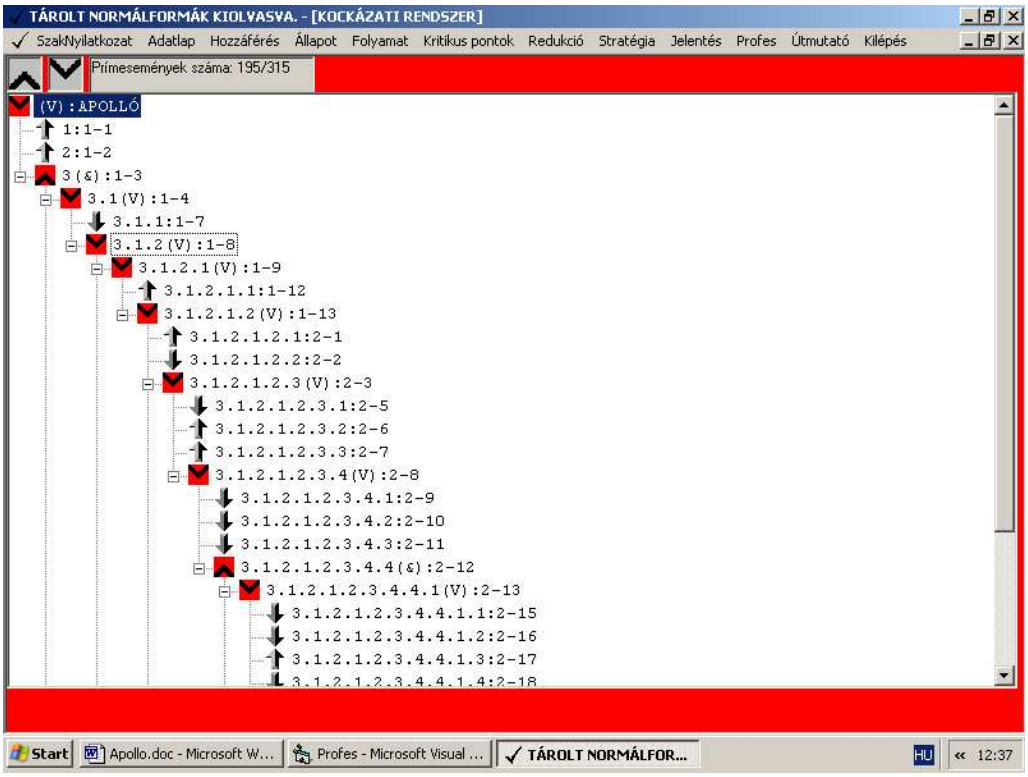

Fig. 4. The (active) state of the risk system, represented by the original Apollo 13 fault tree, represented by the Profes Software Package. Event 1-8 expanded in 6-level depth. The System state is defined by a random assignment of "*True*" to 82 prime events out of 195 ones. In this state, the top event is "*True*".
On clicking an arrow it turns around and the result of the system state is displayed immediately.
(With the permission of PROFES® `www.profes.hu`)

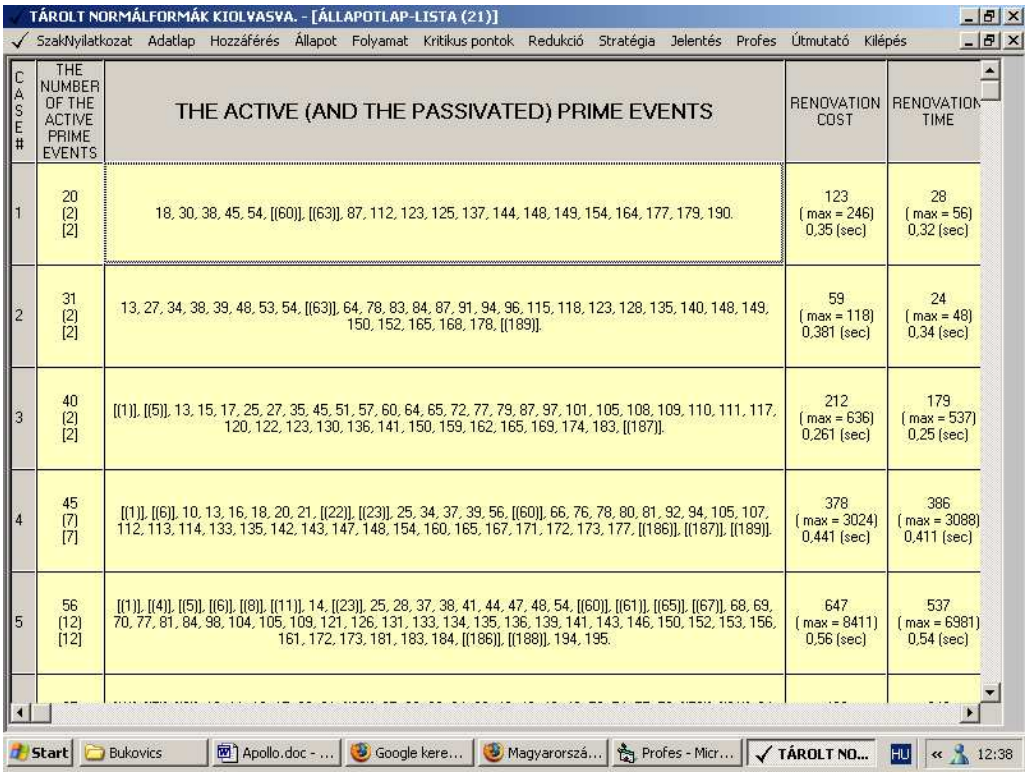

Fig. 5. The suggested prevention of the risk system, represented by the original Apollo 13 fault tree.

Excerpt of a table containing all of the system's active (topevent) states

Column 1: Serial number of the cases.

Column 2:

Row 1: The number of the randomly selected active prime events out of the total 195;

Row 2: The number of the recommended primes to passivate with minimal renovation cost;

Row 3: The number of the recommended primes to passivate with minimal renovation time.

Column 3: The indices of the active/passive primes yielding minimal cost/time after passivation in parenthesis/brackets respectively.

Column 4, 5:

Row 1: Total cost/time of the recommended passivation;

Row 2: Maximal possible cost/time of the recommended passivation;

Row 3: Runtime of algorithm FLORIAN.

(With the permission of PROFES® `www.profes.hu`)

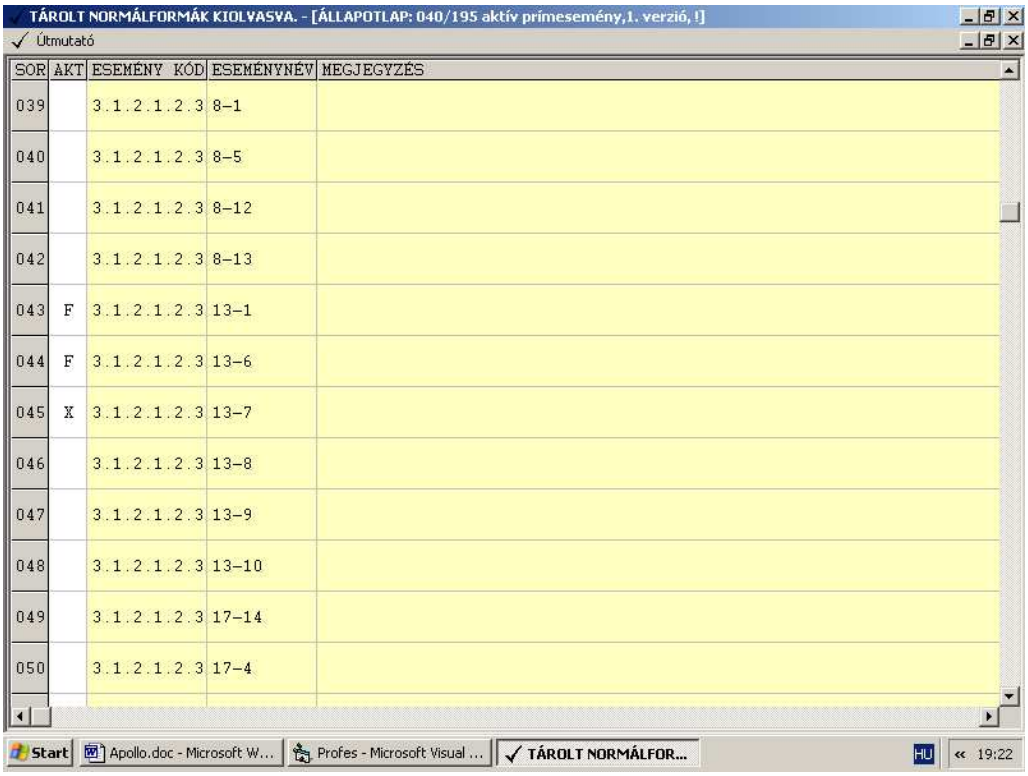

Fig. 6. Excerpt of the State Page corresponding to case #3 in Fig. 5.
Column 1: prime event index.
Column 2: Entry "X" – **active** prime event.
Entry "F" – **forlorn** prime event;
Empty entry – **passive** prime event;
Column 2: prime event name prefix.
Column 3: prime event name content.
(With the permission of PROFES® `www.profes.hu`)

REFERENCES

[1] Henley E. J., H. Kukamoto. Reliability Engineering and Risk Assessment. Prentice Hall, 1981.

[2] http://nssdc.gsfc.nasa.gov/planetary/lunar/ap13acc.html
and for the details:
http://drushel.cwru.edu/apollo13/appF-pt4.pdf.

[3] http://encyclopedia.laborlawtalk.com/Boolean_satisfiability_problem.

[4] Mézard M., G. Parisi, R. *Zecchina Science* **297**, Issue 5582 (2002), 812–815.

[5] Whitesitt J. E. Boolean Algebra and Its Applications. Addison-Wesley, Reading, Massachusetts, USA, 1961

[6] http://drushel.cwru.edu/apollo13/appF-pt4.pdf.

István Bukovics
National Directorate
General for Disaster Management
Ministry of the Interior, Hungary
e-mail: istvan.bukovics@katved.hu

Received February 15, 2006
Final Accepted March 12, 2007