Use of graph-based and algebraic models in lifecycle of real-time flight control software

A. Tyugashev[1]

[1]Samara State Transport University, 18 1st Bezymyanny Per., 443067, Samara, Russia

Abstract

Software faults are the causes for repeating catastrophes in modern space missions. There are various problems in lifecycle of flight control software including lack of adequate models of real-time control algorithms. Real-time control algorithms have the totally distinct nature in contrast to computational algorithms. The paper presents mathematical models suitable for analysis, design and formal verification phases of lifecycle of spacecraft's flight control software. Two kinds of models - graph-based and algebraic, are being described. These models were successfully introduced in computer aided software engineering toolset for design and verification of spacecraft's real-time onboard control software.

Keywords: real-time control algorithm; real-time flight control software; lifecycle of the flight control software; computer aided software engineering; graph based model; algebraic model

1. Introduction

The modern spacecraft usually has various onboard systems such as Energy Supply System, Motion Control System, Onboard Control System, Autonomous Navigation System, Telemetry System, Thermal Control System, etc. In turn, each onboard system consists of a set of devices, aggregates, sensors. We can state that spacecraft is a system of systems or complex of complexes. Functioning of all these devices should be coordinated both in time and logically. The real-time mode of functioning is a very important issue entailing more complex nature of required models to be used for adequate reflecting of features of onboard apparatuses. This aspect is also quite important when we deal with the problems connected with designing and implementation of dependable control system for the spacecraft. In accordance with the Ashby's Law of Requisite Variety, variety of onboard equipment's behavior requires variety of onboard control system. Today the control logic of onboard control system is implemented in onboard Flight Control Software. Roughly speaking, there is a special control software module for each onboard device or aggregate. There are also a lot of supplemental modules involved into organization of computational process, etc. This is an illustration to a structural aspect of complexity of modern spacecraft's onboard software. Another essential aspect corresponds to behavioral aspect of complexity. We can compare the onboard apparatus of the spacecraft with the orchestra with the string and wind instruments, drums, etc. But we need a conductor to get a symphony – violin should start at the right moment, next cello should start with accurately fulfilled delay, and so on. So, we need also a special sort of real-time software which will serve as an orchestra's conductor.

For example, there are about 500 software modules concurrently running at real-time mode onboard the modern spacecrafts manufactured by Samara Rocket and Space Center 'Progress' [1-4]. These modules have a different nature and objective – system, service, computational, support, etc. The very important part of the onboard flight control software is real-time control algorithms (analog of orchestra's conductor), or so named 'programs for complex functioning' (it means cooperative functioning of various onboard spacecraft's subsystems such as Motion Control System, Telemetry System, Energy Supply System, etc.). The purpose of this part of onboard software is to run needed 'functional' program modules, and execution of the needed commands by particular onboard equipment at 'right' moments of time with the proper considering of current situation. It is clear, that the overall success of space missions has a straight dependence on the correct functioning of program for complex functioning. This is an explicit example of mission-critical software. Herewith, the cost of the errors in such algorithms made at analysis, design and development phases of lifecycle is too high. The usual way for providing of reliability and quality of flight control software is many-staged testing and debugging with utilization of specially built test beds. This process is very labor and time consuming, but unfortunately it cannot guarantee the absence of the errors [3]. Unfortunately, we face with repeating catastrophes and faults in space missions caused by software errors. First well known incident happened in 1962 with Mariner-I space probe. Probably, the most expensive one was the explosion of Ariane-5 European Space Agency rocket during its first flight in 1996. The amount of loss was estimated more than 500 millions euros. We can also mention relatively recent widely discussed failures of onboard software of Mars Polar Lander, Mars Climate Orbiter and mars rovers.

What is a reason for it? Complexity of onboard equipment entails failures of devices (which can be parried by switching to reserve equipment executed by special software module, but it requires more complex control logic). Complexity of the spacecraft tasks entails more complex behavior (especially in abnormal situations). Complex behavior also requires complex control logic. Complex Control Logic entails 'broken phone effect' between onboard devices' specialists and programmers during coding it in Onboard Flight Control Software. Nowadays, complex control logic requires complexity of Flight Control Software. Complexity of Software means higher costs of software lifecycle. Moreover, complexity of software means more errors in software itself. Is it a vicious circle?

Summarizing, we can emphasize the following modern trends and problems in spacecraft control:

1. use of onboard computers as a main control system;

2. transfer of 'decision-making point' from Earth onboard;
3. growing of size and complexity of Flight Control Software (concurrent multi-tasking, hundreds of interacting modules, millions of lines of code);
4. software-based support of spacecraft's fault tolerance feature;
5. dozens of people including non-programmers, involved in lifecycle of Flight Control Software;
6. costs of Flight Control Software's lifecycle became a very significant part of space mission's total costs, moreover - design, development and testing of the Onboard Software often is a 'critical path' in network schedule of spacecraft 's producing as a whole;
7. labor costs of control system software's creation and testing is 10 times bigger than hardware related costs [3].

The very promising way in this area is application of formal verification methods [3,5]. Unfortunately, the main efforts in area in software formal verification is oriented to computational (data transforming) algorithms and software where adequate mathematical models and methods considering semantics of the algorithms were developed and researched. We can state the inadequacy of these models and methods to nature of real-time control algorithms consisted of not elementary computations but actions related to actuators and other spacecraft's hardware. Accordingly, the development of the adequate models for this kind of software is very important. The developed models can be utilized in methods of analysis, design and verification which reduce real-time lifecycle labor costs and provide needed level of dependability of real-time control algorithms. In [6], the 'basic' algebraic based mathematical model of real-time control algorithms was presented. This is a constructive model, allowing step-by-step building of control algorithms on the basis of 'elementary' actions – so called 'functional tasks', time intervals, and logical conditions. This paper presents some extended models which supplements and clarifies the basic model for further use with various purposes.

2. Real-Time Control Algorithms

The object of the study is real-time control algorithms. It should provide coordinated and well synchronized functioning of onboard spacecraft's systems containing various sensors, actuators, devices.

The very important features distinguishing the control algorithms from the data transformation algorithms are the following. First, we cannot correlate the function (in mathematical understanding), and the control algorithm. Moreover, the correctness of the control algorithm cannot be defined by the contents of the computer memory at the moment of algorithm's end. The correctness of the control algorithm depends on its behavior in full time interval of functioning. Moreover, the values of conditions during execution of data transforming program are totally defined by the input data while the values of the conditions to be considered in control algorithm, are unpredictable because they are formed by the parameters of physical processes in controlled object (for example, velocity of the spacecraft). Actions executed by the control algorithm also can change not only the data in memory of the onboard computer, but influence on the state of the controlled object. When we need, for example, land the spacecraft on the Mars, we need to implement the very complex sequence of the operations with the participation of various onboard devices and mechanisms – but all of them are under control of onboard software.

Unfortunately, the major efforts in the modeling of algorithms historically were focused on data transformers, since earliest models like Turing and Post Machines, Church's recursive functions and Markov's 'normal algoriphmes'. But if we want to apply the promising modern methods like formal verification or automated synthesis of the control software with the guarantee of its properness, we need the adequate semantic model for the control algorithm.

We can describe the following features of 'traditional' computational programs. Their main goal is transformation of input data to output data. The main components are the data transformers. The computational program correct if it successfully finishes (if input data is right) meanwhile output data matches specification. Structure of the traditional step-by-step sequential computational algorithm can adequately be represented by flowchart with begin and end(s) nodes. There are no time constraints and timer(s). Semantics could be adequately formalized by

1. axiomatic semantics [8] (Hoare, Floyd): *Pre {S} Post*;
2. denotational semantics describing mapping between sets [9].

Many control algorithms are being used by 'reactive' systems. The main features of such systems could be described as follows. The reactive system should right process the input event flow. The main components are actions (in contrast to computational algorithms). The following semantic models are used for reactive systems:

1. Kripke structures [10];
2. finite state machines - automata;
3. Petri Nets.

Other important features of algorithms used in reactive systems distinguishing them both from the traditional programs and real-time control systems:

1. there are no end of algorithms, use of infinite loop of event processing instead;
2. there are no time constraints and time scale (timers) - asynchronous nature;
3. correct, if algorithm implements required model and execute right actions as reaction on pre-defined events.

We deal with Time-Driven Real-Time Control Algorithms (RTCA). This kind of algorithms has very important distinguishes from the reactive systems. Time-Driven Real-Time Algorithm should implement required schedule(s) (the term 'cyclogramme' widely used in space industry). The main components of algorithm are actions.

Other essential features can be described as follows. There are begin and ends in contrast to reactive systems. There are 'hard' time constraints. There is time scale (timers) for quantitative description of time parameters. In other words, RTCA has a synchronous nature. This kind of algorithm correct if it execute right actions at right time (more precisely, right time moments with right considering of the current situation). Thus, the adequate semantic models should consider the stated features.

The known semantic models are

1. timed automata [11];
2. timed Petri Nets [12].

These models are based on the idea of 'state'. Unfortunately, if we try to apply this approach to real world system, we suffer because of 'state explosion' problem. This is why the author try to develop and use in CASE toolset another model based on idea of process. In some vision, this model utilizes more high level of abstraction allowing avoiding undue detalization and using in practice.

As it presented in [6], we can use the following set of tuples ('quads') for representation of semantics of real-time algorithm built from actions executed at particular time if the values of specified logical conditions are equals to 1:

$$UA = \{\langle f_i, t_i, \tau_i, \vec{l_i} \rangle\}, i = 1,..N$$

Each i-th quad in the above set describes one action executed by the real-time control algorithm; N is a number of actions executed. Here f_i is an identifier of the action, t_i – starting time of the action, τ_i - duration. Starting time and duration defined as integers, this is adequate time model in this case because the minimal time difference recognized by the control algorithm is a 'tick' of onboard clock generator. The set of elementary actions F should be previously defined, $f_i \in F$. Logical vector $\vec{l}_i$ specifies the combination of the conditions, allowing action f_i to be executed in time interval $[t_i;\ t_i + \tau_i]$. For example, logical vector can looks as follows $([\alpha_1=1], [\alpha_2=H], ...[\alpha_M=0])$. The 1 and 0 values recognized as true and false, and the third value 'H' means that this condition does not have an impact to execution of the action at specified time. The number of conditions actual for the control algorithm as well as the set L of the condition itself should be settled simultaneously with the set F of actions. We can interpret the logical vector in the model as an analog of the well known 'guard' conditions.

Described model have a very clear and intuitive visual representation looks like Gantt diagram. This was a reason because this model and models inherited from it were intended to use and successfully applied during development of the GRAFKONT/GEOZ [6] integrated development suite for automated design and verification of onboard flight control 'complex functioning' software.

But the 'basic' algebra of the control algorithms had the restricted descriptive power, for instance, there was no possibility to specify arbitrary time intervals between the actions, so it was necessary to introduce 'fictive' actions to taking in account the delays. And the time had the 'relative' nature only; we had no mechanism for binding the actions to the particular moment.

3. Methods

3.1. Extended algebraic model of the real-time control algorithm

There are some known models for parallel systems utilizing higher level of abstraction than state-based models. First of all, we should mention the following approaches:

1. process Algebras (Milner's CCS [14], Hoare's CSP [15], Bergstra's ACP [16], etc.);
2. temporal Logics (LTL, CTL, RTTL, etc.) [17-19];
3. Allen's Interval Logic [20].

Unfortunately, there are serious limitations for use of named models for Flight Control Software. If we talk about process algebras, we should underline the following. They provide just 'common' means for description of parallel execution of processes without opportunity to define coordination of processes' begins and ends. Process algebras do not support logical inference (reasoning about algorithms needed for verification purposes). And there are no any tools for description of various situations (conditioning for different variants of situations – regular, abnormal, emergency, etc.) of system's functioning. But in space missions we definitely need such instruments.

In contrast to process algebras, temporal logics natively have special means for description of concurrency of processes extended in time. But initially, there are no means for quantitative aspects of synchronization in temporal logics. In addition, the semantic models are not advanced enough for our practical purposes.

Allen's Interval Logic is a very interesting and promising approach providing means for description of all possible overlay of processes extended in time. But there are no means in Allen's logic for description of quantitative aspects of synchronization. Moreover, this approach also has no means for description of 'different branches' of Real-Time Control Algorithms.

The method we developed is free from the limitations described above. It was initially developed for description of complex Real-Time Control Algorithms with considering internal logic, different variants (branches) and situations. There are advanced means for description of synchronization of parallel processes. There are means for quantitative descriptions of synchronization both for relative and absolute timing.

The proposed model uses 'constructive' approach. We can construct new control algorithms using the existing ones by application of the set of operations. The basic operations introduced by Anatoly Kalentyev had only four operations – *CH*, *CK*, $\rightarrow$, and $\Rightarrow$. The extended algebraic model contains the following operations:

Table 1. Operations of the extended algebraic model of real-time control algorithms.

Name	Mean	Signature
CH	synchronization 'begin-begin'	$(UA_1, UA_2) \rightarrow UA$
CK	synchronization 'end-end'	$(UA_1, UA_2) \rightarrow UA$
$\rightarrow$	direct following	$(UA_1, UA_2) \rightarrow UA$
H	Overlay	$(UA_1, UA_2, \text{integer}) \rightarrow UA$
3A	parameterized following	$(UA_1, UA_2, \text{integer}) \rightarrow UA$
@	absolute time binding	$(UA, \text{integer}) \rightarrow UA$
$\Rightarrow$	qualification by the condition	$(\text{condition}, UA) \rightarrow UA$

'Begin-begin' synchronization applicable to the two control algorithms (denoted in the table above as UA_1 and UA_2) and forms a control algorithm which includes all quads from both UA_1 and UA_2, but with the correction of the start time of each action inherited from the UA_2. To make the correction, we should calculate overall starting time of the UA_1 and UA_2. It can be calculated as a minimum of the starting times t_i of the actions included into the UA: $t_{UA} = \min_{i=1..N} t_i$. After starting times for UA_1 and UA_2 will be found, we should calculate the difference $\Delta = t_{UA2} - t_{UA1}$. And finally we must add the difference to the all t_i inherited from the UA_2. As a result, we will have the control algorithm where all actions from the UA_2 will be shifted and the first action from UA_1 and UA_2 begins at the same time. The *CH* operation is transitive, associative but not communicative.

'End-end' synchronization operation *CK* has the same signature as *CH*, and its result also includes all quads from both arguments. Again, the actions inherited from UA_2 should be shifted by Δ. But the rule for calculation of the Δ is different. The latest action of UA_2 in resulting algorithm should ends at the time when ends the latest action of UA_1. So, we need calculate overall finish time *et* for UA_1 and UA_2 as follows: $\mathrm{et}_{UA} = \max_{i=1..N}(t_i + \tau_i)$. And in this case $\Delta = et_{UA1} - et_{UA2}$. The *CK* operation is transitive, associative, but not communicative like *CH*.

Direct following means that in the resulting algorithm the first action inherited from UA_2 starts when the latest action inherited from UA_1 ends. For this, we need make shift like in the cases above, but the rule is different. We need set the starting time of the earliest action of UA_2, as et_{UA1}. Difference $\Delta = et_{UA1} - t_{UA2}$ we then need to add to all starting times of actions in resulting algorithm, which are inherited from UA_2. In contrast to basic algebra of real-time control algorithms, initially proposed by A.A. Kalentyev, the extended algebraic model includes also the following operations.

Overlay operation *H* is similar to parameterized following *3A* operation. We form the control algorithm including all actions from the arguments UA_1 and UA_2 like for *CH* and *CK* operations, but applying difference is defined as the additional argument of the operation – integer number. The difference between *H* and *3A* is defined by the following. In the result of H the first action inherited from the UA_2 should starts before the end of the latest action inherited from UA_1 – so, we deal with 'overlay'. In case of *3A* operation, conversely, the earliest action inherited from the second argument, should starts after the end of the latest action inherited from UA_1, and plus one.

Absolute time binding operation allows setting the particular time as the starting time of the control algorithm (starting time of the earliest action). We should find the overall starting time for the existing UA, and then calculate the difference between t_{UA} and the second integer argument of the @ operation. After this, the difference Δ should be added to all starting times t_i of the all actions to be executed by the algorithm.

Finally, the 'qualification' operation has the UA and the condition as the arguments. We can write $(\alpha_1=0)\Rightarrow UA_1$, and it will mean that in all logical vectors in the UA, we need to update the corresponding component (initially all values for all conditions for all actions settles as '*H*', i.e. action is to be executed imperative).

This model can be successfully utilized for the purposes of specification and verification of the real-time control algorithms. For example, author's applied it at the corresponding stages of flight control software lifecycle for spacecraft [6].

3.2. Graph-based model

Anyway, if we want to solve the synthesis problem of the program, we need the adequate model with the more deep detailing.

The model in previous section specifies the control algorithm at the 'semantic' level. We can see parallel with the denotational semantics of the data transformation algorithms when we nominate the function to be calculated by the program. But the same semantics can be provided by different implementations. Moreover, it is well known fact that in case of software, the quality and characteristics – including efficiency, of the programs with the same semantics, can be quite different.

We need the models for the next degree of detalization. The very popular and effective in practice models in theoretical computer science are graph-based. They are applicable also at the stages of design and analysis of efficiency of implementation of flight control software as well. However, the known models require clarification and extension.

For example, the control flow graph is a very useful and popular model. But initially it describes sequential imperative program executed by the single CPU. Understandably, there are no any essences applicable for describing phenomena of time interval.

But the nature of the complex technical system likes modern spacecraft urgently requires concurrent (multi-tasking) model of computation. It is unsurprising that the onboard software of the modern spacecraft is executed by the control of multi-tasking operating system. Such systems have an application programming interface allowing one process to be started by another (fork). In many cases, onboard real-time operating system allows starting the process with the specified time delay or at absolute time.

This is the reason for defining the extended model – 'timed logical scheme' of the real-time control algorithm. First, we take the well known program control flow graph. In case of control algorithms, nodes with the single outgoing arcs will be associated with any actions executed by the algorithm. Nodes with the two outgoing arcs will be associated with the conditions formed not by the CPU flags only, but related to parameters of spacecraft motion, state of the onboard systems, etc.

We introduce also the special ‘weighted’ arcs. The weight specified by the integer, will denote the delay before the action associated with the following node be executed. The very important issue is that any ‘action’ node can have arbitrary number of outgoing ‘weighted’ arcs additionally to one ‘usual’ unweighted. The example of the timed logical scheme is presented in Figure 1 in alongside with the visual representation of the corresponding semantic model of control algorithm.

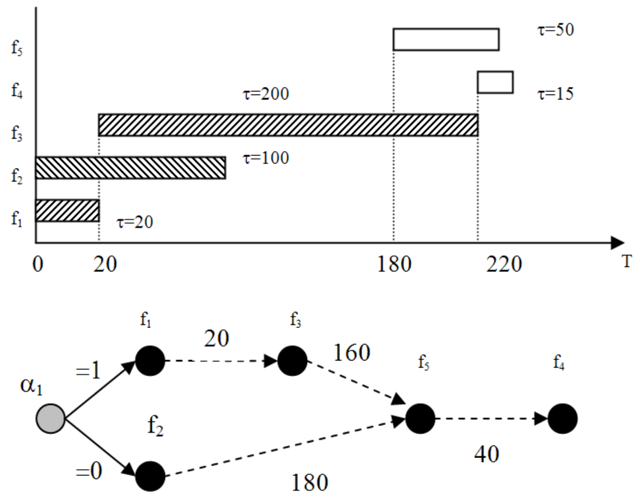

Fig. 1. Example of timed logical scheme.

The depicted UA is *{<f_1,0,20,(α_1=1)>, <f_2,0,100,(α_1=0)>, <f_3,20,200,(α_1=1)>, <f_4,180,50,(α_1=H)>, <f_5,220,15,(α_1=H)>}*. Different qualification by the logical vectors can be shown by different texture or color on visual representation of the control algorithm’s semantics. Timed logical scheme implements the semantic by the fixation of the ‘key’ time moments – 0, 20, 180, 200 when the algorithm must perform actions. This ‘activations’ are divided by relatively time delays: 20, 160, 40, 180 associated with the weighed arcs in the graph-based model. We can see the ‘qualified’ branches of the timed logical scheme with the corresponding *(α_1=1)* and *(α_1=0)* conditions (in this case, there is only the single condition in the logical vector). Then, at timestamp 180, these branches join again.

4. Results and Discussion

The presented models looks be much more adequate to the nature of the onboard spacecraft’s flight control software than approaches oriented to computational sequential algorithms. Nature of the presented models is quite corresponds to the domain of real-time control of technical complexes consisting of many subsystems, devices, aggregates, etc. The main components of control programs are actions which can be executed both by software modules and equipment. The conditions which influences the process of computation, also does not formed by the input data only, but permanently changing in accordance with the state of controlled object. The semantic model presented in the paper, initially oriented to this particularities.

These features provide possibility to potential application of these models in such area as SCADA systems, power plants, transport, etc. [7].

If we compare presented timed logical scheme of the algorithm with the timed automaton, for example, we will discover that despite there are possibility to describe time related issues, timed automata cannot be used to adequate and unambiguous descriptions of control programs. Timed logical scheme, conversely, initially was developed with this purpose and good corresponds to the factors of problem domain. Moreover, the features of real-time operating systems are taken into account.

5. Conclusion

The paper presents two extended models for representation of real-time control algorithms implemented by spacecraft’s flight control software. One model is algebraic and another is graph-based. Algebraic model can be applied for ‘high level’ semantic modeling of the real-time control algorithms. This is important because known models were oriented to computational algorithms and did not take in account the nature of real-time control systems. Semantic models can be used for the accurate and unambiguous specification of flight control software and then applied for formal verification, which is reviewed nowadays as very promising method around the world.

Another presented model is graph-based. It allows analyze the efficiency issues and can be successfully used at the design stage. Constructive nature of these models and their clear visual representations allow developing of GRAFKONT/GEOZ software toolset for specifying and verification of real-time control software. The toolset was introduced at Samara Space Rocket Center.

Acknowledgements

Author should say many thanks and acknowledgements to President of the Samara University Victor Soifer who granted him a possibility to research and develop the methods and tools for Russian space industry for many years.

References

[1] Kozlov DI, Anshakov GP, Mostovoy YaA, Sollogub AV. Control of Earth's Remote Sensoring Spacecrafts: Computer Technologies. Moscow: Mashinostroenie, 1998; 368 p. (in Russian)

[2] Kirilin AN, Anshakov GP, Akhmetov RN, Storozh DA. Spacecrafts Building. Samara: Agni Publishing House, 2011; 280 p. (in Russian)

[3] Tyugashev A, Ermakov I, Ilyin I. Ways to get more reliable and safe software in Aerospace Industry. Proc. Program Semantics, Specification and Verification: Theory and Applications (PSSV), Russia: Nizhni Novgorod, 2012; 121–129.

[4] Filatov AV,Tkachenko IS, Tyugashev AA, Sopchenko EV. Structure and algorithms of motion control system's software of the small spacecraft. Proceedings of Information Technology and Nanotechnology (ITNT-2015). CEUR Workshop Proceedings, 2015; 1490: 246–251.

[5] Holzmann GJ, Havelund K, Joshi R, Xu R-G, Groce A. Establishing flight software reliability: testing, model checking, constraint-solving, monitoring and learning. Annals of Mathematics and Artificial Intelligence 2014; 70(4): 315–349.

[6] Tyugashev AA. Integrated environment for designing real-time control algorithms. Journal of Computer and Systems Sciences International 2006; 45(2): 287–300.

[7] Tyugashev A. Language and Toolset for Visual Construction of Programs for Intelligent Autonomous Spacecraft Control. IFAC-PapersOnLine. 4th IFAC Conference on Intelligent Control and Automation Sciences. France: Reims, 2016; 49(5): 120–125.

[8] Hoare CAR. An axiomatic basis for computer programming. Communications of the ACM CACM 1969; 12(10): 576–580.

[9] de Bakker JW. Least Fixed Points Revisited. Theoretical Computer Science 1976; 2(2): 155–181.

[10] Schneider K. Verification of reactive systems: formal methods and algorithms. Springer, 2004.

[11] Bengtsson J,Wang Yi. Timed Automata: Semantics, Algorithms and Tools. Lectures on Concurrency and Petri Nets. Lecture Notes in Computer Science; 3098: 87–124.

[12] Zuberek WM. Timed Petri nets definitions, properties, and applications. Microelectronics Reliability 1991; 31(4): 627–644.

[14] Milner R. A Calculus of Communicating Systems. Springer Verlag, 1980.

[15] Hoare CAR. Communicating sequential processes. Communications of the ACM 1978; 21(8): 666–677.

[16] Bergstra JA, Klop JW. ACPτ: A Universal Axiom System for Process Specification. CWI Quarterly 1987; 15: 3–23.

[17] Pnueli A. The temporal logic of programs. Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS) 1977; 46–57.

[18] Emerson EA, Halpern JY. Decision procedures and expressiveness in the temporal logic of branching time. Journal of Computer and System Sciences 1985; 30(1): 1–24.

[19] Ostroff JS. Temporal Logic of Real-Time Systems. Advanced Software Development Series. Research Studies Press Ltd, England, 1990.

[20] Allen JF. Maintaining knowledge about temporal intervals. Communications of the ACM 1983; 26: 832–843.