Available online at www.sciencedirect.com

ScienceDirect

Transportation Research Procedia 61 (2022) 447–454

Transportation Research Procedia

www.elsevier.com/locate/procedia

XII International Conference on Transport Infrastructure: Territory Development and Sustainability

Towards Modeling and Verification of Eurobalise Telegram Encoding Algorithm

Sergey Staroletov*

Polzunov Altai State Technical University, 46, Lenina avenue, Barnaul 656038, Russia

Abstract

A eurobalise is designed to broadcast the current state of the railway to a passing train. Such a transmitter is activated with an approaching train and its goal is to dispatch messages called telegrams. In this article, we consider the encoding process of such telegrams. Since the CRC checksum building method is based on operations with polynomials over the Galois field modulo 2, we describe a proven library to work with polynomials. Next, we present the implementation of the encoder in a modeling language according to the open specification and then discuss its verification strategies using the Model Checking method.

Keywords: Eurobalise; encoding; galois polynomials; model checking.

1. Introduction

Our long-term goal is to increase interest in the use of formal methods while ensuring the reliability of cyber-physical systems. Examples of such systems can be real ones with which a person comes into contact most often. While human presence is still legally required to operate the systems in the automotive domain, there are examples of mass train systems functioning entirely autonomously (for example, metro lines in Paris (Behm et al, 1999) and Singapore (May, 2004). Accordingly, during their development, modeling and verification methods have been used for a long time to avoid the appearance of potential critical errors in production.

In this study, we consider the encoding process of telegrams that are broadcasting by eurobalises (transmitters installed on the train tracks at critical points to send current information on the railway state). This process is implemented in accordance with the open international standard adopted by Alstom, Ansaldo, Bombardier, Siemens and Invensys (Alstom et al., 2007). Our motivation is to apply formal methods as well as non-deterministic programming to check the operation of such systems as well as create verifiable models based on uniform requirements.

The contributions of the paper are (a) focus on the telegram encoding process according to the standard (b) show how to implement real algorithms in the Promela modeling language (c) discuss verification of a related polynomial library (d) use the Model Checking method to prove some properties of the encoder.

* Corresponding author. Tel.: +7-923-643-36-00.
E-mail address: serg_soft@mail.ru

10.1016/j.trpro.2022.01.073

2. Related work

Since cyber-physical systems are complicated, more and more adequate models are required to use in the trains domain. To discuss the application of formal methods in this area, since 2016, the International Conference on Reliability, Safety, and Security of Railway Systems is organized, and the meetings were successfully held in Paris (Lecomte et al., 2016), Pistoia (Fantechi et al., 2017) and Lille (Collart-Dutilleul et al., 2019).

In the railway area, there are also many publications on reliability assessment, among other things, safety of train motion (Pshinko et al., 2020), which are based on the construction of physical and mathematical models. However, these models may not take various non-deterministic changes in values of variables and are not able to prove the correctness of behavior with the number of cases in tend to infinity. Using dynamic logic extensions, it is possible prove the formalized properties of continuous systems, such as stability (Staroletov, 2021a), but the process of model encoding and proving is very challenging.

For cyber parts of cyber-physical systems, for example, communications protocols such as the exchange of a eurobalise with a train, we need to apply much better examined discrete models. Such models ultimately lead to the use of Büchi automata (Büchi, 1962) and Kripke structures (Kripke, 1963).

As for the checking of the operation of eurobalises, there are some works on modeling and simulating the physical medium of data transmission (Giuliani et al., 2016). In the work (Mendizabal et al., 2016), the authors turn to the implementation of reliable coding by using FPGA devices construction and applying the fault injection method.

Regarding proving the operations on polynomials, in the work (Lv et al., 2013) the authors present a method of formal verification of Galois field arithmetic circuits using efficient Gröbner basis reductions.

3. Methods

3.1. Model Checking

Model Checking (initially proposed by Clarke et al. (1986, 1999) is a verification method based on the construction of models. In this method, we build a model for the target software system and then automatically check it for compliance with some formal requirements. This approach is now applied primarily for communicating processes, distributed algorithms and protocols (Karpov, 2010). This method allows us to prove the correct operation of the model on all possible source data, all possible methods of switching processes and interactions (however, the models here are usually simplified, and the adequacy of the model still needs to be proved).

In Model Checking, it is necessary to build essentially two models:

- a model of the algorithm, which describes the intended functionality of the system;
- a model of system requirements, which defines formal system requirements, how it should lead, which invariants or restrictions should contain.

Note that both models should be adequate. This is the main problem in verification based on models because one can build a minimal model with simple requirements that are always satisfied, and confidence in their feasibility can make a false sense of reliability in the original software system.

In the verification process, the verifier tool traverses through all the model states and verifies contraventions of requirements. If the requirements are violated, the verifier shows a counterexample as a sequence of control states of the system violating the requirements.

3.2. Model Checking with SPIN

SPIN verifier (Holzmann, 1997) is a tool for model checking software systems. The SPIN stands for Simple Promela INterpreter where Promela is the name of input modeling language.

The SPIN system verifies not the programs themselves, but their models (Staroletov and Dubko, 2019). To build a model for an original program or an algorithm, the engineer (usually manually) builds a prototype of this program in Promela (Protocol MEta-LAnguage). The requirements for the model are expressed in the form of linear-time temporal logic (LTL) (Pnueli, 1997).

The main purpose of the language is to model interoperable systems and protocols, which allows us to use it for the implementation of encoder models.

In this work, we use the following language features (Spin, 2021):

- the usage of the C pre-processor;
- the presence of arrays;
- the presence of byte and bit datatypes;

- the presence of do-while loops and iterations;
- the presence of if clause including the non-deterministic choice (Staroletov and Shilov, 2019).

A well-known problem in Model Checking is the state explosion (Clarke et al., 2011): when we move forward from toy models to real-world ones, the number of internal states is exponentially growing and checking the models may not be completed in the foreseeable time and/or take too much memory. To neglect it in SPIN, its author proposed the Swarm technology (Holzmann et al., 2008), which runs a bunch of VT tests on the set of hashed states.

3.3. Operations on polynomials over $\mathbb{F}_2$

$\mathbb{F}2$ or GF (2) is the Galois field (Fröhlich, 2012; Mullen and Panario, 2013) with two elements. It comprises additive and multiplicative identities, respectively, denoted 0 and 1. Its addition is the same as the logical XOR operation; the subtraction is the same operation as the addition, while the multiplication is identical to the logical AND operation.

Operations with polynomials in this field are similar to operations over $\mathbb{R}$ field, but in our case, all operations are performed by modulo 2. Next, it is assumed that P1 and P2 are polynomials over $\mathbb{F}2$ having given degrees, and R is the result of the operation. Note that, we consider here original methods, not optimized ones (like, for example in work (Brent et al., 2008)) by reason of the generalization for further verification.

3.3.1. Addition

$$\begin{aligned}
&R_i = P_{1_i} + P_{2i} (\operatorname{mod} 2), i \in (\deg(P_1)..0], \deg(P_1) == \deg(P_2) \vee \\
&R_i = P_{1_i}, i \in (\deg(P_1)..\deg(P_2)] \wedge R_j = P_{1j} + P_{2_j} (\operatorname{mod} 2), j \in (\deg(P_2)..0], \deg(P_1) > \deg(P_2) \vee \\
&R_i = P_{1_i}, i \in (\deg(P_2)..\deg(P_1)] \wedge R_j = P_{1_j} + P_{2j} (\operatorname{mod} 2), j \in (\deg(P_1)..0], \deg(P_1) < \deg(P_2) \\
&\deg(R) = \max(\deg(P_1), \deg(P_2))
\end{aligned} \tag{1}$$

The resulting degree should be adjusted according to the actual degree as a position of first 1 for the cases like 1+1=0.

3.3.2. Multiplication

$$\begin{aligned}
&R_{i+j} = R_{i+j} + P_{1_i} \cdot P_{2j} (\operatorname{mod} 2), i \in (\deg(P_1)..0], j \in (\deg(P_2)..0] \\
&\deg(R) = \mathit{check}(\deg(P_1) + \deg(P_2))
\end{aligned} \tag{2}$$

Note for check (): the resulting degree should also be adjusted.

3.3.3. Division

$$\begin{aligned}
&Rem = P_1, R = 0, (P_1 < P_2 \vee \deg(P_1) < \deg(P_2)) \vee \\
&rd = \deg(P_1), qd = \deg(P_1) - \deg(P_2) + 1, \\
&k = Rem_{rd-i-j-1} / P_{2\deg(P_2)-1}, R_{qd-i-1} = k, \\
&(Rem_{rd-i-j-1} = Rem_{rd-i-j-1} + k \cdot P_{2\deg(P_2)-j-1} (\operatorname{mod} 2), j \in [0..\deg(P_2)), i \in [0..qd), \deg(P_1) \geq \deg(P_2) \\
&\deg(R) = r_{\deg}, r_{\deg} \in (rd..0] \mid \forall x \geq r_{\deg} : R_x == 0 \wedge R_{r_{\deg}-1} == 1 \\
&\deg(Rem) = r_{rem}, r_{rem} \in (rd..0] \mid \forall x \geq r_{rem} : Rem_x == 0 \wedge Rem_{r_{rem}-1} == 1
\end{aligned} \tag{3}$$

The results of division are a quotient (R) and a remainder (Rem) whose degrees are adjusted after calculating the long division. In the above, i and j are temporary variables that run through their values in a given interval with an increment of 1; at the same time, rdeg and rrem are used to iterate from the maximum possible degree to 0 and find a first 1 in the polynomial to specify the resulting degree. See also schemes of such algorithms in book (Cox et al., 2015).

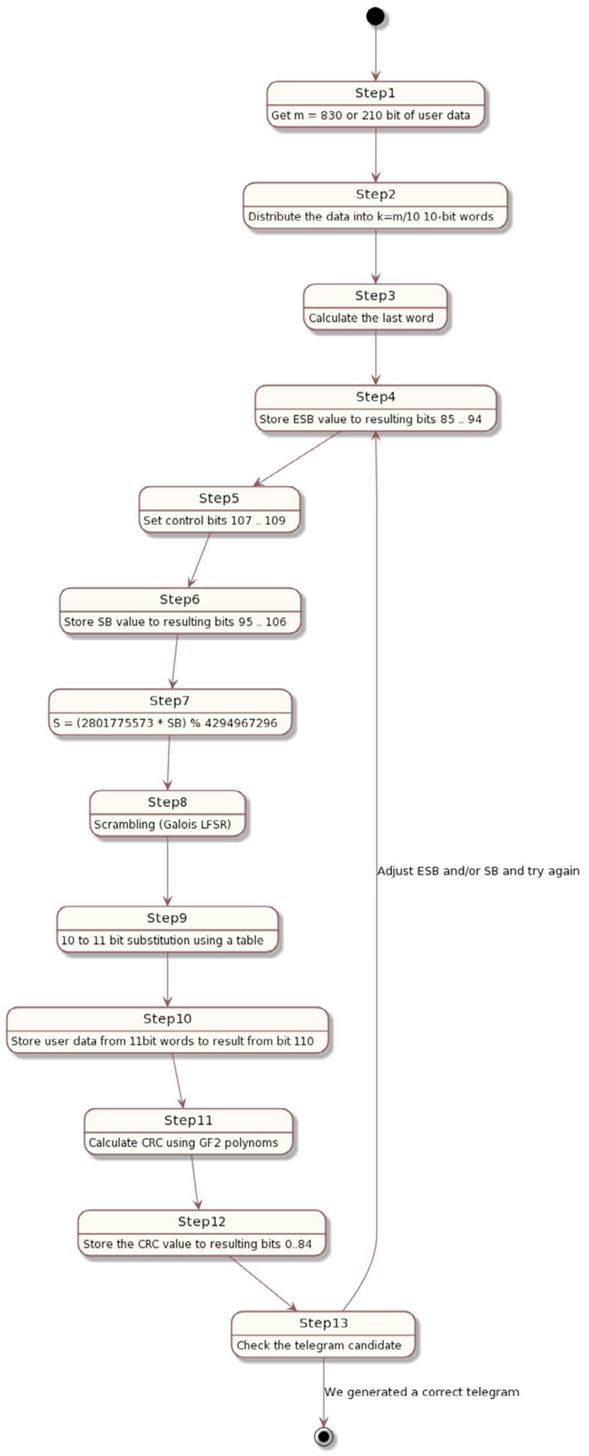
Step1
Get m = 830 or 210 bit of user data
Step2
Distribute the data into k=m/10 10-bit words
Step3
Calculate the last word
Step4
Store ESB value to resulting bits 85 .. 94
Step5
Set control bits 107 .. 109
Step6
Store SB value to resulting bits 95 .. 106
Step7
S = (2801775573 * SB) % 4294967296
Step8
Scrambling (Galois LFSR)
Adjust ESB and/or SB and try again
Step9
10 to 11 bit substitution using a table
Step10
Store user data from 11bit words to result from bit 110
Step11
Calculate CRC using GF2 polynoms
Step12
Store the CRC value to resulting bits 0..84
Step13
Check the telegram candidate
We generated a correct telegram

Fig. 1. Telegrams encoding algorithm according to the open standard.

3.4. Eurobalise telegram encoding process

The process is discussed in the open official document (Alstom et al., 2007). We depicted it in Fig. 1.

The encoding process begins with splitting user data into 10-bit words, then using a lookup table to replace them with 11-bit words. The data is then scrambled using Galois LFSR (Goresky and Klapper, 2002) and written to the result along with an initial state of the scrambler and some extra shaping bits. The result also stores the CRC value obtained due to the operation with polynomials over $\mathbb{F}2$ representing user data and prefitted values (4).

$$b_{84} \cdot x^{84} + .. + b_1 \cdot x + 1 = Rem_{f(x) \cdot g(x)}[b_{n-1} \cdot x^{n-1} + .. + b_{85} \cdot x^{85}] + o(x) \tag{4}$$

Where $f(x) = f_l(x), g(x) = g_l(x)$ and $o(x) = g_l(x)$ are for the long format, while $f(x) = f_s(x), g(x) = g_s(x)$ and $o(x) = g_s(x)$ are for the short format. These polynomials can be viewed in the following binary form (Alstom et al., 2007):

$$\begin{aligned} f_l &= 11011011111 \deg(10) \\ g_l &= 101110001000\,011100111001101001111010001011101101010100100011101110100001 0011 \deg(75) \\ \\ f_s &= 10110101011 \deg(10) \\ g_s &= 100111110111\;1001000011000010111111101111011111001010010010100011110001001011 \deg(75) \end{aligned} \tag{5}$$

Note that $g_l(x)$ and $g_s(x)$ from (5) satisfy some rule which implies that the three fold repetition of a short telegram meets the parity check of the long format (Alstom et al., 2007).

4. Results and Discussion

4.1. Verification of our polynomial library

When implementing a reliable library to work with $\mathbb{F}_2$ polynomials, we used the following steps:

1. We have implemented formulas (1) – (2) as a program in the C language.
2. We tested the implementation on several tests according to (Sharetechnote, 2021).
3. Next, we have implemented the algorithms in Promela, taking into account the Promela language features. The Promela implementation is based on the original C implementation.
4. We also tested the resulting program in the modeling language according to (Sharetechnote, 2021).
5. After that, we actually proceeded to the verification. We implemented a non-deterministic transition system using *od-do* operator and conditions that can be true simultaneously inside a non-deterministic *if* clause (Fig. 2).
6. During the verification run, we got several erroneous paths of execution, which made it possible to correct the implementation errors.

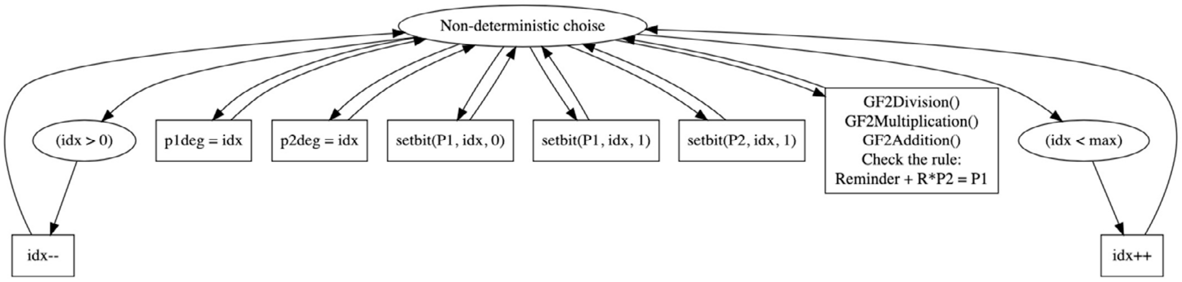

Fig. 2. Verification strategy for our polynomial library using a non-deterministic transition system.

Here, in the cycle, we change the size of the first or second polynomial, as well as their content. Since the dimensions of polynomials are finite, then during the verification, we will have a finite number of states. In this case, the verifier will not visit possible states with the same polynomial values due to the optimized DFS algorithm.

To verify all the three operations we need, we use the fact that if P1 and P2 are polynomials, then P1 = R * P2 + Rem (R is the quotient and Rem is the remainder). This equality can prove the correctness of our library operations.

Sketch code of the solution in Promela is presented below:

```
do
  //Change the index
  ::(idx > 0) -> idx--;
  ::(idx < max) -> idx++;
  //Change the degrees
  ::true -> p1deg = idx; //multiple true => the non-deterministic choice
  ::true -> p2deg = idx;
  //Put 1/0 at some place in polynomials P1, P2
  ::true -> setbit(P1, idx, 0);
  ::true -> setbit(P2, idx, 0);
  ::true -> setbit(P1, idx, 1);
  ::true -> setbit(P2, idx, 1);
  ::true -> {
     // GF2Division()
     // GF2Multiplication()
     // GF2Addition()
     // Check the rule Reminder + R*P2 = P1
  }
od
```

4.2. Modeling and verification of telegram encoding

In this work, we implemented in Promela the encoding process described in Section 3.4 and specified in the standard (Alstom et al., 2007). Since the language is somewhat similar to C, the implementation of this algorithm as a whole does not cause problems. To store data, we used byte arrays and created macros for convenient access to specific bits of information.

One of the implementation problems was the lack of unsigned int data type in Promela, so we had to look at the bit representations of such numbers and convert them to signed format.

As a strategy for verifying the coding process, we propose to use the rule "for a given user information, sometimes it is possible to create a valid telegram" that can be encoded in LTL. The crucial thing is that the telegram after encoding should be validated according to the standard. Here, we should check some pre-defined properties allowing the decoder to perform correct error-resistant decoding of this telegram and ensure that both short and long telegrams can be correctly decoded. For this, two sets of bits are used, which affect the scrambling parameters and user data shape.

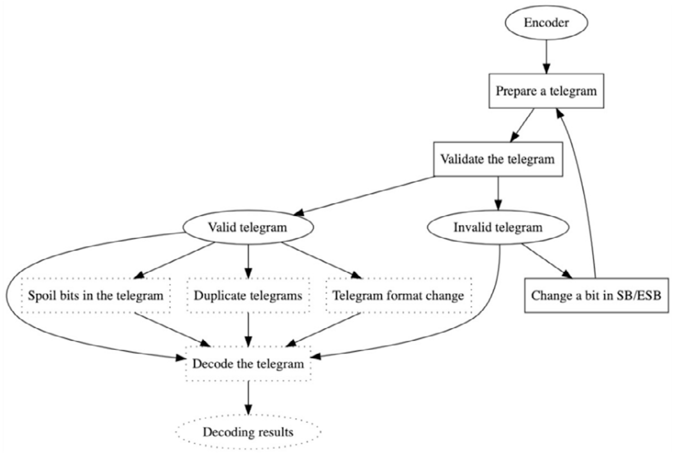

Fig. 3. Our verification strategy for encoding/decoding.

Thus, we come again to the non-deterministic programming. We randomly select these bits and start the encoding process. Next, we validate the obtained telegram candidate. If such validation is done, our LTL property holds, and it means that this algorithm is valid for a given user message.

Our model is freely available on GitHub (Staroletov, 2021b).

5. Conclusion

In this article, we examined the use of modeling languages and non-deterministic programming for solving the problem of encoding telegrams in modern railway networks.

We can note that such algorithms are well specified in the Promela language because it was created specifically for modeling data transmission protocols.

We have learned how to verify the correctness of encoding according to the validation requirements from the industrial standard.

In the future, we are going to implement a model for the eurobalise telegram decoder and verify both encoding and decoding processes by inserting random data into telegrams, duplicating them, using short telegrams instead of long ones, and so on (the strategy is depicted in Fig. 3).

References

Alstom, Ansaldo, Bombardier, Invensys, Siemens, Thales, 2007. ERTMS / ETCS – Class 1. FFFIS for Eurobalise. https://www.era.europa.eu/sites/default/files/filesystem/ertms/ccs_tsi_annex_a_-_mandatory_specifications/set_of_specifications_1_etcs_b2_gsm-r_b1/index009_-_subset-036_v241.pdf

Behm, P., Benoit P., Faivre, A., Meynadier, J., 1999. Meteor: A successful application of B in a large project. In: International Symposium on Formal Methods. Springer, 369-387.

Brent, R. P., Gaudry, P., Thomé, E., Zimmermann, P., 2008. Faster multiplication in GF(2)[x]. In: International Algorithmic Number Theory Symposium. Springer, 153-166.

Büchi. J. R., 1962. On a decision method in restricted second order arithmetic. In: Logic, Methodology and Philosophy of Science (Proc. 1960 Internat. Congr.), Stanford Univ. Press, Stanford, Calif, 1-11.

Clarke, E. M., Emerson, E. A., Sistla, A.P., 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transactions on Programming Languages and Systems (TOPLAS). 8, 244-263.

Clarke, E. M., Klieber, W., Nova cek, M., Zuliani, P., 2011. Model checking and the state explosion problem. In: LASER Summer School on Software Engineering. Springer, 1-30.

Clarke, E., Grumberg O., Peled, D. A., 1999. Model checking, MIT press.

Collart-Dutilleul, S., Lecomte, T., Romanovsky, A. (Eds.), 2019 Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification: Third International Conference, RSSRail 2019, Lille, France, Proceedings, 11495, Springer.

Cox, D., Little, J., O'Shea, D., 2015. Ideals, varieties, and algorithms. Undergraduate Texts in Mathematics, Springer.

Fantechi A., Lecomte, T. and Romanovsky, A. (Eds.), 2017. Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification: Second International Conference, RSSRail 2017, Pistoia, Italy, Proceedings, 10598, Springer.

Fröhlich, A, 2012. Galois module structure of algebraic integers, 1. Springer Science & Business Media.

Giuliani, F., Ottavi, M., Cardarilli, G. C., Re, M., Di Nunzio, L., Fazzolari, R., Bruno, A., Zuliani, F., 2016. Design and characterization of a high-safety hardware/software module for the acquisition of eurobalise telegrams. In: 2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), IEEE, 111-114.

Goresky, M., Klapper, A. M., 2002. Fibonacci and Galois representations of feedback-with-carry shift registers. IEEE Transactions on Information Theory. 48, 2826-2836.

Holzmann, G. J., 1997. The model checker SPIN. IEEE Transactions on software engineering. 23, 279-295.

Holzmann, G. J., Joshi, R., Groce, A., 2008 Swarm verification. In: 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, IEEE, 1-6.

Karpov, Y. G., 2010. Model checking. Verification of parallel and distributed program systems (in Russian), BHV-Petersburg, Saint Petersburg, ISBN 978-5-9775-0404-1.

Kripke, S. A., 1963. Semantical analysis of modal logic I normal modal propositional calculi. Mathematical Logic Quarterly. 9, 67-96.

Lecomte, T., Pinger, R., Romanovsky A. (Eds.), 2016. Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification: First International Conference, RSSRail 2016, Paris, France. Proceedings, 9707, Springer.

Lv, J., Kalla, P., Enescu, F., 2013. Efficient Gröbner basis reductions for formal verification of Galois field arithmetic circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 32, 1409-1420.

May, A., 2004. Singapore: The development of a world class transport system. Transport Reviews. 24, 79-101.

Mendizabal, J., Solas, G., Valdivia, L.J., de Miguel, G., Uranga, J., Adin, I., 2016. ETCS's eurobalise-btm and euroloop-ltm airgap noise and interferences review. In: International Workshop on Communication Technologies for Vehicles. Springer, 27–39.

Mullen, G. L., Panario, D., 2013. Handbook of finite fields, 17. Boca Raton: CRC Press.

Pnueli, A., 1977. The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), IEEE, 46-57.

Pshinko, O. M., Ursulyak, L. V., Zhelieznov, K. I., Shvets, A. O., 2020. To the problem of train running safety. IOP Conference Series: Materials Science and Engineering, 985. IOP Publishing, 012014.

Sharetechnote, 2021. GF (2): Galois Field 2. http://sharetechnote.com/html/Handbook_Communication_GF2.html/(accessed 20 September 2021).

Spin, 2021. Spin Version 6 – Promela Grammar. http://spinroot.com/spin/Man/grammar.html/(accessed 20 September 2021).

Staroletov, S., 2021a. Automatic proving of stability of the cyber-physical systems in the sense of Lyapunov with KeYmaera. In: 28th Conference of Open Innovations Association. IEEE, 431-438.

Staroletov, S., 2021b. Eurobalise encoder model. https://github.com/SergeyStaroletov/PromelaSamples/blob/master/Eurobalise.pml (accessed 20 September 2021).

Staroletov, S., Dubko, A., 2019. A method to verify parallel and distributed software in C# by doing Roslyn AST transformation to a Promela model. System Informatics. 15, 13-44.

Staroletov, S., Shilov, N., 2019. Applying model checking approach with floating point arithmetic for verification of air collision avoidance maneuver hybrid model. In: International Symposium on Model Checking Software. Springer, 193-207.