https://www.sworldjournal.com/index.php/swj/article/view/swj18-01-054
DOI: 10.30888/2663-5712.2023-18-01-054

УДК 004.415.2.043

TECHNOLOGY OF CREATING RELIABLE PROGRAMS
ТЕХНОЛОГІЯ СТВОРЕННЯ НАДІЙНИХ ПРОГРАМ

Holub S.V. / Голуб С.В.
d.t.s., prof. / д.т.н., проф.
ORCID: 0000-0002-5523-6120
Salapatov V.I. / Салапатов В.І.
c.t.s., as. prof. / к.т.н., доц.
ORCID: 0000-0001-7567-637X
Cherkasy State Technological University, Cherkasy, Shevchenko 460, 18006
Черкаський державний технологічний университет, м. Черкаси, Шевченка 460, 18006

Abstract. *This work deal with the creation of program models with the help of their description by means of temporal logic. As a result, a program model is created in the form of a non-deterministic finite automatons. Using this model, you can create programs in any procedural programming language.*

Key words: *Kripke structure, program models, predicates, database, primitive-monitor, primitive-protocol.*

The problem of creating reliable error-free programs is still quite important and relevant. This is especially important for management systems in aviation, military affairs, financial and banking affairs and many other areas. Errors in control programs can lead to accidents and disasters, as well as other irreversible processes. The main source of errors in programs is the lack of handling of possible branches in programs, as a result of which programs performed unexpected actions. It is almost impossible to prove the correctness of the programs, especially when it comes to parallel programs. As you know, the scientific research of Floyd and Hoare [1], as well as Robin Clark [2] in this direction did not find practical application. Therefore, it is expedient to create program models and prove the correctness of these models. The correctness of the models is proved by the correspondence of the models to their description. Recently, a similar technique has been used by the Model Checking technology [3], which involves the creation and subsequent verification of models. But even when building a model using the MOLEL CHECKING technology [3], the model must be verified using a special verifier program. It is more natural to describe the model directly using predicates, as groups of algorithmists used to do in terms of first-order logic. This article is devoted to the description of models of programs using predicates and creation of these models based on the description.

The method of describing models of program is based on a mathematical model in the form of a Kripke structure [3]. As already mentioned, of all the technologies for creating reliable programs, the most successful is the MOLEL CHECKING technology, which is now implemented in industrial operation. This model has the following appearance.

$$M = (S, S_0, R, AP, L)$$

S – is the set of states of the model,
S_0 – is the set of initial states of the model,
R = S x S – is the complete relationship between S, that is, transitions from one state

to another, which may be possible,
AP - is a finite set of predicates,
$L = 2^{AP}$ – is a marking function, where each state defines a set of true possible combinations of predicates.

The main elements of describing models are predicates, which describe individual actions of algorithms under certain conditions. The main elements of describing models are predicates, which describe individual actions of algorithms under certain conditions. In this regard, a full description of the algorithm using predicates is more appropriate. In the future, such a description is transformed into a computer representation, which creates a model according to the description. At the same time, the model will be created exactly according to the description and therefore it does not need further verification. It is convenient to present predicates separately in the form of a logical part and a meaningful part, which should be highlighted in curly brackets for the logic and in square brackets for the meaningful part. The most convenient internal representation of models in memory is a representation in the form of a database, which consists of a description of all predicates and the relationships between them [4] and is presented in Fig. 1. Such a representation will graphically correspond to the block diagram of the algorithm of program. Both the model and the block diagram of the algorithm of program will be fully consistent with the previous description and will actually show the correct block diagram of the program. When describing models, all possible ramifications are taken into account as fully as possible, so software models are created complete and accurate. According to the obtained model, it is much easier to create a reliable program without errors, because if the models are correctly described, the model itself will be correct too.

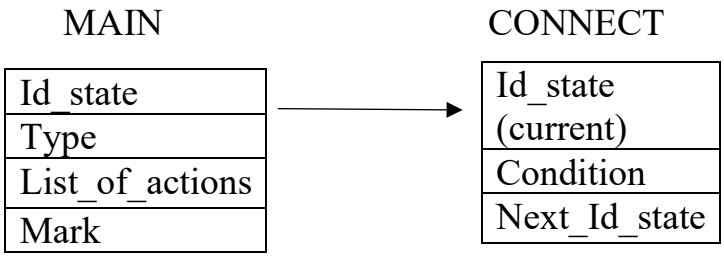

Fig. 1. The structure of the program model database

It should be noted that the proposed technology, like MOLEL CHECKING, involves the description of parallel programs. At the same time, in case of transformation of all meaningful and conditional parts of the predicates of the program model into any imperative programming language. At the same time, it is possible to convert all meaningful and conditional parts of predicates of the program model into any imperative programming language. In our case, only the following temporary operators are used from temporal logic: *pUq* and *Nr*, where *p* is a predicate that will be true until (*U*) predicate *q* becomes true, and next (*N*) predicate *r* should be.

Such a graph will be verified in advance, as it will fully correspond to the description and will actually represent the block diagram of the program. According to the obtained model, it is much easier to create a reliable program without errors. As already mentioned, the main source of errors is that not all possible branches can be

taken into account and implemented according to traditional technology. When describing the models, such branches are taken into account as fully as possible, therefore, the program models are created complete and accurate. Let's list the operations on predicates that will be used in the description.

That is p, ¬ p, p | q, p & q, pUq, Nr. List the operations on the predicates that will be used in the description. In our case, the usual logic is extended due to operations U and N, which was already discussed above. All program blocks can be implemented with the help of temporal operators: linear, branched and all types of cycles. Linear blocks represent assignment, input, output operators, as well as various functions that are most often used for a certain class of tasks. Using this logic, branching can be described by the following statement pUq, where p is a predicate before branching and q is a predicate with a direct branching condition. In the case of branching, such as the predicate DO CASE q – this set of branches can be represented as pU(q1 | q2 | q3 |…| *otherwise*). Otherwise, the *otherwise* keyword indicates a transition if no conditions are met. Loops can also be implemented using regular predicates that will be executed if the condition of such predicates is true, and the execution of part of the predicate will be executed in the loop while the loop condition is true. As soon as the loop condition becomes false, the loop terminates.

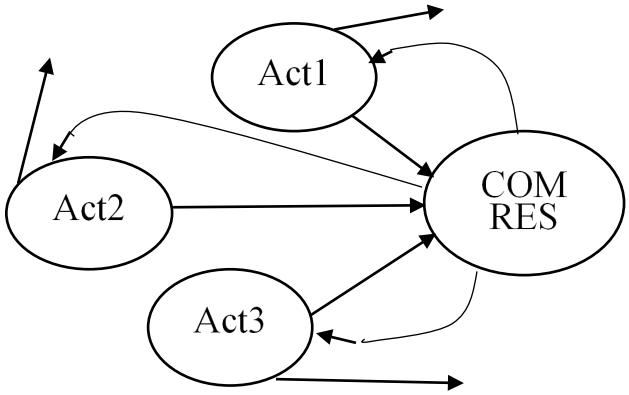

Fig. 2. The structure of the program model database

If the monitor functions for capturing shared resources and their subsequent release are built into the execution states of Act 1, Act 2 and Act 3, then such states can be represented as shown in Figure 3, where each of these states has access to the shared resource Com and to semaphore variable Sem. Data of type Com and Sem must be public for all states Act1, Act2, Act3. Figure 3 shows a conditional display of states that will be served by the monitor.

The reserved words *true* define unconditional execution, and *actions* indicate actions in parallel branches. A primitive *protocol* that defines the parallel the parallel actions in parallel branches. A primitive *protocol* that defines the parallel execution of individual branches of the program, using temporal logic operators, can be represented as ({busy=false} [actions 1] & {busy=false} [actions 2] &{busy=false} [busy=false] &.. | wait) {busy=true} [joint actions]. Graphically it looks as it shown in figure 4.

The technology for creating reliable programs MODEL CHECKING is the most successful technology and is now implemented in industrial operation. This model has the following appearance.

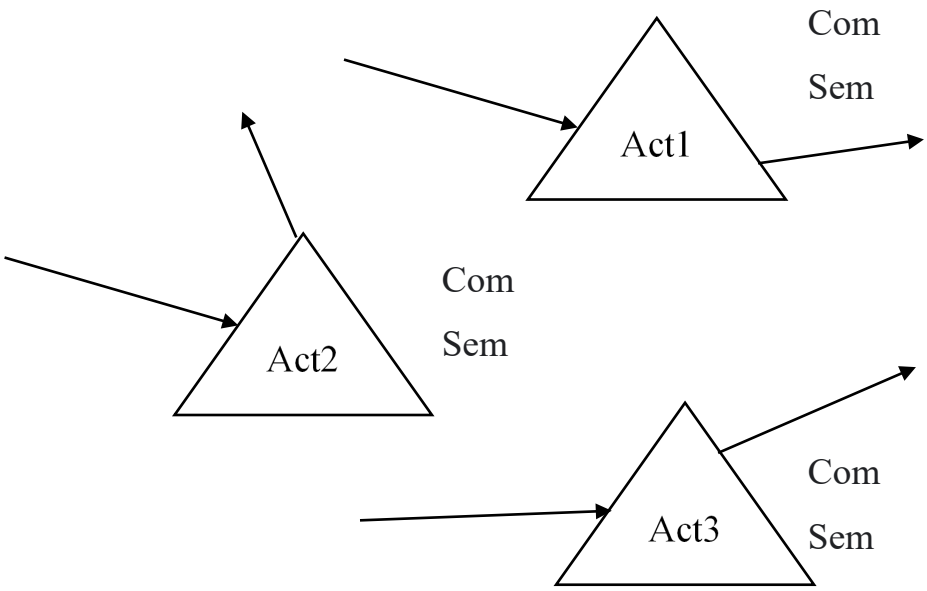

Fig. 3. Designation of monitor states for parallel program branches

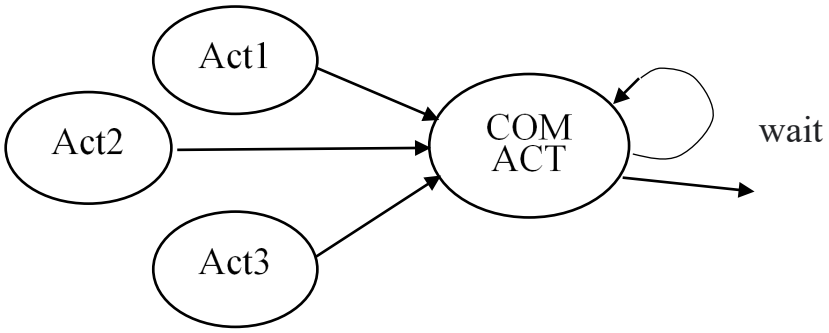

Fig. 4. Graphic display of the protocol

Currently, this technology is being introduced into the educational process of the Cherkasy State University of Technology, work is underway to transform the description of software models into programs in the target programming language. Developments in this direction will make it possible to significantly increase the reliability of programs and, with the correct description of their models, get closer to error-free programs.

References.

1. C. Hoare. Communicating sequential processes. Prentice Hall International. ISBN 978-0-13-153271-7. 238 p. 2015.

2. Robin Milner: A Calculus of Communicating Systems, Springer Verlag, ISBN 0-387-10235-3. 171 p. 1980.

3. Doron A. Peled, Michael Wooldridge. Model Checking and Artificial Intelligence: 5th International Workshop, MoChArt 2008, Patras, Greece, July 21, 2008. ISBN10 364200430X, ISBN13 9783642004308.

4. Салапатов В.І. Порядок опису і обробки графа автоматної моделі програми. ISSN 1028-9763. Математичні машини і системи. 2021. № 3. С.121-125.

Анотація. *У цій роботі йдеться про створення програмних моделей за допомогою їх опису засобами темпоральної логіки. У результаті створюється програмна модель у вигляді недетермінованого скінченного автомату. Використовуючи цю модель, можна створювати програми на будь-якій процедурній мові програмування.*

Ключові слова*: структура Крипке, програмні моделі, предикати, база даних, примітив-монітор, примітив-протокол.*