TECHNOLOGY FOR MODELLING SOFTWARE SYSTEMS BASED ON NON-DETERMINISTIC FINITE AUTOMATOS

Docent Salapatov V., PhD
Faculty of Computer Engineering, Intelligent and Control Systems – Chercassy National University named by B. Khmelnytsky, Ukraine
v_salapatov@ukr.net

Abstract: The technology of constructing models of software systems using non-deterministic finite state machines is offered. In this case, each vertex of the automaton model (state) is matched by the output function and the transition function. The first determines the actions performed in this state, and the second defines the conditions for the transition to other states. Bypassing an automated model tree of a program to implement it in some programming language, you can create your own program.

Keywords: MODELLING SOFTWARE SYSTEM, NON-DETERMINISTIC FINITE AUTOMATOS, OUTPUT FUNCTION, FUNCTION OF TRANSITION

1. Introduction

The problem of the correctness and reliability of programs, which is related to their modeling, verification of program models, subsequent validation and realization, is very urgent and actual. Specialists have long been working on a possible formal description of the algorithm, describing the program in a special way. On the other hand, it is important to prove the correctness of such software and hardware systems. Existing software modeling, verification and development technologies do not guarantee their accuracy and reliability. This is confirmed by recent events accidents of two BOINGs that had malfunctioning control program. Recently used MODEL CHECKING technology requires the use of special verification programs, which are not always available.

In this article proposes program modeling technology using non-deterministic finite automats. The program model description allows you to create a graph where each of its states will be matched by an exit function and a transition function. The exit function determines the list of actions to be performed in this state, and the exit function determines the transitions to other states from the current state.

This approach allows to build the program model as precisely as possible according to the technical specification. After minor modification, the program model can be used to create programs themselves. This process is subject to automation to finally create the reliable programs.

2. Software system modelling technology

As already mentioned, a non-deterministic finite state machine is chosen as the basis for modeling [1] M = (S, S0, R, AP, L) where S is the finite non-empty set of states of the automaton; S0 is a finite set of initial states of an automaton (S0 ⊆ S); R ⊆ S x S is the total relation to S, that is, transitions from one state to another are possible; AP is a finite set of atomic predicates; L: S → 2AP is a markup function where each state of the map L defines a set of true predicates. Predictors in our case are specific actions (such as a sequence of calculations) that may be true or false.

This model naturally describes the algorithm of any complex program. The automat is transferred from one state to another under certain conditions, which are defined by the model description for each state. The description of the algorithm defines the logical conditions for such transitions. An undetermined finite state automat involves multiple transitions from one state to another. The description of the algorithm defines the logical conditions for such transitions [2]. The description of the program algorithm consists of two parts: a logical one, which determines the relationship between the states of the machine and the executive, where each state determines the sequence of certain actions in the form of predicates. The transition from the state Si to the state Sj is possible if the condition P1 is true, i.e. Sj (P1) → Sj. As a result of the description of each state of the automaton model, we will have several possible transitions from each of these states. In this way, each status will be linked to a list of conversions to other states. Here is an example of a fragment of a automat (Figure 1) with a limited number of states.

![Figure 1. Example of fragment of a automat](image_url)

As can be seen from this figure, states A1, A2, A4 are associated with several of the following states. Thus, each of the states A1, A2, A4 has a list of transitions to the other states. For state A1, the list consists of transitions to states A3 and A4. For state A2, it includes transitions to states A4, A5, and A6. When all the states of the automaton model are fully defined, that is, their output and transition functions will be known, we will have a complete description of the model according to the specification.

After possible modification and some changes to the model, you can start creating your own program. To do this, you must bypass the tree of automat of the program model in all possible directions. When processing all possible branches of the junctions from each current vertex, it is necessary to make a return to the previous state and continue bypassing the tree from the next branch of the branch at the previous level. That is, upon completion of the next branch of the automaton tree, a step back to the previous vertex at the last transition is performed. This process is displayed as follows.

Si (P1) → Sj; Sj ← (P1) Sj; Si (P2) → Sj+1

Here, the transition from the vertex Si under the condition P1 to the vertex Sj is shown. If this completes the bypass of the next branch in vertex Sj, it returns to the previous vertex Si due to the condition P1. The next exit from the vertex Si under the condition P2 is revised and the transition under this condition to the vertex Sj + 1 is made. The vertex will be considered fully processed if all its exits are noticed. If such a bypass of the tree will reveal a transition to a fully processed vertex, then you must immediately return to previous vertex and the process must continue to bypass. This is necessary to avoid repeated bypasses for the treated top. If such a bypass of the tree will reveal a transition to a fully processed vertex, then you must immediately return to previous vertex and the process must continue to bypass. This is necessary to avoid repeated bypasses for the treated tip.
In the process of scanning, the automaton tree realizes the content of the predicates for each vertex in the form of a set of actions in one of the procedural programming languages. (for example C, PASCAL, Java etc.). The most appropriate method for implementing this method of program construction is to use a stack mechanism. The stack is required to return to the previous vertex of the graph. The stack is required to return to the previous vertex of the graph. As already stated, for each state of the automaton model in the process of model construction, a list of predicates that implement the desired program algorithm for a certain transition to other states is fixed. And the number of returns will be equal to the number of branches in the list of each vertex. The sign of the end of the model tree bypass is the absence of fully process peaks. The process described is easy to automate. For this purpose, the description of all vertices should be summarized in a simple database with two ratios - for vertices (states) and their vertices of transition associated.

### 3 Conclusions

The presented technology of program development allows to build their models in the form of non-deterministic finite state automates according to the description of the program algorithm in the terms of technical task. The program verification process consists of verifying that the model description is correct. This software development technology is used in the educational process at the Cherkasy National Bogdan Khmelnitsky University, and can later be used to develop complex software systems.

### Literature

2. Салахов В.І. Моделювання, верифікація та розробка програм. Вісник національного технічного університету України “КПІ”. Інформатика, управління та обчислювальна техніка. № 61. 2015. С. 174-177.