МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 2012
MATHEMATICS AND EDUCATION IN MATHEMATICS, 2012
Proceedings of the Forty First Spring Conference
of the Union of Bulgarian Mathematicians
Borovetz, April 9–12, 2012

VERIFICATION OF PROCEDURAL PROGRAMS VIA BUILDING THEIR GENERALIZED NETS MODELS*

Magdalina Todorova

In the article an approach for verification of procedural programs via building their corresponding generalized nets models is described. This approach integrates the concept of "design by contract" with approaches for verification of type theorem proofs and models consistency check. For this purpose, functions which compose the program, are verified separately according to their specifications. A generalized net model is built, specifying the relationships between functions in the form of correct sequences of calls. For the main function of the program, a generalized net model is built and it is checked whether it complies with the net model of relations between the functions of the program. Each function of the program, which uses other functions defined in the program, is verified also according to the specification set by the net model of relations between the functions of the program.

1. Introduction. The increasing complexity of software leads to an increase in the amount of errors in it, from which it follows the increase loss caused by these errors. This motivates the efforts of scientists and software engineers to conduct active research for the purpose of development of software environments for verification of the software being developed [1–5].

In the foundations of the approach described in this article is the concept of design by contract [6], implying a description of specific statements (contracts) that have to be in force in certain areas of the program. Contracts can describe: simple assertions, preconditions and postconditions of the functions of the program. They are also called program specifications.

Specifications are most often based on statements, describing the core nature of software. In the proposed approach except specifications of this type, specifications setting the relations between the functions defined in the program are used. The latter are defined by generalized nets (GNs) [7].

The choice of GNs as means for modeling is done according to the following considerations:

– Some specifications according to which we verify functions of the procedural program are network data structures defining all eligible states in which a program variable can

*ACM Classification: D.2.4 Software Engineering: Software/Program Verification – Formal methods, Model checking.

Key words: Generalized Nets, Modeling, Verification, Formal Verification Methods, Procedural Programming.

The paper is supported by Grant 182/2011 from Sofia University Research Fund.

enter, as well as all sequences of correct calls to the functions of the program. GNs are a means by which not only we can easily define network data structures but also additionally we can study their properties.

– Limited means of expression in the generalized nets compared to the languages of formal logic simplifies the verification process and makes the method practicable.

– A methodology is developed for building GNs [7] and [8], with which it can easily be built a GN corresponding to the function of a procedural program.

– It is possible through GNs efficiently to run parallel other GNs. The latter is the basis of the algorithm for checking the consistency of models of the functions that are subject to verification by the net model of relations between the functions of the program.

– GNs allow modeling of real processes [9, 10].

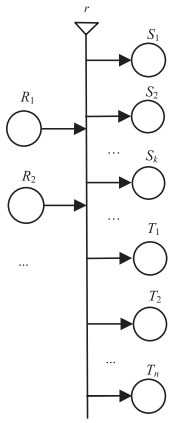
– An environment GN Lite [11, 12] is realized to work with GNs, the capabilities of which are subject to improvement and development [13–16].

2. Approach description. In the description we will consider the language C++, but this does not limit the approach.

a) Step 1

A generalized net model specifying relationships between the functions of the program is built. It defines all possible correct ways to calls to the defined functions. We call this model *a formal net project of the connections between the functions defined in the program* or, *net model of relations between the functions of the program*. The main function of the program and each function that uses other functions of the program should have a behavior, which corresponds to the formal net project of the connections between the functions.

The formal project of the connections between the functions is a generalized net and consists of a set of transitions of the type



where $r =< \{R_1, R_2, \ldots\}, \{S_1, S_2, \ldots, S_k, \ldots, T_1, T_2, \ldots, T_n\}, r' >$.

The predicates P_1, P_2, …, P_k, as well as Q_1, Q_2, …, Q_n, relating to the application of the same function are mutually exclusive. The functions f, …, g are different. The main function of the program is not included in the formal net project of the connections between the functions defined in the program.

$r'=$	S_1	S_2	...	S_k	...	T_1	T_2	...	T_n
R_1	*Function is f. AND Condition* P_1 *holds.*	*Function is f. AND Condition* P_2 *holds.*	...	*Function is f. AND Condition* P_k *holds.*	...	*Function is g. AND Condition* Q_1 *holds.*	*Function is g. AND Condition* Q_2 *holds.*	...	*Function is g. AND Condition* Q_n *holds.*
R_2	*Function is f. AND Condition* P_1 *holds.*	*Function is f. AND Condition* P_2 *holds.*	...	*Function is f. AND Condition* P_k *holds.*	...	*Function is g. AND Condition* Q_1 *holds.*	*Function is g. AND Condition* Q_2 *holds.*	...	*Function is g. AND Condition* Q_n *holds.*
...	...	...	...	...	...	...	...	...	...

Each place of a transition presents a logical state in which a variable of the program can fall. We call it a *logical state of the variable in the place* or, a *logical state of the place.* The logical state is set by a Boolean expression that combines the most preconditions of the program functions. With each variable of the program a value is associated. The tokens in the places of a formal net project of the connections between the functions correspond to the variables defined in the program. Their characteristics are given by pairs of the type *(name_of_variable, position)* where *position* is the name of the place in which the variable is in the generalized net. Besides these two components in the characteristic of the token, by the parameter *name_of_variable* implicitly participates the value of the variable (we will call it also *value the token*) and by the parameter *position* – the logical state of the variable in the place (we will call it a *logical state of the token*).

Transitions assign the functions, which can be executed correctly in prerequisites (preconditions), following from the logical states of the places.

b) Step 2

Each of the functions defined in the program is verified according to the specification, which corresponds to its intended functionality. For this purpose a theorem is built as a Hoare triple [17]. The proof of the theorem can be realized using the technique of the transforming predicates [18] and also with some of the systems for automatic proof of theorems HOL, Isabelle, Coq, etc.

Furthermore, with this step it is proved that the formal net project of the connections between the functions describes correct sequences of calls to the functions of the program. For this purpose, for all places of the formal net project of the connections between the functions, the accuracy of the statements needs to be checked:
where $Body_f$ and $Body_g$ denote the bodies, and pre_f and pre_g – the preconditions of the functions f and g, respectively.

The trueness of these statements, combined with the verification of the functions of the program according to the specification corresponds to its intended functionality ensures that all sequences of calls to functions of the program, corresponding to the execution of transitions in the formal project net of the connections between the functions, are correct.

This and the previous steps are the hardest, but once achieved can be used for verification of all functions that need to be verified according to the formal net project of the connections between the functions of the program.

logical state of place R_l => pre_f ... *logical state of place* R_l => pre_g ($l = 1, 2, \ldots$)	{*logical state of place* R_l && P_i} $Body_f$ {*logical state of place* S_i } (for each $i = 1, 2, \ldots, k$) ... {*logical state of place* R_l && Q_j } $Body_g$ {*logical state of place* T_j } (for each $j = 1, 2, \ldots, n$)

c) Step 3

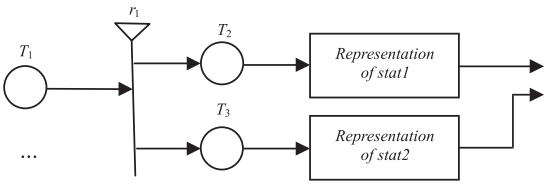
If a function calls another function defined in the program, then it must be verified except for a specification derived from its specificity and for the specification set by the formal net project of the connections between the functions.

Verification of the function on the specification set by the formal net project of the connections between the functions is implemented as a generalized net model of the function is construct and is proved that the formal net project of the connections between functions and the generalized net model of the function are consistent.

For simplicity of description we assume that the function M, which is subject to verification contains operators for input/output, for assignment, conditional (if, if-else), for cycle (while and do-while), and for calls to functions. The GN model of this function we denote by GN_M. Each transition of this net corresponds to the execution of one or more operators of M.

Operators for input/output, for assignment and for calls to functions correspond to a transition with one or more input places and one output place.

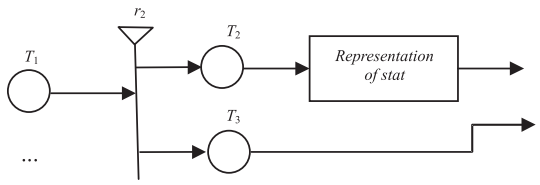
The conditioning operator *if* (*B*) *stat1; else stat2;* corresponds to a net project element of type



where
$r_1 =< \{T_1, \ldots\}, \{T_2, T_3\}, r_1' >$

$r_1' =$	T_2	T_3
T_1	B	$\neg B$
...		

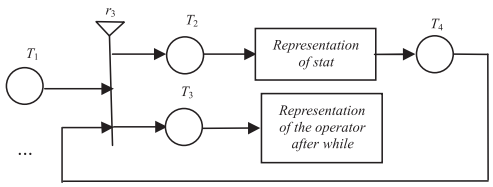
The conditional operator *if (B) stat;* corresponds to a net project element with structure of the type



where
$r_2 =< \{T_1, \ldots\}, \{T_2, T_3\}, r_2' >$

$r_2' =$	T_2	T_3
T_1	B	$\neg B$
...		

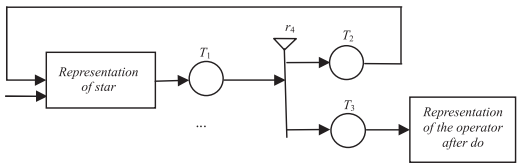
The loop operator *while* (B) *stat;* corresponds to a net project element with structure of the type



where
$r_3 =< \{T_1, \dots\}, \{T_2, T_3\}, r'_3 >$

$r'_3 =$	T_2	T_3
T_1	B	$\neg B$
T_4	B	$\neg B$
$\dots$		

The loop operator *do stat while* (B); corresponds to a net project element with structure of the type



where
$r_4 =< \{T_1, \dots\}, \{T_2, T_3\}, r'_4 >$

$r'_4 =$	T_2	T_3
T_1	B	$\neg B$
$\dots$		

The verification of the consistency of both models – the function M, which is subject to verification (GN_M) and the formal net project of the connections between the functions defined in the program (we denote it by GN_F) is implemented by executing the models in parallel. This article briefly present the idea to its implementation. For simplicity of description we confine to the cases where the actual parameters of calls to functions are variables.

Each execution of the transition in GN_M, containing a call to a declaration (definition) of a variable causes activation of two new tokens – one in the current place of the transition in GN_M, corresponding to the execution of the declaration and another in the current place or in an input place of GN_F. The new tokens in the current places of the generalized nets GN_M and GN_F are obtained as a result of the split of already existing tokens. These tokens have the same names, corresponding to the name of the variable defined. After the execution of the transitions, which contain the execution to the declaration, these tokens take characteristics of the defined above type. If the definition is with initialization, the tokens have values equal to the value of the initializing expression. If the definition is without initialization, the tokens are with undefined values. Each token has a period of activity, beginning with the execution of a transition, executing a definition of a variable and ending after the execution of a transition, realizing the completion of the block (the operator), into which the variable is "visible".

The token is created in a place, which can be empty, but can also contain tokens. All tokens of the net GN_M are gathered in one place and are transferred simultaneously in transition execution. Their number is increasing with the execution of a definition and decreasing when the block in which the variable was defined ends. If the transition corresponds to the execution of a call to a function in the program, with it comes a series of the actual parameters of the function call.

In the case that in a place of the net GN_M there are tokens it is possible:

– *The place is input for a transition, corresponding to an action, which is not a call to a function of the program.*

If the transition can be executed, then it is executed and all tokens in the place are transferred from the input to the corresponding output place of the transition. The corresponding tokens in GN_F do not change their places.

– *The place is input for a transition r, corresponding to executing a call to a function of the program.*

Let the function corresponding to the transition r have a name f and actual parameters $(x, y, \ldots, z)$. In this case, if there are no tokens in the input place with names x, y, $\ldots$, z, then the transition does not run – an error has occurred. Otherwise the transition will be executed if in parallel with it can be run the transition of $GN_F$with an input place, containing tokens with names x, y, $\ldots$, z and with a condition that contains predicate "function is f". The requirement for the parallel execution of the two transitions provides correctness of the call to the function f of M. If the transition r cannot be executed, then the models GN_M and GN_F are not consistent. The latter corresponds to "there is an incorrect call to the function of the program."

3. Conclusion. This article describes a formal method for verification of procedural programs. As a means of building models of the programs and defining the specification, giving the relations between the functions of the program generalized nets are used. The restricted expressibility of the generalized nets in comparison to the languages of formal logics simplifies the process of verification, which makes the approach practically applicable. The research made, as well as the highly activated work on improving the implementation of the programming environment GN Lite motivate continued research.

REFERENCES

[1] D. Hovemeyer, W. Pugh. FindBugs™ Manual, University of Maryland, 2008, `http://findbugs.sourceforge.net/manual/index.html` (last visited on 20.09.2011).

[2] `http://ru.wikipedia.org/wiki/SourceAnalyzer#.D0.A1.D1.81.D1.8B.D0.BB.D0.BA.D0.B8` (last visited on 21.10.2011)

[3] `http://en.wikipedia.org/wiki/Software_testing#Testing_methods` (last visited on 21.10.2011)

[4] D. Petrova-Antonova, I. Krasteva, S. Ilieva. Approaches facilitating WS-BPEL testing. 17th Conference on European Systems & Software Process Improvement and Innovation (EuroSPI2 2010), Grenoble Institute of Technology, France, September 2010, 5.1–5.17.

[5] S. Ilieva, V. Pavlov, I. Manova. A composable framework for test automation of service-based applications. 7th International Conference on the Quality of Information and Communications Technology (QUATIC), Porto, Portugal, 2010, 286–291.

[6] B. Meyer. Applying "design by contract". *Computer*, **25** (1992), No 10, 40–51.

[7] K. Atanassov. Generalized Nets. World Scientific, Singapore, 1991.

[8] K. Atanassov. On Generalized Nets Theory. Prof. Marin Drinov Academic Publishing House, Sofia, 2007.

[9] E. Sotirova, D. Orozova. Generalized net model of the phases of the data mining process. *Developments in Fuzzy Sets, Intuitionistic Fuzzy Sets, Generalized Nets and Related Topics*, Volume II: Applications, IBS PAN – SRI PAS, Warsaw, 2010, 247–260.

[10] K. Atanassov, D. Orozova, E. Sotirova. Generalized net model of an intuitionistic fuzzy expert system with frame-type data bases and different forms of hypotheses estimations, *Annual of BFU*, **XXI** (2010), 257–262.
[11] T. Trifonov, K. Georgiev. GNTicker – A software tool for efficient interpretation of generalized net models. *Issues in Intuitionistic Fuzzy Sets and Generalized Nets*, **3** (2005), 71–78.
[12] T. Trifonov, K. Georgiev, K. Atanassov. Software for modelling with generalized nets. *Issues in Intuitionistic Fuzzy Sets and Generalized Nets*, **6** (2008), 36–42.
[13] K. Atanassov, D. Dimitrov, V. Atanassova. Algorithms for tokens transfer in the different types of intuitionistic fuzzy generalized nets. *Cybernetics and Information Technologies*, **10** (2010), No 4, 22–35.
[14] D. Dimitrov. A graphical environment for modeling and simulation with generalized nets. *Annual of "Informatics", Section Union of Scientists in Bulgaria*, **3** (2010), 51–66.
[15] D. Dimitrov. Software products implementing generalized nets. *Annual of "Informatics", Section Union of Scientists in Bulgaria*, **3** (2010), 37–50.
[16] D. Dimitrov. Optimized algorithm for token transfer in generalized nets. Proc. of 9th IWIFSGN 2010, 8 October 2010, Warsaw, Poland (in press).
[17] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, **1** (1972) No 4, 271–281.
[18] D. Gries. The Science of Programming. Springer-Verlag, Berlin and New York, 1981.

Magdalina Todorova
Faculty of Mathematics and Informatics
St. Kl. Ohridski University of Sofia
5, James Bourchier Blvd
1164 Sofia, Bulgaria
e-mail: magda@fmi.uni-sofia.bg

ВЕРИФИКАЦИЯ НА ПРОЦЕДУРНИ ПРОГРАМИ ЧРЕЗ ИЗГРАЖДАНЕ НА ТЕХНИ ОБОБЩЕНИ МРЕЖОВИ МОДЕЛИ

Магдалина Василева Тодорова

В статията е описан подход за верификация на процедурни програми чрез изграждане на техни модели, дефинирани чрез обобщени мрежи. Подходът интегрира концепцията "design by contract" с подходи за верификация от тип доказателство на теореми и проверка на съгласуваност на модели. За целта разделно се верифицират функциите, които изграждат програмата относно спецификации според предназначението им. Изгражда се обобщен мрежов модел, специфициращ връзките между функциите във вид на коректни редици от извиквания. За главната функция на програмата се построява обобщен мрежов модел и се проверява дали той съответства на мрежовия модел на връзките между функциите на програмата. Всяка от функциите на програмата, която използва други функции се верифицира и относно спецификацията, зададена чрез мрежовия модел на връзките между функциите на програмата.