Доклади на Българската академия на науките
Comptes rendus de l'Académie bulgare des Sciences

Tome 66, No 7, 2013

MATHEMATIQUES
Informatique

USING GENERALIZED NETS FOR PROGRAMME VERIFICATION

Magdalina Todorova

(*Submitted by Corresponding Member K. Atanassov on March 25, 2013*)

Abstract

The article presents elements of a solution to an open problem related to applying the generalized nets apparatus to verification of procedural and object-oriented programmes. An approach to the verification of these types of programmes is described. In the case of object-oriented programmes (OOP), the description concerns the situation in which the classes are not connected into inherent hierarchies. In addition, the question of generalized net models verification is presented, when they give the specifications according to which the relations between the procedural programme functions and the member functions of the OOP classes are verified.

Key words: generalized net, modelling, verification, programming
2000 Mathematics Subject Classification: 68Q90

The paper is supported by the Bulgarian National Science Fund under the Project DFNI-I01/12 "Contemporary programming languages, environments and technologies and their application in building up software developers".

1. Introduction. Some open problems, related to the development of the theory and practice of the generalized nets (GNs), are listed in Chapter 11 of [1]. One of these problems is the problem of GNs verification and GNs application in programme verification. It was formulated as follows:

• *To construct algorithms which transform every algorithmic language programme to a GN, with which particular verification of the programme is to be made* ([1], p. 358),

and

• *To research the possibility of GN's verification* ([1], p. 357).

This article describes elements of a solution of the defined above problem in the case of procedural and object-oriented programmes developed in C++. The solution is based on formulation and mathematical justification of an approach to verifying such programmes. The solution is incomplete due to the wide variety of programming languages and styles, as well as due to the broader understanding of *verification* as a term. The standard IEEE 1012 defines verification as a technique which checks whether the artefacts (such as a technical task, subject area model, architecture description, programme code, user manual, etc.), created in the process of programme development, correspond to other artefacts identified as input data.

A number of verification methods could be found in the literature, which can be divided into the following categories: software expertise, static analysis, formal methods, dynamic methods and synthetic methods. The formal methods are most reliable among them. They are used to verify characteristics, which can be formally presented within the framework of some mathematical models, as well as of these artefacts, for which respective formal methods can be built.

The suggested verification approach can be defined as a formal verification method of the type checking for correspondence as both models were built during their realization – of the specification and of realization – are executable (they are based on GNs [10]).

The choice of GNs for the suggested approach is motivated by the following considerations:

- The specifications, according to which the OOP functions' verification is performed, are presented by net structures.
- GNs have a high level applicability in the field of software modelling [6].
- GNs are a subject to rigorous research from theoretical and practical point of view [3–5].
- Automated tools for GNs implementation are developed and are currently being improved.
- GNs are means of modelling parallel processes, which allows effective execution of GNs in parallel [1, 2].

- A methodology of building GNs is developed [1, 9], which facilitates the building of GNs to procedural and object-oriented programmes.

Apart from GNs, the approach under consideration also uses elements of Hoare's logic, the method of transforming predicates and the model "design by contract".

2. Verification approach – a solution to the problem. In the case of OOP, the approach consists of separating the OOP class verification from the verification of the functions which use them. The verification of each class is performed according to Definition 1 [7].

Definition 1. *Class C is correct in respect to its specification if*:

a) *The Hoare's triple*

$$\{\text{Default}_C \wedge \text{pre}_p(x_p)\}\text{Body}_P\{\text{post}_p(x_p) \wedge \text{Inv}\}$$

holds for each class constructor P and for each admissible set of arguments x_p.

b) *The Hoare's triple*

$$\{\text{pre}_r(x_r) \wedge \text{Inv}\}\text{Body}_r\{\text{post}_r(x_r) \wedge \text{Inv}\}$$

holds for each member function r, different from the class constructor, with a set of admissible arguments x_r,

where, Inv *is the invariant of C, and r is an arbitrary member function of the class.* Body_r *denotes the body of r,* $\text{pre}_r(x_r)$ *and* $\text{post}_r(x_r)$ *are the precondition and postcondition of r, with admissible arguments x_r.* Default_C *denotes a statement which expresses implicit relations among data members of the class C.*

In order to verify a function, which uses OOP classes, the following actions are performed:

i. A specification (invariant of the class, precondition and postcondition of each of its member functions) is given for each class. By applying definition 1, verification of each OOP class is performed in accordance to a specification defined for it.

ii. A GN model, which defines the relations between the class methods in a form of correct sequences of calls, is developed for each class. These models define the specification, according to which the OOP function verification is performed. They are called formal GN models (or formal GN projects) of the classes.

iii. The OOP function, which is to be verified by the specification described in ii), is presented by a GN.

iv. The GNs of the function and specification, defined in ii), are executed in parallel in order to identify whether the function model corresponds to the specification.

A non-formal description of the approach is given in [10]. For the sake of clarity, the following description of the formulated stages is given in the case when the OOP contains a single class.

2.1. Building a formal net model of a class. The formal net model of a class is a GN for which the conditions of Definition 2 (see below) hold. Each transition position of the formal GN model presents a logical state in which a class object can be. It is called *a logical state of the object in the place* or *a logical state of the place* in short. It is defined by a Boolean expression. The tokens in the places of this GN correspond to the class objects. Each class object is connected to a *data* set, defined by member data of the class. The characteristics of the token are given by couples of the type (*object, place*). Here *object* is an identifier which gives the class object name, and *place* is the name of the place in the GN, in which the token (object) is. Apart from these two components, the set of object data is presented implicitly through the parameter *object* in the token characteristics, and the logical state of the object in the place: through the parameter *place.*

The conditions of the net transitions have the following form:

The member function is f. $\wedge$ *Condition P holds.*

They give the member functions which can be executed, and the conditions under which this can be done.

Definition 2. *The generalized net, with the characteristics described above, is called a formal GN-model of the class C, and is denoted by* GN_C*, if the following conditions hold for it*:

a) *The implication*

$$\text{(1)} \qquad \textit{logical state of the place } S \Rightarrow \mathrm{pre}_f$$

holds for each place S of GN_C*, different from the input place. Here f is an arbitrary member function of the class C, which is a part of the conditions of the transition for which S is an input place. The predicate* pre_f *is the precondition of f.*

The implication

$$\text{(1}') \qquad \mathrm{Default}_C \Rightarrow \mathrm{pre}_{Cs}$$

holds for the input place of GN_C*, for each constructor* C_s*. The predicate* pre_{Cs} *is the precondition of Cs.*

b) *The implication*

(2) $$\textit{logical state of the place } S \Rightarrow \mathrm{Inv}$$

holds for each place S *of* GN_C*, different from the input place.*

c) *For each pair at input and output positions* (R, S) *of a transition different from the one realizing the constructor (constructors) of the class* C*, with a transition condition*

$$\textit{Member-function is } f \wedge \text{Condition } P \textit{ holds},$$

the Hoare's triple holds:

(3) $$\begin{gathered}\{\textit{logical state of the place } R \wedge P\}\\ \mathrm{Body}_f\\ \{\textit{logical state of the place } S\}\end{gathered}$$

For each pair at input and output positions (R, B_s) *of the transition, realizing the execution of the constructor (constructors) of the class with a transition condition.*

Member-function is the constructor $C_s \wedge$ Condition Q_s *holds,*
the Hoare's triple holds:

(3′) $$\begin{gathered}\{\mathrm{Default}_C \wedge Q_s\}\\ \mathrm{Body}_{Cs}\\ \{\textit{logical state of the place } B_s\}.\end{gathered}$$

Conditions (1) and (1′) provide the holding, in the input positions of GN_C transitions, of the preconditions of all member functions which the respective transition executes. Condition (2) ensures the trueness of the class invariant for the current values of the token data in each place of the net, different from the input one. Conditions (3) and (3′) ensure keeping the trueness of the predicate *logical state of a place* for the current values of the token data.

We choose the specification of class C for a specification of GN_C.

The definition of correctness of a formal GN project of a class according to its specification is similar to definition 1, and is formulated in [8].

Theorem. *If class* C *is correct in respect to its specification, the formal GN-project* GN_C *of the class* C *is also correct according to its respective specification.*

By means of the formulated above definitions and theorem, a solution is found to the GN verification task, in the case when the GN gives a specification according to which OOP function verification is performed.

A GN model, which defines a relation between the class methods, is built at this stage. A proof that this GN model is a formal net model of a class is performed.

2.2. Defining a GN-model for each function which has to be verified. GN models of all OOP functions which have to be verified are built at this stage. In order to achieve this, GN models of the basic operators of the language are used. This is completed by the developer who uses the class. These models' definition is easy and can be done automatically.

2.3. Consistency check of the model of the function under verification and the formal project of the class. Having built the GN model of the function which is to be verified according to the formal GN model of the class, a consistency check is performed to show whether the two models correspond to each other. If inconsistency is identified, an error in the function under verification is issued.

The approach in the case of OOP is easy to adapt for the case of procedural programmes.

The approach presented as a solution of the open problem can be applied in cases of complete verification of procedural and object-oriented programmes, as well as for interface verification: verification according to a specification which describes the relations between the procedural programme functions and the respective relations among the member functions of a class or relations among classes which are included in the OOP.

3. Realization of the approach. In order to realize the approach, a programming system is developed, whose architecture is presented in Fig. 1. The component *Function* gives the OOP function, which is to be verified. *Preprocessor* stands for a translator which translates the function to be verified: from C++ language into the corresponding GN. The module *Model Checker* takes the GN, which is a result of the preprocessor work and the formal GN project, which

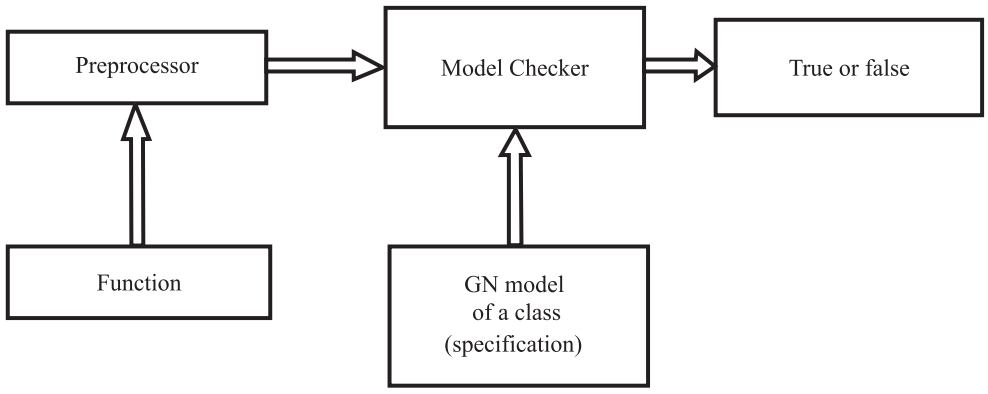

Fig. 1. Architecture of a programme system which realizes the verification approach

presents the specification and determines whether the respective nets correspond to each other. In case the check result is negative, the model checker provides the reason for the error and the place of its occurrence. *Model Checker* module is realized by a GN, which executes in parallel the input GNs.

The translator *Preprocessor* can also be viewed as part of the solution of the open problem, as formulated in [1], in the case of C++ programmes. The realization of such translators for other procedural or object-oriented languages will not pose significant problems.

4. Conclusion. Solving the problem as formulated in [1] is of crucial importance for building up a quality programme code. The approach described in this article is in an initial stage of experimentation. Its application in teaching software specialists of the Faculty of Mathematics and Informatics, Sofia University is being planned. The next step considered is researching the problem in the cases when programmes contain a large number of classes, connected in inheriting hierarchies, as well as in cases of other procedural and object-oriented programming languages, and also in cases of other programming styles.

REFERENCES

[1] Atanassov K. Generalized nets. World Scientific, Singapore, New Jersey, London, 1991.

[2] Atanassov K. On generalized nets theory. Prof. Marin Drinov Academic Publishing House, ISBN 978-954-322-237-7, Sofia, 2007.

[3] Atanassova V. The Minimal Solution of a Problem in Generalized Nets, 6-th International Conference IEEE Intelligence Systems'2012, 6–8 Sept., Sofia, Bulgaria, 2012, **2**, 159–163.

[4] Atanassova V. Compt. rend. Acad. bulg. Sci., **65**, 2012, No 11, 1489–1498.

[5] Gocheva P., V. Gochev. Proceedings of the 13th International Workshop on Generalized Nets, 29 October 2012, London, 69–76.

[6] Tcheshmedjiev P. Annual of "Informatics" Section, Union of Scientists in Bulgaria, **5**, 2012, 111–119.

[7] Meyer B. Object-Oriented Software Construction, Second ed., ISE Inc. Santa Barbara, California, 1997.

[8] Todorova M. Correctness of a Formal Generalized Net Project of a Class of an Object-Oriented Program, Proc. of 12th International Workshop on Generalized Nets, Burgas, Bulgaria, 17 June 2012, 72–77, ISSN: 1313-6860.

[9] Todorova M. Methodological Aspects of an Approach for Verification of Object-Oriented Programs", 6-th International Conference IEEE Intelligence Systems'2012, 6–8 Sept., Sofia, Bulgaria, 2012, 153–158.

[10] Todorova M. Annual of "Informatics", Section of the Union of Scientists in Bulgaria, **4**, 1–28 (in Bulgarian).

Faculty of Mathematics and Informatics
St Kliment Ohridski University of Sofia
5, J. Bourchier Blvd
1164 Sofia, Bulgaria
e-mail: todorova_magda@hotmail.com