INFORMATICA, 1993, Vol.4, No.3–4, 384–398

COMPUTER-AIDED TEST PROGRAM DESIGN SYSTEM (CATPDS) IN ATLAS

Vytautas ŠTUIKYS and Eugenij TOLDIN

University of Technology
3028 Kaunas, Studentų St. 50, Lithuania

Abstract. The configuration and essential features of the Computer-Aided Test Program Design System (CATPDS) which generates test programs in an adapted ATLAS subset for analogue units under test are discussed. The requirements for that class of systems are formulated and how to meet these requirements is proposed. The formal model to describe the process of an interactive test program generation and incremental translation is presented.

Key words: integrated Computer-Aided System, interactive program generation, test program, analogue unit under test (UUT), the ATLAS language.

1. Introduction. In most automated test systems the development of test programs for units under test (UUT) is required. These test programs are typically developed by engineers. In order to improve the test program development process, the design tools are needed. The Computer-Aided Test Program Design System (CATPDS) is an example of that category of tools. CATPDS is a subsystem of the more common system, called CATS (Computer-Aided Testing System). The main features of CATPDS are as follows:

a) test programs are generated by the system in an adapted ATLAS (IEEE Standard ATLAS Test Language, 1981) subset and prepared for execution on CATS;

b) CATPDS is oriented to use for functional testing of analogue UUT;

c) the key program of CATPDS is the Test Program Generator

(TPG), which provides facilities for syntax error-free test program design in an interactive mode;

d) the system includes other closely integrated facilities, such as translators, editors etc.;

e) the user of CATPDS may choose three levels of a test program generation process (statement, frame and frame's library) ;

f) the user may receive a test program written in original ATLAS, i.e., in English or in Russian representation of the same ATLAS program;

g) CATPDS is independent upon physical test instruments of the automated test system and has an emulation facility for test program debugging;

h) the test program design process is combined with on-line help facilities and specific messages to prompt or warn the operator are used;

i) the test program generation process is based on menu-driven application;

k) CATPDS is implemented in the environment of MS DOS PC.

The variety of such kind systems is developed now. Among them the paper (Michael, 1979) in which the computer guided generation of test programs is proposed must be mentioned. Some ideas how to build an interactive tool based on an ATLAS-like language are described in (Ponomariov, Frumkin, Gusinskij, 1984). CATPDS is compared with the SMART system (ARINC Specification 608-1, 1989). The lack of CATPDS lies in configurating capabilities but our system has several possibilities for test program generation, i.e., the statement, frame and frame's library levels.

2. Requirements for test programming tools (TPT). The global requirement which should be assumed by the designer of TPT is as follows: a developed tool has to support the program design and execution processes more efficiently then it is done in the conventional systems.

On the basis of developed and investigated several tools for test systems (Ginkas, Štuikys, 1983; Štuikys, Toldin, 1988) (CATPDS

is the last one) we can reformulate this common requirement in more detail as presented below.

1. A tool is to be built as a system in which the following facilities are closely integrated: an interactive test program generator, translator, editor, linker, emulator and executor.

2. Each statement of a source test program produced by TPG of the tool must be syntax error-free. Some semantic errors also could be detected by TPG.

3. When an interactive facility for the source test program generation is incorporated in a tool the incremental translation scheme at the first translation stage is needed to use.

4. Some others translation schemes (batch, independent) are applicable in the same tool.

5. To assure a more efficiency some source test program designing levels are needed when TPG is used. We propose the following levels of the test program written in ATLAS: a statement, frame and frame's library.

6. To achieve greater flexibility various editors, such as textual, syntax-oriented, screen are to be integrated into the same system.

7. Flexible user interfaces must be designed through the entire test program development stages, including debugging and execution.

8. A tool is to be independent upon changes and extentions in a source high level language.

9. A tool is to be independent upon the physical instruments of the test system.

10. It is a very useful property when some modules of a tool can be easily transported from one computer to another (the property of mobility).

11. Due to the wide spread of ATLAS as a standard test language over the world, it is useful to have a facility which could enable to change an original ATLAS program to other natural language representation of the same test program.

12. The formal models for describing processes which are to be implemented in a tool on the development stage are desirable.

Some requirements shall be needed in a near future. These may be as follows.

13. To enhance the intelligence level of a tool.

14. The previous efforts to carry in a standard for the usage of some modules (translators, for example) could be realised more fully.

A majority of requirements mentioned above are evident. The others, although are comprehensible, need a more detail discussion.

In the next section we shall discuss only those requirements which are related to initial stages of the test program design, i.e., generation and translation.

3. How can designer meet proposed requirements. The need to build an integrated test programming tool lies in the inhierent relation of some processes which are to be incorporated into the tool and a necessity to achieve the functional capabilities of the system. The *internal* integration could be entered by such means as the use of

- a common database,
- common control procedures,
- an internal feedback between different modules.

How to enter the integration we shall illustrate with an example of a database as follows. The main part of the database is the information about the source language syntax. In the case when TPG is incorporated in TPT this information, for example, could be represented in the form of menu tables as shown in Fig. 1. These menu tables can be successfully used also at the program translation stage.

The essential feature of TPG is that a test program produced by the tool is syntax error-free. This requirement may be achieved in the following way. When an error occurs via the statement generation session, it must be immediately detected and deleted. To ensure this some checking facilities are needed. These checking facilities can be interpreted as a part of an incremental translator.

The incremental translation is required when a statement is produced not manually but by the Generator use due to the fol-

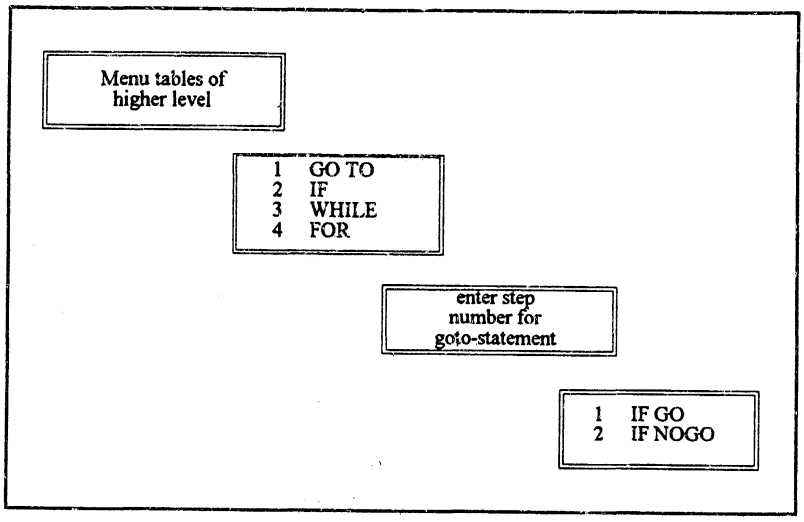

Fig. 1. Menu tables for goto-statement.

lowing reasons:

- to detect errors immediately,
- to create an intermediate representation of a test program after it is translated,
- to make proper further processing of a translated program.

Methods how to check the correctness of produced items of a test program via a generation session are well-known. In our case we would like to emphasize the most significant property that not the entire information produced by the Generator must be checked when an incremental translation scheme is used. This checking is being done only to the part of items which is created manually via generation session (for example, pin names, numbers, variables, etc.). We suppose the menu tables of the database are constructed correctly and appropriate separators can be formed by the Generator automatically.

So, the proposed approach is less complicated then the conventional ones.

The next note must be made about the program representation form after it being translated. This form we named the *Intermedi-*

ate Code (IC), a role of which is very wide as it is stated below.

** A source test program may be saved in the IC representation and the textual form of that program may be reproduced from IC only when it is needed. Such reproduction process we called a *retranslation.*

** When ATLAS is used as a test programming language the common form of IC may be proposed in order to provide the standardization facilities and enhance the test program's mobility.

** We propose further processing of IC in such a manner: to *convert* IC into the C language representation and from this stage use the conventional environment. Due to the wide spread of the C language compilers the proposed approach assures the test program transportability in the scale of the mini and personal computers environment.

4. Formal description of interactive test program generation. As test programming tools continue to increase in complexity, so do the interface requirements that support the corresponding user interaction. Due to this reason it is very useful for the designer to have a precise description and transparent understanding of these processes which are to be incorporated into the tool. To promote such understanding in this section a model which describes an interactive test program generation and incremental translation is presented.

Some approaches to describe program generation were proposed earlier. Among them: a syntax-oriented edition in (Medina–Mora, 1981), a procedure to produce a program from a formal syntax tree representation (Swartz, Delisle,Begwani, *et.al.*,1984). The Program Synthesizer (Teitelbaum, Reps, 1981) uses the templates as a model to produce a program written in PL/CS. The test program generation for logic units on the basis of a s successive selection items from menu tables was discussed in (Gross, Gerg, 1983). The menu-driven systems were classified and models to describe them formally were proposed in (Arthur, 1987).

Our model is based on an automata theory (Lewis, Rozencrants, 1979). According to that model a test program is generated

gradually, in statement-by-statement manner. Moreover, a statement is produced also in step-by-step manner as described in the following.

When a system, i.e., a generator, produces some item of the statement, it is said that a system is in an adequate state. Some kinds of states are distinguished. The *initial*, *current* and *terminal* states are determined. It is assumed that each state is associated with an adequate menu table.

When the system is in current state the current menu table is displayed and the appropriate response is made by the user. Each response causes the *corresponding* operations in the system and the system moves to the next state.

Menu tables and states for the DECLARE statement of the ATLAS subset are shown in Fig. 2.

A move to the next state depends upon the response and the current state. In such model the legal and invalid moves can be easily determined. Let

$$S = \{s_0, s_1, \ldots, s_c, \ldots, s_t\}$$

be the set that represents states of the system.

Let

$$M = \{m_0, m_1, \ldots, m_c, \ldots, m_t\}$$

denote the set of menu tables and

$$A = \{a_0, a_1, \ldots, a_c, \ldots, a_t\}$$

denote the operations which are to be performed by the system. User responses are represented by the set

$$R = \{r_0, r_c, \ldots, r_n\}.$$

Then the transition function T that maps elements of $S \times R$ into elements S, i.e., T: $S \rightarrow S \times R$ can be defined.

The user moves from the current state s_c (initially $s_s \equiv s_0$) to the state s_{c+1} by issuing response r_c. A move is defined by the

Menu table in state S_0

response	menu item
r_0	DECLARE

Menu table in state S_1

response	menu item
r_1	INTEGER
r_2	DECIMAL
r_3	DIGITAL
r_4	CONN
r_5	MSGCHAR

Menu table in state S_2

response	menu item
r_1	STORE
r_2	LIST

Menu table in state S_3

response	menu item
r_6	enter variable name
r_8	exit

Menu table in state S_4

response	menu item
r_7	enter array name
r_8	exit

Fig. 2. Menu tables for DECLARE statement.

relation $T(s_c, r_c) = s_{c+1}$. If a response is invalid, the transition function T maps the current state into itself and an error message is displayed.

So, a behaviour of the generating system can be modelled by the following automata:

$$W = (s_0, S, R, T, s_t),$$

where the current state s_c is in one-to-one correspondence with the pair (m_c, a_c); $s_0, s_c, s_t \in S$; s_t is the terminal state.

This model for the DECLARE statement is detailed in Fig. 3. There the transition function is represented by the transition matrix. Spaces in the matrix imply invalid moves. For example, if

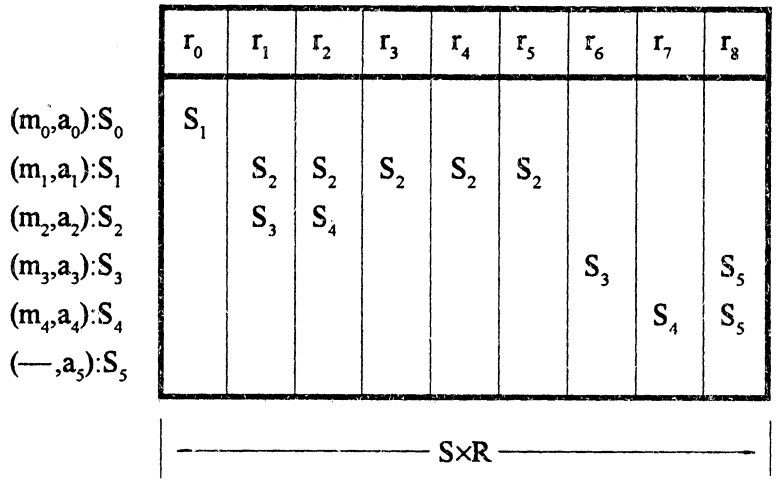

Fig. 3. Model of DECLARE statement generation process.

the response r_c is selected when the system is in the state s_c, it is assumed as an illegal response. The system saves the state s_c and an appropriate message is displayed.

Some notes must be made about responses and operations. Here is a difference between the response r_6 or r_7 (see Fig. 2) and any other response from the list $(r_1, \ldots, r_5)$. It lies in the computational operations which are caused by these responses. The operation a_1, for example, caused by the response r_1 implies retrieving and modifying the source and intermediate code files. As for operations a_3 or a_4 which are caused by responses r_6 or r_7, more sophisticated processing is required. This processing being also the integral part of an incremental translation includes more deep checking of data which were entered by the user response.

It must be outlined that the last checking can be described by similar models which are well-known in compiling theory.

So, the syntax error-free statement generation and translation processes may be described as a permissible sequence of states with operations prescribed in advance, when the generation is initiated in the initial state, then transitions and corresponding operations are performed while the terminal state is achieved.

5. Architecture of CATPDS. The CATPDS architecture is shown in Fig. 4. The system consists of the following parts: database, control module, test program generating subsystem, pin connection table editor, convertor, emulating and debugging module, independent translator, executive code and test instrument libraries. The main files are also denoted. Among them the intermediate code, test program in the C language representation and an executable code could be mentioned.

Test program can be designed by the Generation Subsystem or developed independently.

5.1. Database. The database is specially developed for that tool. The essential part of the database is menu tables. The scale of that part of the database depends on the syntax of the adapted ATLAS subset. In our case the ATLAS subset is oriented to use in analogue circuits testing, so about forty statements were implemented in database. This status may be easily changed due to incorporated refining and adding procedures.

Files created by the system may be also assumed as a part of the database.

5.2. Control module. The control module assures the interaction between the entire modules of the system. The main control functions, such as the maintenance of system regimes, database support, the user interface assurance, etc. are integrated into that module. More specific control functions are incorporated in separate modules.

5.3. Test program generation subsystem. As stated previously, the main function of that module is to provide facilities for a test program design in such a manner that the produced program would be syntax error-free (see Fig. 5). To assure this feature the generating process is implemented on the base of the model which was described in Section 4.

The advantage of our system lies not only in the property mentioned above but also in an integration of such functions as the interactive generation, incremental translation and edition with on-

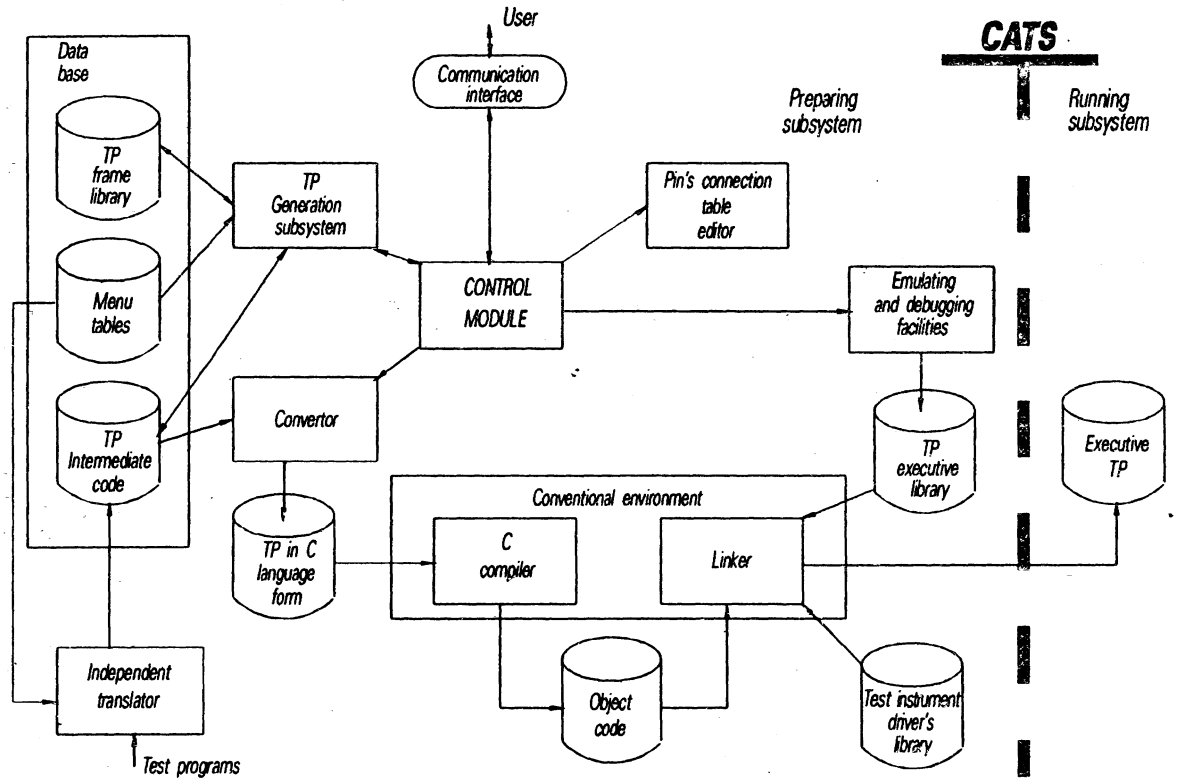

Fig. 4. Configuration of Test program preparation system.

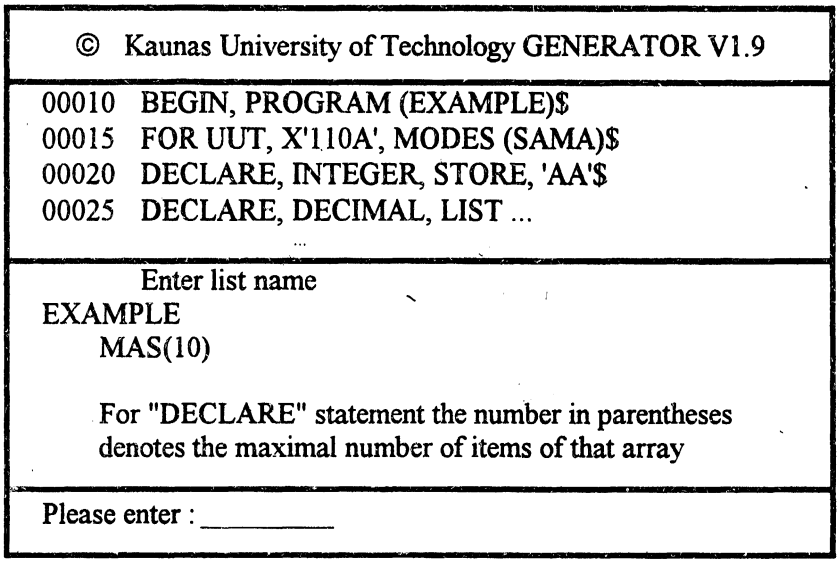

Fig. 5. Interface of test program generation subsystem.

line help facilities into one system. The interactive test program generating process requires some edition facilities, such as killing the last produced statement, etc. The editor which is implemented that system performs also the test program reproducing from its ame representation (see Fig. 6).

```
.....BEGIN, ATLAS PROGRAM.....$
.....FOR UUT, X.....MODES.....$
.....DECLARE, INTEGER, STORE.....$
.....DEFINE, PROCEDURE.....RESULT (.....)$
.....DECLARE, DECIMAL, STORE.....$
.....CALCULATE.....=.....$
.....END.....$
.....TERMINATE, ATLAS PROGRAM.....$
```

Fig. 6. Test program in frame representation.

At last, the subsystem makes further test program processing via the system more easily and properly. As a result the intermediate code file is produced. The possibilities how to achieve these properties were discussed more briefly in Section 3.

5.4. Convertor. The program which produces the test program C language representation form from the intermediate code we called a convertor. The need of such program arises due to the requirement to achieve the mobility of test programs via the use of the conventional C language environment.

5.5. Emulating and debugging facilities. The emulating facilities provide a capability for executing the test program in an off-line mode , i.e., without the running subsystem. Some debugging facilities are implemented into Test Program Executive Library (see Fig. 4) .The others may be incorporated by the user at the test program development stage by means of the ATLAS subset and controlled via this module.

6. Test program design by CATPDS. The test program design process is initiated by opening the source program file and filing up the pin connection table. The latter indicates on which pins testing procedures must be performed and measuring results be received. The pin table filling is initiating only, then on the test program generation stage it may be changed or added by the pin's connection table editor.

In practice, to design a test program the user can choose some possibilities. The first one is that when a program is created in statement-by-statement manner. This situation is named as the statement generation mode. The second possibility is defined as follows. Initially the ATLAS test program frame is created. Next the frame is converted into an applicable test program form by editing means which are incorporated in the TPG subsystem. This is called a design mode on the frame level.

Finally, the third possibility is performed by using the ATLAS test program frame library. The library must be developed by the user in advance. In the following the items of the library can be read

in an appropriate sequence and be connected to form the common file.

The next action is to reduce the frame to the applicable representation. The last possibility is called the test program generation in the frame library level.

When the source test program is created the next processing stage must be selected by the user and the control module enables it to move via the system.

7. Conclusions. The essensial feature of our system is a wide possibility to generate the source test program written in an ATLAS subset. The syntax error-free test program may be created in an interactive mode by the Generator. The Generator itself ensures three levels of the test program generation process: the statement, frame and frame's library. The source program which was prepared ealier by the use of CATPDS with successive modifications made manually or produced in an independently way also may enter the system.

To achieve test program mobility and ensure more efficiency in test program processing, the multi-level translation scheme is implemented, i.e. the generation process is combined with the incremental translation at the first stage; at the next the conversion of an intermediate code which is produced by the Generator to the C language is made and then the conventional compilation is used.

The tool is built as an integrated system where the advanced interfaces are used practically at each program processing stage.

REFERENCES

Arthur, J.D. (1987). Toward a Formal Specification of Menu-Based Systems. *The Journal of Systems and Software*, **7**, 73–82.

Ginkas, and M.L., V.A. Štuikys (1983). Automation of hybrid circuits testing. *Measuring Engineering*, **5**, 6–7 (in Russian).

Gross, O.B., and J.S. Gerg (1983). Automatic ATLAS program generator (AAGP) for the advanced electronic warfare test set. *AUTOTESTCON*, **4**, 286–291.

IEEE Standard ATLAS Test Language. (1981). IEEE std. 416.

Lewis, F., Rozencrants and D. Stearns (1979). *Compiler's Design Theory.* Mir, Moscow. 653pp. (in Russian).

Medina-Mora, R. (1981). An incremental programming environment. *IEEE Trans. of Software Engineering,* **SE-7**(6), 472–482.

Michael, U. (1979). Computer guided generation of test programs for analogues products. *Automatic Testing'79,* Paris. 125–146.

Ponomariov, N.N., I.S. Frumkin, I.S. Gusinskij and *et al.* (1984). *The Design External Tools for Automatic Testing of Radioelectronic Equipments.* Radio and Communication, Moscow. 295pp. (in Russian).

Schwartz, M.D., N.M. Delisle and V.S. Begwani (1984). Incremental compilation in magpie. *SIGPLAN Notices,* **19**(6), 122–131.

Standard modular AVIONICS repair and test system SMART, (1989). ARINC Specification 608-1, September 1.

Štuikys, V.A., and E.J. Toldin (1988). Interactive generator-tool for test program design in high level test language. *Digital Methods and Design Tools and Testing of Electronics Circuits,* **4**, Tallinn. 157–160.

Teitelbaum, T., and T. Reps (1981). The Cornell program synthesizer: A syntax-directed programming environment. *ACM,* **24**(9), 563–573.

Received April 1993

V. Štuikys received the M.S. degree in electrical and computer engineering from Kaunas Politechnical Institute in 1963, and Candidate of Sciences degree from Kaunas Politechnical Institute, Lithuania, 1970. He is currently the associate professor in the Computer Department at Kaunas University of Technology, Lithuania. His current research interests include interactive program generation for problem-oriented systems, expert systems, Computer-Aided Design and manufacturing.

E. Toldin received M.S. degree in Computer engineering from Kaunas Politechnical Institute in 1980, and Candidate of Science from Kaunas Politechnical Institute, Lithuania, in 1987. He is currently the associate professor in the Computer Department at Kaunas University of Technology, Lithuania. His current research interests include analogue circuit testing, test program generation, data base and programming.