Formal Validation of DNA Database Using Theorem Proving Technique

Julaily Aida Jusoh, Md Yazid Mohd Saman, and Mustafa Man
Jabatan Sains Komputer, Fakulti Sains Dan Teknologi,
Universiti Malaysia Terengganu, Mengabang Telipot,
21030 Kuala Terengganu, Terengganu Darul Iman, Malaysia.

j_lily04@yahoo.com, yazid@umt.edu.my, mustafaman@umt.edu.my

***Abstract*- This paper discusses the formal validation of DNA database system. A DNA database is a large, organized body of persistent data, usually associated with computerized software designed to update, query, and retrieve components of the data stored within the system. One of the common difficulties faced by the developer is in designing a robust database system. Even so, in order to solve this matter, developers have to focus their efforts on the formal specifications. This is supposed to reduce the overall development time. Formal specifications can be use to provide an unambiguous and precise supplement to natural language descriptions. Besides, it can be rigorously validate, and verified leading to the early detection of specification errors. Consequently, to validate this problem formally, we specify the DNA database system using Z language and prove by using Z/EVES theorem prover tool. By using this kind of tools, it may helps to reduce time, energy and mistake compared to manual theorem proving which can be error task and tedious.**

***Keywords*- Software Engineering, Formal Specification, Z Specification Language, DNA Database System**

I. INTRODUCTION

According to [1], software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into utilize. In engineering discipline, they apply theories, methods and tools where these are appropriate. Nevertheless, they use them selectively and always try to discover the solutions of the problems even when there are no applicable theories and methods. More to the point, software engineering is not just apprehensive with the technical processes of software development but also with activities such as software project management and with the development of tools, methods and theories to support software production. The software process includes all of the activities involved in a software development. The high-level activities of software specification, development, validation and evolution are part of all software process. Software specification is the process that establishing what services are required and the constraints on the system's operation and development.

The software requirements specification (SRS) is the official statement of what the system developers should implement. It should include both the user requirements for a system and a detailed specification of the system requirements. In a few cases, the user and system requirements may be integrated into a single description. In other cases, the user requirements are defined in an introduction to the system requirements specification. If there are a large number of requirements, the detailed system requirements may be presented in a separate document.

Specification and design are inextricably intermingled. Architectural design is essential to structure a specification and the

specification process. Formal specifications are expressed in a mathematical notation whose vocabulary, syntax and semantics are formally defined [1]. Usually both the system requirements and the system design are expressed in detail and carefully analysed and checked before implementation begins. If a formal specification of the software is developed, this usually comes after the system requirements have been specifying but before the detailed system design. There is a tight response loop between the detailed requirements specification and the formal specification. One of the main benefits of formal specification is its ability to uncover problems and ambiguities in the system requirements.

II. FORMAL SPECIFICATION

The principal value of using formal methods in the software process is that it forces an analysis of the system requirements at an early stage. Correcting errors at this stage is cheaper than modifying a delivered system. Formal methods consist of formal specification, specification analysis and proof, transformational development and program verification. Formal specification languages comprise VDM (Jones, 1980), Petri Nets (Carl Adam Petri, 1962), CSP (Hoare, 1985), Z (Spivey, 1992) and B (WordSworth, 1996). Formal specification techniques are most cost-effective in the development of critical systems where safety, reliability and security are particularly important.

Formal proving is a complete argument of mathematical representation and it is used to validate statement about system description. Usually, formal proving can be done manually. Nowadays, formal proving can be done with the support of formal method tools such as theorem prover tools. Theorem prover is a tool that implements automatic theorem proving need of user support. Regularly, developers cover a long times and looping process, so there might be a great possibility of mistakes. The proofs are efficiently when it been presented in a user-friendly approach and it should not be unreasonably large. Nevertheless, a lot of the proof that involved in software validation is naturally detailed, low-level and repetitious. So we can briefly state that it is unsuitable for human checking. Thus, formal proving supported by tool, which is not only reduce the possibility of mistakes but also not totally removes it [9]. Hence, the use of support tool is a main factor that can affect the acceptance of formal method practically [10].

The Z specification language is a way of decomposing a specification into small pieces called schemas [2]. Each piece can be linked with a commentary that gives explanation informally the importance of the formal mathematics. A schema is essentially the formal specification analogous to programming language subroutines that are used to structure a system, where the schemas are used to structure a formal specification. The Z is physically powerful on sets and functions. Generally, Z notation is use for sequential situation. We interested in using Z notation because it is a mature technique for model-based specification [8]. It combines formal and informal description and uses graphical highlighting when presenting specifications.

In this research, one of the theorem proving tool that has been choose is Z/EVES. Z/EVES can be applied in most process and need only a minimum background education to use it. It can be learned in a few months depending on the type of applications and can be run in various platforms consist of Unix, SunOS, Linux and Windows.

III. DNA DATABASE SYSTEM

DNA (Deoxyribonucleic acid) is a unique, quantifiable human characteristic, an amalgamation of nucleic acids that constitute genetic code. A DNA database is a gathering of genetic sequences that are organized, which it can easily be accessed, managed, and updated. The system architecture model for a basic DNA database system is divided into three parts consists of Add-DNA-Data,

Find-DNA-Data and Remove-DNA-Data. Table 1 describes the informal specifications of the DNA database [6].

TABLE 1
INFORMAL SPECIFICATIONS FOR A DNA DATABASE

Inputs	Processing	Outputs
Add a record		
Data	If the data is already stored in database, report 'already known'. Otherwise, report 'success'. But if there is no more room in database, report 'no room'.	report
Remove a record		
Data	If the data is already in database, report 'success'. But if the data is not in database, report 'not known'.	report
Find a record		
Data	If the data is in database, report 'success'. Otherwise, report 'not known'	report

IV. Z SPECIFICATION OF DNA DATABASE

In DNA database system, we shall need to deal with DNA data types and with they are representing by DNA sequences. Below is a declaration of Z specification for DNA database system:-

- A 'given set' called Data is to represent all data in the DNA database (it consists of dna type and it's sequences).

[ID, TYPE, NAME, SEQUENCES]

- Database of Data : DNA
- *REPORT* ::= *successs* | *alreadyKnown* | *notKnown*
- Note : Δ symbol means there is a change in dbDna

Ξ symbol means there is NO change in dbDna

The schema displays the main functional of the DNA database system that use for all data in database.

$$
\begin{aligned}
&\cup\_DNA\_\_\_\_\_\\
&\rightarrow KNOWN: \Pi\ ID\\
&\rightarrow TYPE: ID \clubsuit TYPE\\
&\rightarrow NAME: ID \clubsuit NAME\\
&\rightarrow SEQUENCES: ID \clubsuit SEQUENCES\\
&\cap\_\_\_\_\_\\
&\rightarrow KNOWN = \text{DOM } TYPE\\
&\rightarrow KNOWN = \text{DOM } NAME\\
&\rightarrow KNOWN = \text{DOM } SEQUENCES\\
&\angle\_\_\_\_\_
\end{aligned}
$$

A following below a schema for adding data into database, delete data from database and find data from database: -

- Adding data in DNA database – the data record will be added to the database if the record does not exist in database.

$$
\begin{aligned}
&\cup\_AddDNA\ \_\_\_\_\_\\
&\rightarrow \Delta DNA\\
&\rightarrow id?: ID\\
&\rightarrow type?: TYPE\\
&\rightarrow name?: NAME\\
&\rightarrow sequences?: SEQUENCES\\
&\rightarrow report!: REPORT\\
&\cap\_\_\_\_\_\\
&\rightarrow id? \text{ ™ } known\\
&\rightarrow type' = type\ \text{Y}\ \{(id? \div type?)\}\\
&\rightarrow name' = name\ \text{Y}\ \{(id? \div name?)\}\\
&\rightarrow sequences' = sequences\ \text{Y}\ \{(id? \div sequences?)\}\\
&\rightarrow report! = Success\\
&\angle\_\_\_\_\_
\end{aligned}
$$

- Deleting data in DNA database - the data record is allowed to delete if the data record is exist in database.

∪_*DelDNA* _____
→Δ*DNA*
→*id?:* *ID*
→ *report!:* *REPORT*
∩_____
→*id?* ε *known*
→*type'* = {*id?*} ψ *type*
→*name'* = {*id?*} ψ *name*
→*sequences'* = {*id?*} ψ *sequences*
∠_____

- Finding data in DNA database – the data record can be find depends on input from user.

∪_*FindDNA*_____
→Ξ*DNA*
→*id?:* *ID*
→*type!:* *TYPE*
→*name!:* *NAME*
→*sequences!:* *SEQUENCES*
∩_____
→*id?* ε *known*
→*type!* = *type* *id?*
→*name!* = *name* *id?*
→*sequences!* = *sequences* *id?*
∠_____

Below show how all of the above schemas are combines in a user interface: -

AddThenFind | *AddDNA* ; *FindDNA*
RAddDNA | *AddDNA* *f* *Success* ϖ *AlreadyKnown*

V. FORMAL VALIDATION OF DNA DATABASE

Validation is intended to ensure that the software system meets the customer expectations. It goes beyond checking that the system conforms to its specification to showing that the software does what the customer expects it to do. Formal design validation combines aspects of traditional checking and dynamic simulation based verification with the symbolic simulation and static analysis techniques of formal verification to provide optimized trade-offs in scalability and completeness so improving verification effectiveness.

Theorem proving involves verifying the truth of mathematical theorems that are postulated about the design using a formal specification language. It remains largely an academic area today and is unlikely to be viable for widespread commercial use in the near future. This section describes regarding the theorems that may represents the properties of operations in similarity string matching. This property can be proved by using initial state theorem.

theorem *InitIsOK*
E *DNA* ∞ *InitDNA*

a) Safety property of predicate DNA

theorem *DNAPredicate*
DNA ⇒ *known* = dom *type* *f* *known* = dom *name* *f* *known* = dom *sequences*

b) Safety property of predicate add record DNA

theorem *AddDNAPre*
A *DNA; id?: ID; type?: TYPE; name?: NAME; sequences?: SEQUENCES* ∞ *id?* ™ *known* ⇒ pre *AddDNA*

c) Safety property of predicate delete record DNA

theorem *DelDNAPre*
A *DNA; id?: ID* ∞ *id?* ε *known* ⇒ pre *DelDNA*

c) Safety property of predicate show/find record DNA

theorem *FindDNAPre*
FindDNA $\Rightarrow$ Ξ*DNA* f *id?* ε *ID* f *type!* ε *TYPE* f *name!* ε *NAME* f *sequences!* ε *SEQUENCES*

All of the theorems above have been proved using Z/EVES theorem prover. This shows that the Z specifications that have been discussed above are reliable.

VI. CONCLUSION

In this paper, we demonstrate a validation on Z formal specification of a case study in DNA similarity string matching using theorem proving technique. According to our experience, many theorems have been through a long and repetitious proving process. If the proving is done manually by humans, the possibility a mistake may be made is high. Using Z/EVES, not only this possibility can be reduced, the proving can be done fast and reliable. One of our future works shall deal with complete and precise specification and validation for distributed DNA database systems.

REFERENCES

[1] I. Sommerville, *Software Engineering*, 6th ed. UK: Addison-Wesley, 2001.

[2] J. Bowen, *Formal Specification and Documentation using Z: A Case Study Approach*: Thomson Publishing, 2003.

[3] A. V. C. Aho, M. J., "Efficient String Matching : An Aid to B ibliographic Search," *Communication of the ACM*, vol. 18, pp. 333-340, 1975.

[4] M. Y. M. S. M Nordin A Rahman, Aziz Ahmad, A Osman M Tap, "Automaton Based Filtering in Optimal DNA Sequence Similarity Search," presented at Proceeding 0f 1st Regional Conference on Computer Science & Technology, 2007.

[5] J. C. F. S. Dupuy, M.Chabre-Peccoud, Y.Ledru, "Formal And Informal Specifications : a proposal for a coupling," presented at 13th International Conference Software and Systems Engineering, Paris, France, 2000.

[6] M. N. A. R. M YAZID M SAMAN, AZIZ AHMAD AND A OSMAN M TAP, "A Minimum Cost Process in Searching for a Set of Similar DNA Sequences," in *5th WSEAS International Conference on Telecommunications and Informatics*. Istanbul, Turkey, 2006, pp. 348-353.

[7] D. W. C. Lok Lam Cheng, Siu-Ming Yiu, "Approximate String Matching in DNA Sequences," The University Of Hong Kong, 2003.

[8] M. Saaltink, "The Z/EVES System," presented at ZUM, LNCS, 1997.

[9] Bowen J.P and M.G. Hinchey, 1995. Ten Commandments of Formal Methods. Computer, 28(4): 56-63.

[10] Babich F. and L. Deotto, 2002. Formal Methods for Specification and Analysis of Communication Protocol. IEEE Communication Surveys & Tutorials., 4(1): 2-15.