International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:9, No:8, 2015

Validation of the Formal Model of Web Services Applications for Digital Reference Service of Library Information System

Zainab M. Musa, Nordin M. A. Rahman, Julaily A. Jusoh

Abstract—The web services applications for digital reference service (WSDRS) of LIS model is an informal model that claims to reduce the problems of digital reference services in libraries. It uses web services technology to provide efficient way of satisfying users' needs in the reference section of libraries. The formal WSDRS model consists of the Z specifications of all the informal specifications of the model. This paper discusses the formal validation of the Z specifications of WSDRS model. The authors formally verify and thus validate the properties of the model using Z/EVES theorem prover.

Keywords—Validation, verification, formal, theorem proving.

I. Introduction

FORMAL proving is the act of showing the correctness of a system with respect to a certain formal specification or property using mathematical methods. After the formal model of a program is built, a variety of properties can be validated over it. The formal specification of a system can also be verified to ensure its correctness and to prove its consistency and completeness using formal verification techniques before system design and implementation [1], [3].

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 or automatically using formal method tools such as theorem provers. Developers usually cover a long time while performing the theorem proving process, so there might be a great possibility of mistakes. The proofs are efficient when presented in a user-friendly approach and not in an unreasonable large size. However, a lot of the proofs that are involved in software validation are naturally detail, low-level and repetitious. So we can briefly state that it is unsuitable for human checking. Thus, formal proving supported by tool, do not only reduce the possibility of mistakes but also removes it totally. Hence, the use of support tool is a main factor that can affect the acceptance of formal method practically [1].

The Z specification language is a way of decomposing a specification into small pieces called schemas. Each piece can be linked with comments that give informal explanation about 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 and model-based specification. It combines formal and informal description and uses graphical highlighting when presenting specifications [2], [7].

In this paper, we validate the Z specifications of WSDRS model by using theorem proving technique based upon two aspects: the initial state and the pre-conditions. Validation of the initial state is to show that the Z specifications developed were consistent. While the validation of the pre-conditions is to show that the z specifications developed were complete, consistent and were applied in the right domain.

In order to implement the validation process, a few theorems will be developed for both aspects. Each theorem will be checked using Z/EVES theorem prover tool. The tool will help in reducing time, energy and mistakes compared to manual theorem proving which can be error full and tedious.

II.Theorem Proving (Deductive Verification)

Theorem-proving means that systems satisfy their specification, which given by temporal Logic formulas, using deductive (i.e. theorem proving) methods. It involves generating a collection of mathematical proof obligations from a system and its specifications, the truth of which imply conformance of the system to its specification, and discharging these obligations using theorem provers such as interactive theorem provers, automatic theorem provers, satisfiability modulo theories (SMT) solvers. This approach requires the user to understand in detail why the system works correctly, and to convey this information to the verification system, either in the form of a sequence of theorems to be proved or in the form of specifications of system components (e.g. functions or procedures) and subcomponents (such as loops or data structures) [4]. These techniques are not fully automatic and require user interaction and the effective guidance of a theorem-proving tool. It is the most powerful and least restricted verification technique because it can prove anything [5].

Theorem-provers are softwares that help in solving problems and answering questions that involve reasoning. The assistance can either be interactive, where one instructs the program to draw some conclusions, present them to the user, and then to ask for a new set of instructions; or fully automatic, where the program is assigned an entire reasoning task [5].

Zainab Musa Magaji is with the Universiti Sultan Zainal Abidin, Malaysia (e-mail: zeemusa5@gmail.com).

There are several automated theorem-proving tools with different properties used in theorem proving. A few examples are Z/EVES, E, Otter, SETHEO, Vampire, SPASS.

In this research work, the formal verification technique that would be used is theorem proving because it can be implemented on any logic in mathematics notations in Z language. Formal specification is complete and consistent if it can be proven as true. Completeness and consistency for Z specification can be articulated by proving the following aspects: initial state and pre-condition. First, each aspect is presented in a theorem form to be proved and then theorem-proving process will be carried out. Z/EVES theorem prover will be used because it is suitable for formal specification developed in Z language.

III. Theorem Proving (Initial State of the WSDRS Z Specifications)

Initial state theorem proving begins with the development of an initial state theorem. The skeleton for an initial state theorem is as:

Theorem TheoremName
E State . InitState

where Theorem Name: name of the theorem, State: state schema, InitState: Initial state schema [6].

Validation process of the initial state theorem begins with the development of the initial state schema. Next is the development of the initial state theorem depicted in Fig. 1 (a).

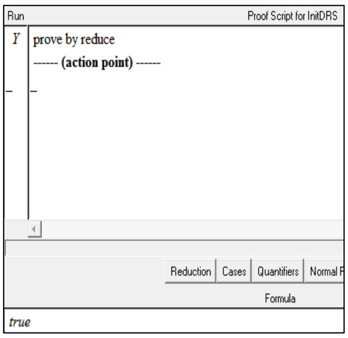
theorem *InitDRS* E *DRS'* ∞ *Ινιτ* (a)	 Run Proof Script for InitDRS Y prove by reduce ------ **(action point)** ------ *true* (b)

Fig. 1 Theorem Initial state for digital reference service web services of LIS

Fig. 1 (b) indicates that the theorem proves to be "true". This theorem proving process shows that the Z specifications that have been developed and discussed in our previous paper "Formal Specification of Web Services Applications for Digital Reference Services of Library Information System" are consistent and reliable.

IV.Theorem Proving Pre-Condition

The process of proving the pre-condition theorems of all the operation schemas that were discussed in the previous chapter is presented in this section. Theorem proving is done after all theorems that have been developed are free from any syntax errors.

Pre-condition theorem proving is to show that each operation is applied in the right domain. It begins with the development of pre-condition theorem using the following format.

Theorem TheoremName
A S; in? : N | P . pre Op

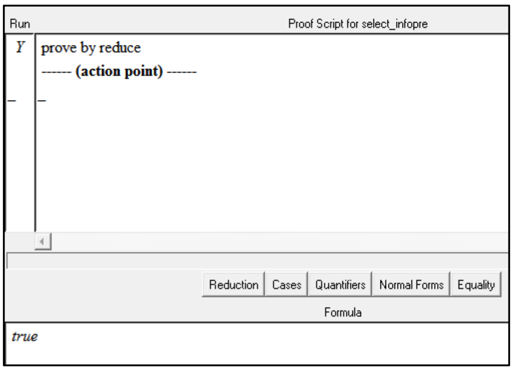
where TheoremName: name of the theorem, S: other schemas that are involved in the operation schema, In?: input variable in declaration part of the operation schemaN: type of the input variable in the declaration part of the operation schema, P: pre-condition of the operation, Op: operation schema that will be proved to be with or without domain error .

A correct precondition theorem, of the above form can be a useful form of documentation of a specification [6]. It is a state before an operation and it relates to a state, which might happen after an operation. State before or after an operation must comply with a specified property.

If the result of a pre-condition theorem proving is true, then the pre-condition operation schema relates accurately to a state after an operation. Otherwise, if the theorem proving is not true then the precondition of the operation schema is not consistent and therefore the operation schema or the precondition theorem of the schema must be corrected due to unidentified problem in the domain.

There are eighteen pre condition theorems that have been developed for web services of digital reference service of LIS. These theorems were developed based on the eighteen operation schemas of the Z specifications discussed in our previous paper "Formal Specification of Web Services Applications for Digital Reference Services of Library Information System". Below are the precondition theorems developed for the operation schemas. Each theorem is followed by a screenshot that displays its proof script as validated by the Z/EVES theorem prover using prove by reduce command. All the theorems are proved 'True'. This result showed that the operations in Z specifications for WSDRS of LIS were applied in the right domain. Moreover, it means that the pre-condition operation schemas of the specifications are related accurately to a state after an operation and have comply with the property been specified in the state schema. So, it can be concluded that the Z specifications that are discussed in the said previous paper are consistent.

1. Theorem *select_infopre*
A *DRS; patron?: PERSON; active_system_ws!: SYSTEM_WEBSERVICES;*
action!: ACTION
| *patron?* ε *users*
ƒ selection patron? = infor_service_ws
ƒ action! = invoke_infor_service_ws
ƒ active_system_ws! = infor_service_ws ∞ pre *Select_info*



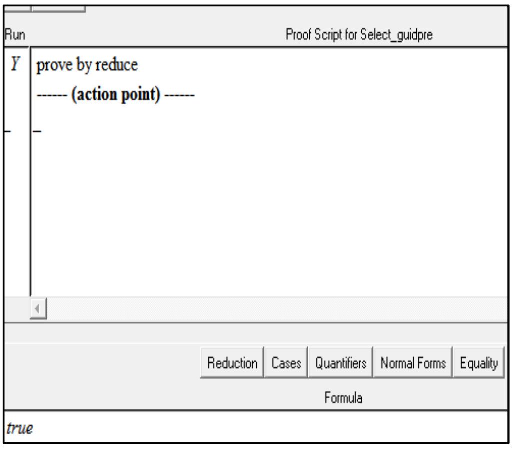
prove by reduce
------ **(action point)** ------

true

Fig. 2 Proof script for select_infopre theorem

2. Theorem *Select_guidpre*

A *DRS; patron?: PERSON; active_system_ws!: SYSTEM_WEBSERVICES;*
action!: ACTION
| *patron?* ε *users*
f selection patron? = guidance_ws
f action! = invoke_guidance_ws
f active_system_ws! = guidance_ws ∞ pre *Select_guid*



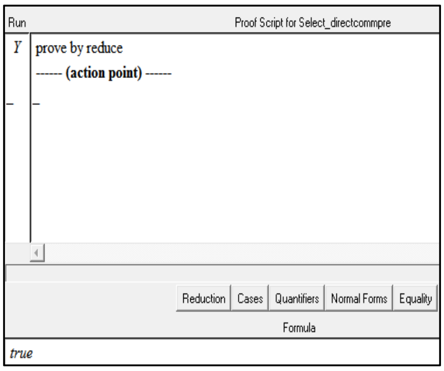
prove by reduce
------ **(action point)** ------

true

Fig. 3 Proof script for select_guidpre theorem

3. Theorem *Select_directcommpre*

A *DRS; patron?: PERSON; active_system_ws!: SYSTEM_WEBSERVICES;*
action!: ACTION
| *patron?* ε *users*
f selection patron? = direct_comm_ws
f action! = invoke_direct_com_ws
f active_system_ws! = direct_comm_ws ∞ pre *Select_directcomm*



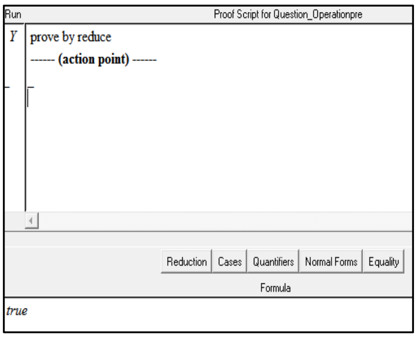
prove by reduce
------ **(action point)** ------

true

Fig. 4 Proof script for select_directcommpre theorem

4. Theorem *Question_Operationpre*

A *DRS; patron?: PERSON; patronquestion?: MESSAGE*
| *patron?* ε *users*
f (*selection patron? = infor_service_ws*
ϖ *selection patron? = guidance_ws*) ∞ pre *Question_Operation*



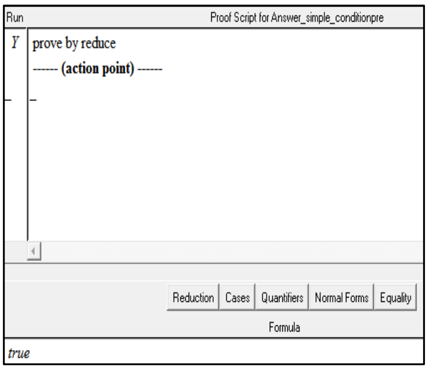
prove by reduce
------ **(action point)** ------

true

Fig. 5 Proof script for Question_Operationpre theorem

5. Theorem *Answer_simple_conditionpre*

A *DRS; patron?: PERSON; patronquestion?: MESSAGE; correctanswer!: MESSAGE;*
clearity!: MESSAGE__STATUS
| *patron?* ε *users*
f clearity! = questionClear
f patronquestion? ε *questions*
f correctanswer! = answerbase patronquestion?
∞ pre *Answer_simple_condition*

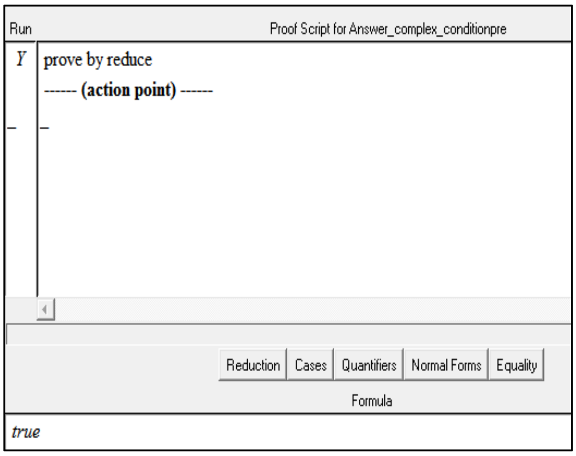


prove by reduce
------ **(action point)** ------

true

Fig. 6 Proof script for Answer_simple_conditionpre theorem

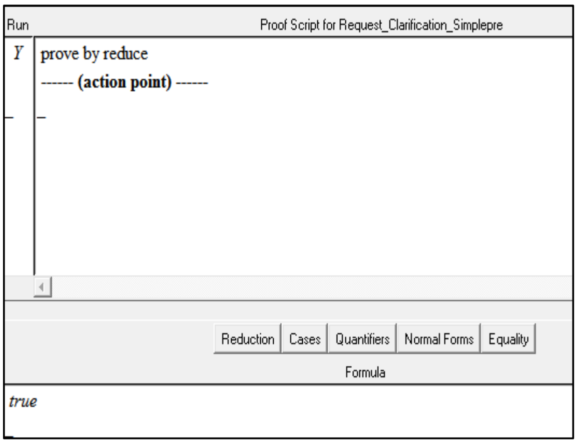
6. Theorem *Answer_complex_conditionpre*
A *DRS; patron?: PERSON; patronquestion?: MESSAGE; correctanswer!: MESSAGE;*
action!: ACTION; clearity!: MESSAGE_STATUS; outside_ws?: WEBSERVICES;
outsideAnswerbase?: MESSAGE ♣ *MESSAGE*
| *patron?* ε *users*
ƒ *clearity! = questionClear*
ƒ *patronquestion?* ™ *questions*
ƒ *action! = invoke_other_ws*
ƒ *outside_ws?* ε *outsideWebservices*
ƒ *patronquestion?* ε dom *outsideAnswerbase?*
ƒ *correctanswer! = outsideAnswerbase? patronquestion?*
∞ pre *Answer_complex_condition*



prove by reduce
------ **(action point)** ------
–
true

Fig. 7 Proof script for Answer_complex_conditionpre theorem

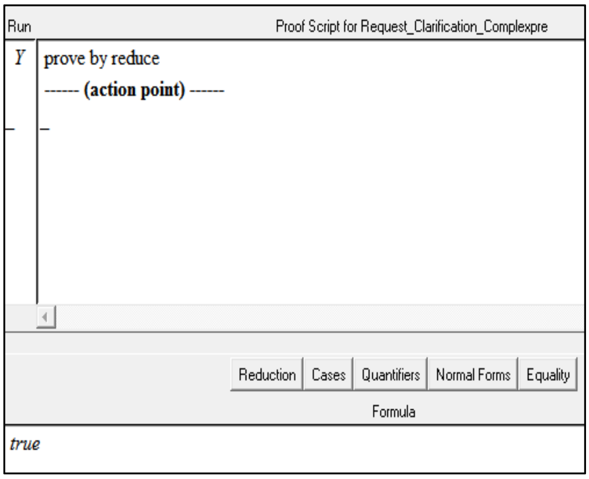
7. Theorem *Request_Clarification_Simplepre*
A *DRS; patron?: PERSON; requestClarificationMessage?: MESSAGE;*
clearity!: MESSAGE_STATUS
| *patron?* ε *users* ƒ *clearity! = questionNotClear* ƒ *call_out* = 0
∞ pre *Request_Clarification_Simple*



prove by reduce
------ **(action point)** ------
–
true

Fig. 8 Proof script for Request_Clarification_simplepre theorem

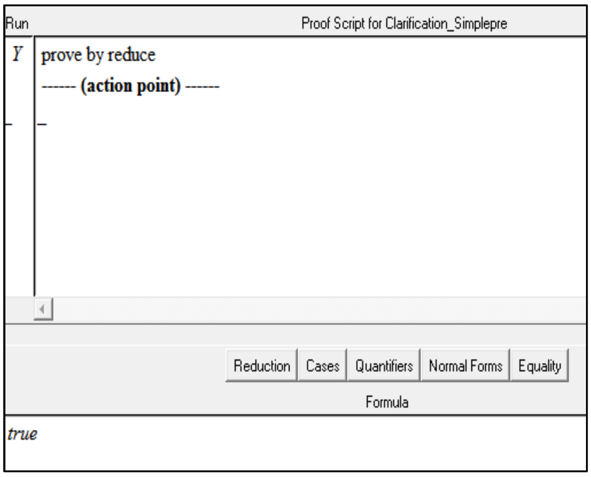
8. Theorem *Request_Clarification_Complexpre*
A *DRS; patron?: PERSON; requestClarificationMessage?: MESSAGE;*
clearity!: MESSAGE_STATUS; outside_ws?: WEBSERVICES
| *patron?* ε *users*
ƒ *clearity! = questionNotClear*
ƒ *call_out* ⌊ 0
ƒ *outside_ws?* ε *outsideWebservices*
ƒ *outside_ws?* ε ran *call_out* ∞ pre
Request_Clarification_Complex



prove by reduce
------ **(action point)** ------
–
true

Fig. 9 Proof script for Request_Clarification_Complepre theorem

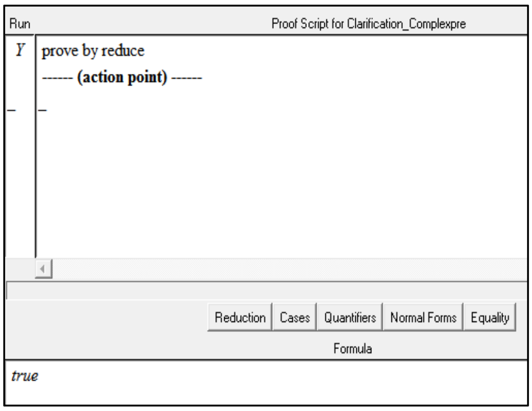
9. Theorem *Clarification_Simplepre*
A *DRS; patron?: PERSON; ClarificationMessage?: MESSAGE*
| *patron?* ε *users* ƒ *call_out* = 0 ∞ pre *Clarification_Simple*



prove by reduce
------ **(action point)** ------
–
true

Fig. 10 Proof script for Clarification_Simplepre theorem

10. Theorem *Clarification_Complexpre*
A *DRS; patron?: PERSON; ClarificationMessage?: MESSAGE;*
outside_ws?: WEBSERVICES
| *patron?* ε *users*
ƒ *call_out* ⌊ 0
ƒ *outside_ws?* ε *outsideWebservices*
ƒ *outside_ws?* ε ran *call_out* ∞ pre *Clarification_Complex*

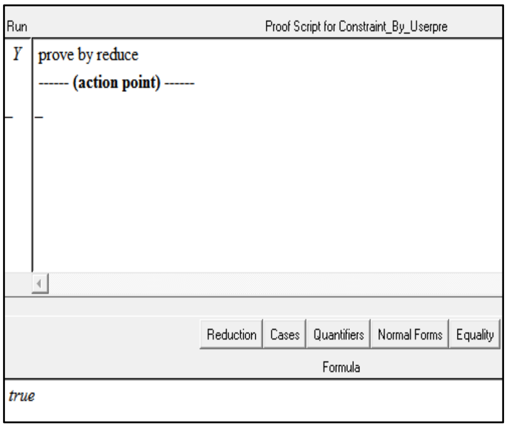


Y prove by reduce
------ **(action point)** ------

true

Fig. 11 Proof script for Clarification_Complepre theorem

11. Theorem *Constraint_By_Userpre*
A *DRS; patron?: PERSON; constraintMessage?:*
MESSAGE | *patron?* ε *users*
∞ pre *Constraint_By_User*

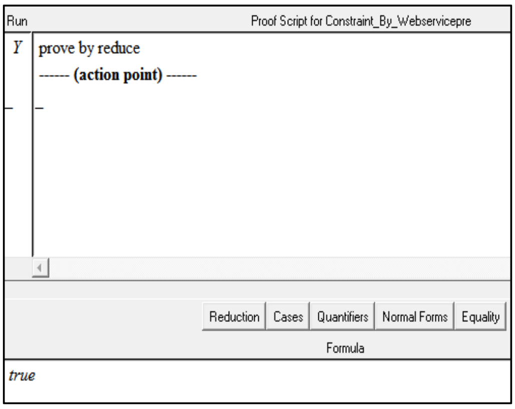


Y prove by reduce
------ **(action point)** ------

true

Fig. 12 Proof script for Constraint_By_Userpre theorem

12. Theorem *Constraint_By_Webservicepre*
A *DRS; patron?: PERSON; constraintMessage?:*
MESSAGE | *patron?* ε *users*
∞ pre *Constraint_By_Webservice*

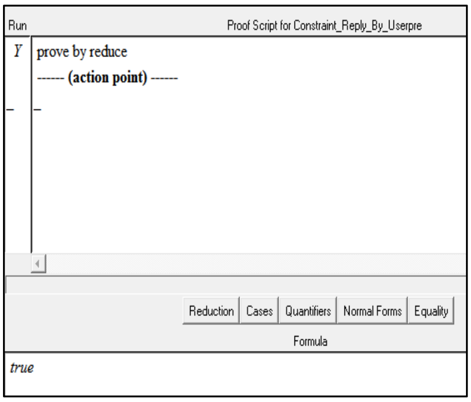


Y prove by reduce
------ **(action point)** ------

true

Fig. 13 Proof script for Constraint_By_Webservicepre theorem

13. Theorem *Constraint_Reply_By_Userpre*
A *DRS; patron?: PERSON; constraintReplyMessage?:*
MESSAGE;
constraintStatus?: MESSAGE_STATUS;
acceptanceFlag?: RESPONSE
| *patron?* ε *users*
f (**if** *acceptanceFlag? = True*
then *constraintStatus? = Accepted*
else *constraintStatus? = NotAccepted*) ∞ pre
Constraint_Reply_By_User

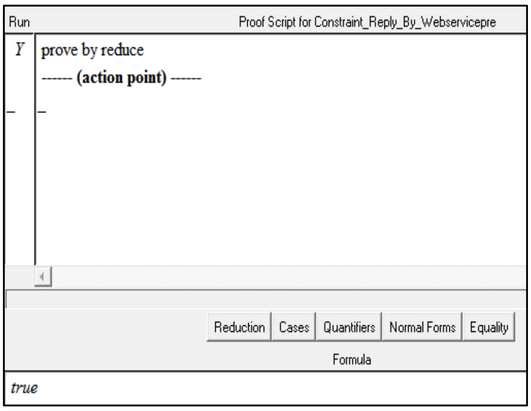


Y prove by reduce
------ **(action point)** ------

true

Fig. 14 Proof script for Constraint_Reply_Bye_Userpre theorem

14. Theorem *Constraint_Reply_By_Webservicepre*
A *DRS; patron?: PERSON; constraintReplyMessage?:*
MESSAGE;
constraintStatus?: MESSAGE_STATUS; acceptanceFlag?:
RESPONSE
| *patron?* ε *users*
f (**if** *acceptanceFlag? = True*
then *constraintStatus? = Accepted*
else *constraintStatus? = NotAccepted*)
∞ pre *Constraint_Reply_By_Webservice*



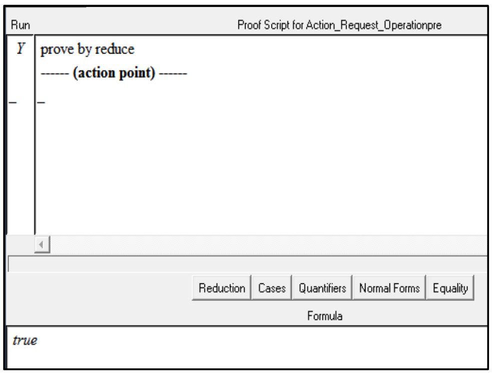
Y prove by reduce
------ **(action point)** ------

true

Fig. 15 Proof script for Constraint_Reply_Bye_Webservicepre theorem

15. Theorem *Action_Request_Operationpre*
A *DRS; patron?: PERSON; actionRequestMessage?:*
MESSAGE;
actionRequest?: ACTION
| *patron?* ε *users*

ƒ (*actionRequest? =*
suspend_process_until_another_time
ϖ *actionRequest? = resume_process*
ϖ *actionRequest? = reset_activity_time*
ϖ *actionRequest? = close_transaction*
ϖ *actionRequest? = request_for_status_report*)
∞ pre *Action_Request_Operation*



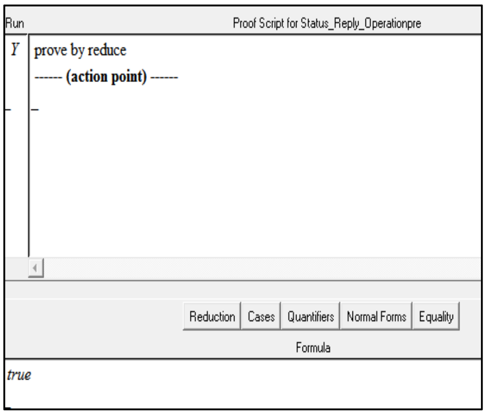
Run   Proof Script for Action_Request_Operationpre

Y   prove by reduce
------ **(action point)** ------
–   –

true

Fig. 16 Proof script for Action_Request_Operationpre theorem

16. Theorem *Status_Reply_Operationpre*
A *DRS; patron?: PERSON; statusReplyMessage?: MESSAGE;*
actionRequestStatus?: MESSAGE_STATUS; statusReply?: RESPONSE
| *patron?* ε *users*
ƒ (**if** *statusReply? = Success*
then *actionRequestStatus? = Accepted*
else *actionRequestStatus? = NotAccepted*)
∞ pre *Status_Reply_Operation*



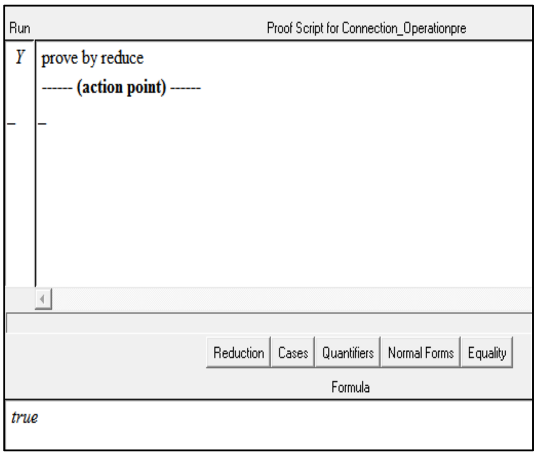
Run   Proof Script for Status_Reply_Operationpre

Y   prove by reduce
------ **(action point)** ------
–   –

true

Fig. 17 Proof script for Status_Reply_Operationpre theorem

17. Theorem *Connection_Operationpre*
A *DRS; patron?: PERSON; Select_Comm?: PERSON* ♣ *COMM_MEDIUM; action?: ACTION*
| *patron?* ε *users*
ƒ selection patron? = direct_comm_ws
ƒ patron? ε dom *Select_Comm?*
ƒ (**if** *Select_Comm? patron? = Call*
then *action? = connect_video_and_voice_call_service*
else *action? = connect_text_chat_service*) ∞ pre *Connection_Operation*



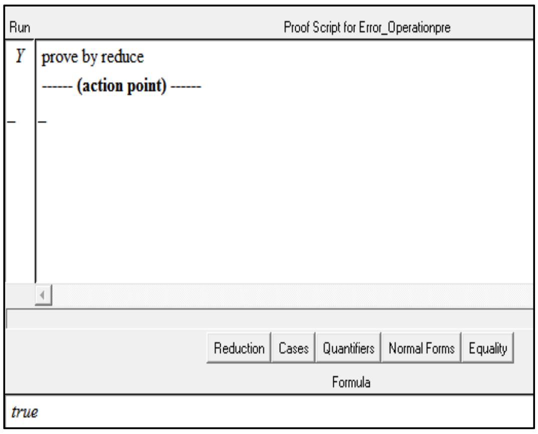
Run   Proof Script for Connection_Operationpre

Y   prove by reduce
------ **(action point)** ------
–   –

true

Fig. 18 Proof script for Connection_Operationpre theorem

18. Theorem *Error_Operationpre*
A *DRS; patron?: PERSON; report!: RESPONSE; clearity?: MESSAGE__STATUS*
| *patron?* ε *users ƒ clearity? = questionEmpty ƒ report! = questionInvalid*
∞ pre *Error_Operation*



Run   Proof Script for Error_Operationpre

Y   prove by reduce
------ **(action point)** ------
–   –

true

Fig. 19 Proof script for Error_Operationpre theorem

Figs. 2-19, it is clear that the proving of all the precondition theorems returns true. This means that the specifications we have developed which are defined in our previous paper titled “Formal Specification of Web Services Applications for Digital Reference Services of Library Information System” are correct, consistent and are applied in the right domain. This means that the model, which we develop for web services application for digital reference services of the library information system, is valid. Thus, it can be adopted and adapted by any library to provide a networked asynchronous digital reference service across the globe.

V. Conclusion

This paper demonstrated that all the theorems developed are proved to be true and are found to be error free, consistent and in the correct domain. Therefore, the Z specifications for web service applications for digital reference service of LIS are reliable as such they can be transformed into programming. At the same time, the ambiguity problem of informal specifications that may occur to the developers is overcome. Therefore, program development time and errors occurrence will decrease.

References

[1] Jusoh J. A. (2009). *Formal Specification and Validation for Pattern Scanning.* (Master Thesis, Universiti Malaysia Terengganu).

[2] Jusoh J.A., Saman M.Y.M. and Man M. (2011). "Formal Validation of DNA Database Using Theorem Proving Technique". In *International Journal of the Computer, the Internet and Management.* 19:74 – 78.

[3] Li G. (2010). *Formal Verification of Programs and Their Transformations.* (PhD Thesis, University of Utah).

[4] Pederson D. O. (2010). 'Introduction to Formal Verification. Center for Electronic Systems Design". *Http//:embedded.eecs.berkeley.edu/research/vis/doc/VisUser/vis-user/node4.html/accessed on 04/07/2014.*

[5] Pnueli A. (2002). "Deductive Verification in Action. Analysis of Reactive Systems, NYU, Fall, 2008". *http://www.wisdom.weizman.ac.il/~amir/course02a/header.html/accessed on 04/07/2014.*

[6] Saaltink M. (1999). "Proving Theorems" in *The Z/EVES 2.0 User's Guide* Ch. 5.

[7] Spivey J. M. (1998). "Tutorial Introduction" in *The Z Notation: A Reference Manual.* pp 1-17.