Trigger2B: A Tool Generating Event-B Models from Database Triggers

Hong Anh Le(✉)

Hanoi University of Mining and Geology, 18 Pho Vien, Bac Tu Liem, Hanoi, Vietnam
lehonganh@humg.edu.vn

Abstract. Triggers are commonly used many traditional database applications that can be checked if they are correct after execution or manual inspection. Formal methods are techniques complementing to testing that ensure the correctness of software. In practical aspect, one limitation of formal techniques is the complexity that makes software developers lazy to use in the development. This paper introduces a tool named Trigger2B which partly supports for translating DML triggers to Event-B models. From the targeted model, we can verify the data constraints and detect infinite loops of trigger execution with RODIN. Finally, we take an example for illustration purpose. The proposed tool overcomes the complexity of formal modeling and makes them practical in the development.

Keywords: Trigger · Event-B · Formal modeling · Generation

1 Introduction

Database is one of the most important and essential aspects in software development. The relational database has shown preeminent advantages as simple structure, easy access, and high performance. Nowadays, relational database become the most widely used in many software and the number of Database Management Systems (DBMS) increase such as MySQL, PostgreSQL, SQLite, SQL server, Oracle, etc. DBMS are passive as they execute commands when applications or users perform appropriate queries. However, in many cases, we want DBMS perform some tasks, for example, we want to record login and logout of table users. For that reason, most of modern DBMS include these features as triggers (or active rules) that monitor and react to specific events happened in and outside the system. They also use triggers to implement automatic tasks when a predefined event occurs.

DBMS usually have two types of triggers: data manipulation language (DML) and system triggers. The former is fired whenever the DML statements such as *deleting*, *updating*, and *insert* statements are executed, the latter is performed in case that system events or data definition language (DDL) ones occur. A trigger has the form of an Event-Condition-Action (ECA) rule informally written as "if a set of *events* occur and a set of *conditions* hold, then perform *actions*".

P. C. Vinh and A. Rakib (Eds.): ICCASA 2020/ICTCC 2020, LNICST 343, pp. 103–112, 2021.
https://doi.org/10.1007/978-3-030-67101-3_8

It is made of a block of code and has a syntax, for example, an Oracle trigger is similar to a stored procedure containing blocks of PL/SQL code. Trigger codes are human readable and do not have any formal semantic. The fact that the main purpose of trigger is maintaining the integrity of the information on the database. These weaknesses come from informal semantic of the trigger. Therefore, we only can verify that if a trigger terminates or conflicts to integrity constraints after executing it or with human inspection step by step. As a result, we cannot verify the correctness of the data system as we do not have any supported tool which used to prove triggers. Then, the problem needed to resolve is that how to specify and verify database system formally.

Many researchers have been working on analyzing triggers (or active rules). The research results of [14,15], proposed in the early 90's, transformed ECA rules to some types of graphs and applied various static analysis techniques to check properties such as *redundancy*, *inconsistency*, *incompleteness*, and *termination*. [6] proposed a technique, based on relational algebra, to check if active rules are terminated or confluent. Formal methods for techniques to mathematically build rigorous models of software and hardware systems. Complement to system testing, formal methods use mathematics proofs to verify the system behavior correctness. In this direction, Eun-Hye Choi et al. [10] and Chavarria et al. [7] addressed to both *termination* and *confluence* properties using model checking techniques.

Our previous work [13] initially proposed to use Event-B to formalize and verify a database triggers system at early design phase. The main idea of the method comes from the similar structure and working mechanism of Event-B events and database triggers. We presented a set of translation rules to translate a database system including triggers to an Event-B model. In practical aspect, however, one limitation of formal techniques is the complexity that makes software developers lazy to use in the development. In this paper, we present the design and implementation of a tool called Trigger2B which partly supports automatic modeling process with the RODIN. In the supporting tool Rodin, almost proofs are discharged automatically, hence it reduces complexity in comparison with manual proving. The tool makes the proposed method feasible in the database development process.

The remainder of this paper is organized as follows. Section 2 provides background of database triggers and Event-B. In Sect. 3, we present the architecture of the tool. In Sect. 4, we present the tool Trigger2B, which supports for partly automatic translation. Section 6 summarizes the related work. We conclude the paper and present the future work in Sect. 7.

2 Background

In this section, we first briefly introduce about database triggers and their SQL syntax. Then we give an overview of Event-B formal method.

2.1 Database Triggers and ANTLR

A relational database system, based on the relational model, consists of collections of objects and relations, operations for manipulation and data integrity for accuracy and consistency. Modern relational database systems include active rules as database triggers which response to events occurring inside or outside of the database. Triggers are commonly used in some cases: to audit the process, to automatically perform an action, and to implement complex business rules. The structure of a trigger has the form: *rule_name:: Event(e) IF condition DO action.*

The definition of SQL:1999 trigger has the syntax as follows:

```
CREATE [OR REPLACE] TRIGGER <trigger_name>

{BEFORE|AFTER} {INSERT|DELETE|UPDATE}
ON <table_name>

[REFERENCING [NEW AS <new_row_name>]
[OLD AS<old_row_name>]]

[FOR EACH ROW
[WHEN (<trigger_condition>)] ]

<trigger_body>
```

ANTLR [1] stands for ANother Tool for Language Recognition that is a powerful parser generator for text or binary files. It uses an input user-defined grammar to create a parse tree or abstract syntax tree. ANTLR v4 supports several languages such as Java, C#, Python, etc. The following snippets show a basic grammar named Hello

```
grammar Hello;
r : 'hello' ID;
ID : [a-z]+ ;
WS : [ \t\r\n]+ -> skip ;
```

2.2 Event-B and Rodin

Event-B [3] is a formal method for system-level modeling and analysis. Key features of Event-B are the use of set theory as a modeling notation, the use of refinement to represent systems at different abstraction levels, and the use of mathematical proof to verify consistency between refinement levels. A basic structure of an Event-B model consists of MACHINE and CONTEXT.

An Event B CONTEXT describes a static part where all the relevant properties and hypotheses are defined. A CONTEXT consists of carrier sets, constants, axioms. Carrier sets, denoted by s, are represented by their names, and are non-empty. Different carrier sets are completely independent. The constants c are

defined by means of a number of axioms $P(s, c)$ also depending on the carrier sets s.

A MACHINE is defined by a set of clauses. A machine is composed of variables, invariants, theorems and events. Variables v are representing states of the model. Invariants $I(v)$ yield the laws that state variables v must always be satisfied. Each event has the form evt = any x where $G(x, v)$ then $A(x, v, v')$ end where x are local variables of the event, $G(x, v)$ is a guard condition and $A(x, v, v')$ is an action. An event is enabled when its guard condition is satisfied. The event action consists of one or more assignments. We have three kinds of assignments for expressing the actions associated with an event: (1) a deterministic multiple assignment $(x := E(t, v))$, (2) an empty assignment (skip), or (3) a non-deterministic multiple assignment $(x : |P(t, v, x'))$.

Rodin [4], built on top a Eclipse, is a part of European Deploy project. It is customizable by a sets of plug-ins tools which support for modeling and analyzing with Event-B. A Rodin project contains seven file XML-based divided into three groups shown in Table 1 Group unchecked files contain user input text, and files in two group static checked and proof are automatically generated by static checker and provers, respectively.

Table 1. Rodin project file types

File extension	Root element type	Contents	Group
.bum	IMachineRoot	Event-B machine	Unchecked
.buc	IContextRoot	Event-B context	Unchecked
.bcm	ISCMachineRoot	Event-B checked machine	Static checked
.bcc	ISCContextRoot	Event-B checked context	Static checked
.bpo	IPORoot	Event-B proof obligations	Proof
.bpr	IPRRoot	Event-B proofs	Proof
.bps	IPSRoot	Event-B proof statuses	Proof

3 Translating Triggers to Event-B

In this section, we recall translation rules from database trigger system to Event-B models which have been defined in our previous work [13]. A database trigger is translated to an Event-B event where trigger conditions are guards of the event.

The rules translating the *Action* part of a trigger are described as follows:

– Insert: This statement has the form "Insert into T values $(val1, ..valn)$", where $val1, .., valn$ are column values of the new record of the table T. We encode this new row as a parameter $r \in T$ of the event. More specifically, the translated event has the form Evt= **Any** r **Where** $r \in T \wedge e \wedge c$ **Then** $t := t \cup r$.

– Delete: This statement is generally written in the form: "Delete from T where $column1 = some\_value$". It will delete the record that has the first column's value equal to $some\_value$. We add a parameter for the event representing the value $some\_value$. The event is specified in detail as follows Evt= **Any** v **where** $v \in TYPE_1 \wedge e \wedge c$ **Then** $t := t - f(v)$.
– Update: The general syntax of this statement is "Update T set $column1 = value1$, $column2 = value2$ where $column1 = some\_value$". This statement will update a record where value of the first column is equal to $some\_value$, similar to the case of delete statement, we encode the input values as parameters of the event. The description of the translated event is as follows: Evt= **Any** $v1, v2$ **where** $v1 \in TYPE_1 \wedge v2 \in TYPE_2 \wedge e \wedge c$ **Then** $t := \{1 \mapsto v1, 2 \mapsto v2\} \oplus t$.

4 Architecture and Design

Following the method presented in Sect. 3, we implement a tool called Trigger2B[1] to support designing and modeling a database system including trigger. In this section, we present the architecture and detailed design of the tool.

4.1 Architecture

This tool can generate multiples XML-based format output which can be used later in verification phase with Event-B supporting tools such as RODIN platform. The architecture of this tool is illustrated in Fig. 1. The UI provides a graphic interface for users. Module Data Adapter is responsible connect to the database and extract metadata that is SQL statements describe about given database schema. SQL extractor is the module recognize SQL structure then extract database model. Translator component is the heart of our tool. It translates database models to event-b models. Shared Component shares common libraries is used by different modules. Module Event-B Builder exports handles exporting event-b model received from previous module into Rodin project. Finally, system is handled and logged error by System Handler & Logging module.

Figure 2 below described process of translating between models of this tool has been implemented. The processing is done through the six major steps as in figure. Firstly, users browse a database want to verify. Then, our tools will connect to database and dump the database in an SQL text format. After that, SQL structure is parsed to extract database model. The result of the previous process is used as input in the translation process. Finally, the Trigger2B tool will export results into a Rodin project. Users can import the archived project into Rodin platform and continue to perform the verification and modeling.

[1] The Trigger2B source code can be found at: https://github.com/anhfit/Trigger2B.

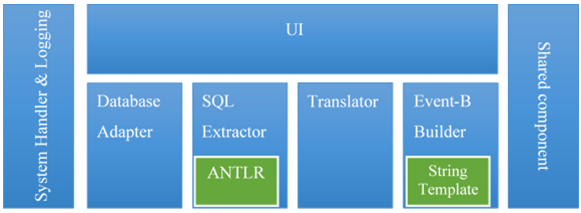


Fig. 1. Architecture of the proposed tool

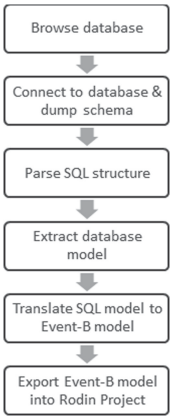


Fig. 2. Processing flow of the tool

4.2 Component Design

Database Adapter. Database Adapter component, based on JDBC technology, connects to the DBMS and extracts database schema. For each DBMS, we need to define a specific JDBC driver. Subsequently, Trigger2B is flexible enough to handle with various DBMS.

SQL Extractor. The input of this component is text-based values of database schema received from Database Adapter component. SQL Extractor makes this value in standard form, removes redundant or duplicated data, parses SQL statements, and extracts needed information. The process of extraction is illustrated in Fig. 3.

Translator. Translator component is the most important part of the tool following the method presented in Sect. 3. This component translates database model achieved from SQL Extractor to Event-B models using the proposed rules.

Event-B Builder. This component exports constructed Event-b models into Rodin project. Basically, a Rodin project contains seven XML-based file types

illustrated in Sect. 2. Group unchecked files contain user input text, and files in two group static checked and proof are automatically generated by static checker and provers, respectively. So we only need to export into .bum and .buc files. The Event-B Builder component uses a template engine named StringTemplate [2] that provides features like combining one or more templates with a data model to produce target Rodin documents. This engine is provided as the ANTLR library built-in.

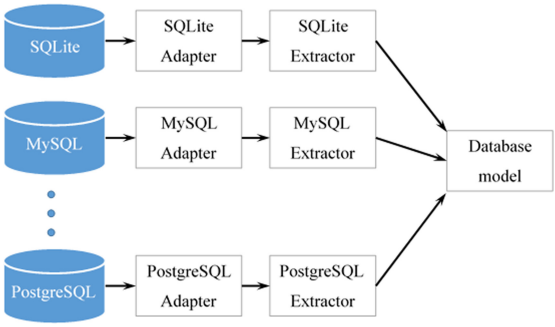


Fig. 3. Constructing database model

5 Implementation and Results

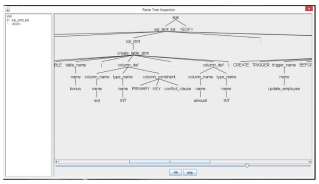
The Trigger2B tool is implemented in Java programming language and structured several packages corresponding to specific component mentioned above. In order to parse trigger statements, we define grammar files which contains grammars and rules for trigger SQL syntax. The snippet codes below show a part of grammar file for defining SQLite trigger parser.

```
grammar SQLite;

create_trigger_stmt

    : CREATE ( TEMP | TEMPORARY )? TRIGGER ( IF NOT EXISTS )?

        ( database_name '.' )? trigger_name

    ( BEFORE | AFTER | INSTEAD OF )?

    ( DELETE | INSERT | UPDATE ( OF column_name ( ',' column_name )* )? )

    ON ( database_name '.' )? table_name

    ( FOR EACH ROW )?

    ( WHEN expr )?
```

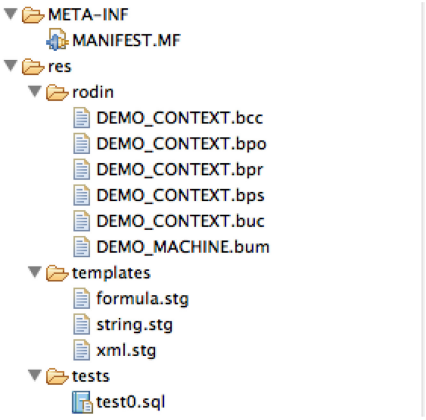
```
        BEGIN ( ( update_stmt
| insert_stmt
| delete_stmt
| select_stmt ) ';' )+

END

;
```

Even though, ANTLR provides an automatic tree-walking mechanism, Trigger2B uses -visitor option to generate a visitor interface from the defined a grammar and invokes explicitly interface *ParseTreeVisitor.visit(node)* method on child nodes.

Figure 4 depicts the results when we run Trigger2B tool in a case study of human resource management. It shows the parsed tree from trigger statements and Rodin project files which are generated automatically.



(a) Parsed tree from triggers



(b) Monitoring with on-line sources

Fig. 4. Generated Rodin project

6 Related Work

From theoretical aspect, there are a number of works proposed for checking and verifying database trigger. In [14,15], Sin-Yeung Lee and Tok-Wang introduced algorithms to detect the correctness of updating triggers. R. Manicka Chezian and T. Devi [17] introduced a new algorithm which does not pose any limitation on the number of rules but it only emphasizes on algorithms detecting the termination of the system. Baralis [6] improved existing techniques and proposed propagation algorithms, based on relational algebra, to statically check if active rules are terminated or confluent. This approach, can be applied widely for active

database rule languages and for trigger language (SQL:1999). Zang *et al.* [18] proposed an approach to checking structural errors such as inconsistency, circularity, and redundancy of ECA rule-based systems. Their method classifies three different levels of verification and builds an EA tree to check each level. These works focused on theoretical aspects and did not propose any practical tool.

Another direction is using model checking techniques such as Tarek and Huth [11] presented an abstract modeling framework for active database management systems and implemented a prototype of a Promela code generator. Choi *et al.* [9] proposed a general framework for modeling active database systems and rules. Their framework is feasible by using a model checking tool, e.g., SPIN. Ksystra [12] proposed a method to express and verify safety properties of reactive rules which also are ECA rules. It provides verification mechanism of termination, confluence and safety properties using CafeOBJ method. With these direction, practical tools have been mentioned and proposed, but they required users to specify the model manually.

Chavarria and Li [7,8,16] have had a series of work which focus on verifying active rules by using conditional colored Petri nets. They also proposed a tool called ECAPNSim which supports to achieve rules from a text file, then translate them to a Petrinet automatically. Our tool is different because we use Event-B and Rodin for modeling and verification.

Regarding to support tools for modeling in Event-B and Rodin. Most of tools are in the reversed direction, i.e., code generation from Event-B. Most related work is UB2DB [5] which assists users on incremental database design with Rodin. It generates database models from Event-B ones. To the best of our knowledge, Trigger2B is the first tool for translating SQL-based statements to Event-B.

7 Conclusions and Future Work

Support tools for formal modeling and verification are significantly important in software development. These tool will help engineers to ease the complexity of formalization. In this paper, we presented the Trigger2B tool for modeling and checking database triggers. The proposed tool accesses DBMS and automatically generates the Event-B models and corresponding Rodin project. The database developers using Trigger2B do not need much skills and do not have to spend time in detail modeling with Event-B. The current implementation, however, works only with SQLite. The tool needs to be improved with more complex cases and more DBMS. We intend to make it available as a Rodin plug-in. Enhancing the translation rules to only triggers but also a database schema is one of our future work.

Acknowledgments. This work is supported by the project no. CT.2019.01.05 granted by Ministry of Education and Training (MOET).

References

1. Anltr (2014). http://www.antlr.org
2. String template (2014). http://www.stringtemplate.org
3. Abrial, J.-R.: Modeling in Event-B - System and Software Engineering. Cambridge University Press, Cambridge (2010)
4. Abrial, J.-R., Butler, M., Hallerstede, S., Voisin, L.: An open extensible tool environment for Event-B. In: Liu, Z., He, J. (eds.) ICFEM 2006. LNCS, vol. 4260, pp. 588–605. Springer, Heidelberg (2006). https://doi.org/10.1007/11901433_32
5. Al-Brashdi, A., Butler, M., Rezazadeh, A.: UB2DB Rodin plug-in for automated database code generation. In: 6th International ABZ Conference ASM, Alloy, B, TLA, VDM, Z, 2018Programme (2018)
6. Baralis, E.: Rule analysis. In: Paton, N.W. (ed.) Active Rules in Database Systems. MCS, pp. 51–67. Springer, New York (1999). https://doi.org/10.1007/978-1-4419-8656-6_3
7. Chavarría-Báez, L., Li, X.: Verification of active rule base via conditional colored petri nets. In: SMC, pp. 343–348 (2007)
8. Chavarriéa-Baéez, L., Li, X.: Termination analysis of active rules - a petri net based approach. In: IEEE International Conference on Systems, Man and Cybernetics, SMC 2009, pp. 2205–2210, October 2009
9. Choi, E.-H., Tsuchiya, T., Kikuno, T.: Model checking active database rules. Technical report, AIST CVS, Osaka University, Japan (2006)
10. Choi, E.-H., Tsuchiya, T., Kikuno, T.: Model checking active database rules under various rule processing strategies. IPSJ Digit. Cour. **2**, 826–839 (2006)
11. Ghazi, T., Huth, M.: An abstraction-based analysis of rule systems for active database management systems. Technical report, Kansas State University, April 1998. Technical report KSU-CIS-98-6, pp15
12. Ksystra, K., Triantafyllou, N., Stefaneas, P.: On verifying reactive rules using rewriting logic. In: Bikakis, A., Fodor, P., Roman, D. (eds.) RuleML 2014. LNCS, vol. 8620, pp. 67–81. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09870-8_5
13. Le, H.A., Truong, N.T.: Modeling and verifying DML triggers using Event-B. In: Selamat, A., Nguyen, N.T., Haron, H. (eds.) ACIIDS 2013, Part II. LNCS (LNAI), vol. 7803, pp. 539–548. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36543-0_55
14. Lee, S.-Y., Ling, T.-W.: Are your trigger rules correct? In: Proceedings of the 9th International Workshop on Database and Expert Systems Applications, DEXA 1998, pp. 837–842. IEEE Computer Society, Washington, D.C. (1998)
15. Lee, S.-Y., Ling, T.-W.: Verify updating trigger correctness. In: Bench-Capon, T.J.M., Soda, G., Tjoa, A.M. (eds.) DEXA 1999. LNCS, vol. 1677, pp. 382–391. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48309-8_36
16. Li, X., Medina, J., Chapa, S.: Applying petri nets in active database systems. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **37**(4), 482–493 (2007)
17. Manicka Chezian, R., Devi, T.: A new algorithm to detect the non-termination of triggers in active databases. Int. J. Adv. Netw. Appl. **3**(2), 1098–1104 (2011)
18. Zhang, J., Moyne, J., Tilbury, D.: Verification of ECA rule based management and control systems. In: IEEE International Conference on Automation Science and Engineering, CASE 2008, pp. 1–7, August 2008