International Workshop on the Use of Formal Methods in Future Communication Networks (UFMFCN 2015)

An Operational Semantics for UML 2 Sequence Diagrams Supported by Model Transformations.

Nabil Messaoudi\textsuperscript{a,b}, Allaoua Chaoui\textsuperscript{b}, Mohamed Bettaz\textsuperscript{c}

\textsuperscript{a}University Abbes Laghrour Khenchela, Algeria
\textsuperscript{b}MISC Laboratory, Department of Computer Science and its applications Faculty of NTIC University Constantine 2, Algeria
\textsuperscript{c}Ecole Superieure d’Informatique (ESI) Algiers, Algeria

Abstract

In this paper, we propose an operational semantics for UML2SD (Unified Modeling Language 2 Sequence Diagrams) to its equivalent Büchi automaton. The objective of this paper is twofold; first we provide UML2SD with Büchi automaton formal semantics, and second we bridge the gap between theoretic studies to practical studies by improving model transformations. The approach is based on Algebraic graph transformation and uses AGG (Attribut Graph Grammar) tool. The rules of the graph grammars specifying transformation of basic interactions and combined fragments are based on the proposed semantics. A scenario of ATM (Automatic Teller Machine) as case study will illustrate our approach.

Keywords: Sequence diagrams; Semantics; Büchi automaton; AGG

1. Introduction

Nowadays, model checking has been widely used in software engineering as a method to verify the models of systems. The most important aspect of reactive systems to be checked is the communication in form of interactions between the various entities in the system. Several graphical languages have been proposed to model interactions of computer systems. UML is a famous standard language widely used to describe system behavior in general and communications in a particular way. It offers a set of diagrams where the sequence diagrams are the most important ones for modeling communications and intended specially to capture the behavior of reactive systems that maintain an ongoing interaction with their environment\textsuperscript{2}. The extended version of UML2 makes significant changes to the sequence diagrams. In particular, the addition of combined fragments and an informal specification of operators like: seq, strict, alt, neg, and loop.

* Corresponding author. Tel.: +213-07-73-13-56-33 ; fax.: +213-032-72-51-60
E-mail address: messaoudi@misc-umc.org

1877-0509 © 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
Peer-review under responsibility of the Conference Program Chairs
doi:10.1016/j.procs.2015.07.253
As the professionals of computer systems design usually work on finite models in the form of finite state automata, such as Büchi automata, the construction of automata for interactions is a fundamental link between the specification and verification of a system by using graph transformations. Therefore, the definition of a formal semantics for interactions in the form of automata overcomes the deficiencies of the UML2 informal semantics and bridges a gap between modeling and verification.

Algebraic graph transformation has been recognized by several researchers as a tool to specify model transformations. It has a well-founded theory and associated tools to elaborate a graph transformation process. For specifying the transformation of UML2SD, it is necessary to understand the executions that are specified with precision. The informal semantics used in the UML specification is not sufficient to build automatic tools (in particular the construction of automata). For this reason various researchers have proposed sequence diagrams with formal semantics. Their abstract syntax is based on interaction, basic interaction and operators (seq, strict, alt, opt, par, neg, not, loop, restr). The operators are applied to interactions to describe interaction behaviors. In the authors define the allowed syntactic constructs for basic interactions defined as set of lifelines and event occurrence specifications to those lifelines. Since a basic interaction is also a subset of the live sequence chart, it can be applied to basic interactions to construct automata. The interactions describe finite state systems. For such systems, it is sufficient to use automata that accept only finite words. However, interactions in reactive systems can create potentially infinite words. This motivates the use of ω-automata such as Büchi automata to describe interactions in sequence diagrams.

To achieve this transformation, we propose a metamodel for UML2SD and another one for Büchi automaton. After that we have proposed a graph grammar performing the transformation UML2SD to an instance of the metamodel of Büchi automaton.

AGG is one of the standard graph transformation tools which implements the algebraic graph transformation approach; we will thus use this tool to perform this transformation. The rules for specifying transformation from basic interaction to automaton are based on the semantics of live sequence charts, while the rules for specifying transformation of combined fragments are based on the semantics of combined fragments.

The main steps of our approach can be summarized as follows: we first propose an abstract syntax for UML2SD. Given an abstract syntax of UML2SD, we transform each basic interaction into a Büchi automaton. Given Büchi automata as operands, we apply semantics of operators to generate a Büchi automaton. The rest of the paper is organized as follows: in section 2, we situate our work among related ones. Section 3 is devoted to the description of the syntax of UML2SD. Section 3 motivates the choice of our semantics. In section 4, we show how to transform UML2SD into Büchi automaton through model transformations.

2. Related Work

Many research works have been conducted to deal with the lack of precise constructs of UML. In the authors define a transformation from UML2SD to automata by using traditional algorithms (unwinding algorithms), which build an internal representation of the input file by using the JavaCC parser. Next, by the use of successive visits (Pattern Visitor) and by applying the semantics of operators, they obtain the corresponding automaton. Compared to, our approach avoids the manipulation of classical abstract syntaxes to implement semantics. We obtain abstract Büchi automaton directly by using transformations of graphs.

In the authors define a transformation from live sequence charts to automata. They use the unwinding algorithms to generate automata. In our approach, we apply only transformation rules to generate Büchi automata. In the authors specify a transformation from state machine to Petri nets by using attributed graph grammar. The implementation of transformation rules is realized with AGG. This work converges with our methodology in the implementation of transformation rules; it uses three layers to realize this transformation. In our approach, the management of combined fragments and the choice of automata for the model checker create many differences in implementation, such as: the use of operators and the generation of transition instances.

In the authors specify an algebraic transformation using the AGG tool for transforming sequence diagrams into state machines. The specification of transformation rules uses the concrete syntax of sequence diagrams, it highlights a new graphical operator with an aim of matching and transforming combined fragments. Compared to our approach, we don’t use a new operator to transform combined fragments; we only transform the basic interactions associated with combined fragments to automata. In the phase of implementation, we apply the semantics associated
to operators for transforming abstract Büchi automaton to concrete Büchi automaton. So, we are in the stage that precedes the call to the model checker.

Our main contribution consists in the use of algebraic graph transformation to perform the transformation of UML2SD to abstract Büchi automaton.

This paper is an upbeat to the limit encountered in our work\textsuperscript{15}, where the ATL language doesn’t support the application of multi-layer transformations but AGG enables the use of multi-layer transformations by defining the sequence of transformation rules. Compared to\textsuperscript{15}, this one illustrates the use of a clear operational semantics and allow us to validate the equivalence of semantics between the input model and the output one. We realize that a valid transformation ensures this semantics. However, the demonstration of equivalence is out scope of this paper.

3. Background

In this section, we briefly describe the informal semantics of sequence diagrams and recall some basic concepts of graph transformations

3.1. Sequence Diagram

Sequence Diagram is a two-dimensional graphical notation (horizontal and vertical). The horizontal dimension is used for representing the objects and the vertical dimension is used to represent the time. The objects are identified by lifelines; they represent the existence of the object at a particular time and these objects produce a set of events. The order of events along lifeline specifies the order in which these events will occur\textsuperscript{3}. An important concept used in\textsuperscript{3} is the occurrence specification. It specifies the basic semantics of an interaction. Its order along lifeline is the meaning of the interaction and it references the lifeline on which the occurrence specification appears. The most visible aspects of an interaction are messages between lifelines. The sequence of the messages is the spine for understanding the situations. A message links two distinct instances which can cause an operation to be invoked. Graphically, we use an horizontal arrow from the lifeline of one instance to the lifeline of another one. It specifies the sender and the receiver events occurrences. As specified in\textsuperscript{3}, UML2SD is not limited only to basic interactions but it may contain combined fragments which can be structured using interaction operators; the lifelines are connected to the combined fragments that cover them. The interaction operand contains the interaction fragments which are enclosed by this operand. Each operator is characterized by its own informal semantics specified in\textsuperscript{3}. Many operators have been adopted. We mention for example, alt (alternative behavior), par (parallel behavior), loop (repeated behavior) and negate which captures the forbidden behavior. Each operand has a guard attribute and includes a subset of lifelines and the combined fragment covers the operand. Elements not used like state invariants have been excluded.

3.2. Operational Semantics

Our semantics for sequence diagrams is defined in the form of Büchi automata by using model transformations. The semantics for basic interactions is defined by transformation rules showing how to create Büchi automata from scratch, while the semantics of combined fragments is defined by rules that delimit the beginning and the ending of combined fragments. The semantics of combined fragments enables us to delimit all operands in the form of Büchi automata which are related by operators such as seq, strict, alt, opt, neg and loop. We obtain a system which contains a set of operands where each operand corresponds to Büchi automaton delimited by its correspondent operator. We transform a sequence diagram to Büchi automaton which contain only positive behavior. The negative behavior is ignored by using a transformation rule which deletes the negate operator (neg) of the sequence diagram. To be able to develop the transformation rules, it is interesting to understand the semantics of interactions and the related automata. To achieve this goal, we use\textsuperscript{2,5} to specify transformation rules of UML2SD into Büchi automata and to show how to generate Büchi automaton from NDFA (Non Deterministic Finit Automaton). In the section below, we give the formal definition of the NDFA and the corresponding Büchi automaton. The formal definition of the operators on automata will also be illustrated.

Definition 1 (Finite Automaton): a finite automaton A over finite words is a 5-tuple $A = (\Sigma, S, T, S_0, F)$ where:

1. $\Sigma$ is the finite alphabet which contains all events send (message, $ob_{i, j}$), receive (message, $ob_{i, j}$).

\textbf{Definition 1 (Finite Automaton):} a finite automaton $A$ over finite words is a 5-tuple $A = (\Sigma, S, T, S_0, F)$ where:

1. $\Sigma$ is the finite alphabet which contains all events send (message, $ob_{i, j}$), receive (message, $ob_{i, j}$).
2. $S$ is the finite set of states.
3. $T \subseteq S_i \times \Sigma \times S_j$ is the set of transitions for $i \neq j$.
4. $S_0 \in S$ is the initial state.
5. $F \subseteq S$ is the set of final states.

The most suitable formal model for UML2SD over infinite words is $\omega$-automata. In particular, Büchi automata which are $\omega$-automata that require an accepting state to be visited infinitely often for every accepted input sequence. A simple way to ensure the transition from non deterministic finite automaton (N DFA) to Büchi automata: in order to avoid premature termination within a finite automaton, it should allow to loop forever in each accepting state labeled by $\Sigma$. A $\varepsilon$ self-loop to every state describes the possibility for stuttering between events. To formulate the Büchi automaton, we propose the definition 2, definition 3 and the definition 4.

**Definition 2 (Stutter event):** to allow that on object of UML2SD can stay infinitely long (standard of UML2SD), we use the event called stutter and denoted $\varepsilon$, this later has no effect on the system. This event will permit to stay infinitely long in the same state in order to perform another event send message or receive message, it will be denoted that an interaction $I$ applied to $\varepsilon$ is always the interaction $I$ i.e. $I \cup \varepsilon = I$.

**Definition 3 (Transition Label):** every transition of the automaton is annotated with an event $\tau$ which is a sending message or receiving message and a guard condition $\rho$ for an operator of a combined fragment. The event $\tau$ is the actual send or receive event encoding the communication. The guard condition $\rho$ is evaluated at runtime to determine if the transition can be fired. So, transitions labelled $\mu \in (\tau, \rho)$ represent the set of events in the interaction $I$. Transitions describe the differences between two states connected by these later.

**Definition 4 (Accepted States):** an interaction progress among life line and will terminate when the ending of lifeline is encountered. As an UML2SD contains $n$ lifelines, the Büchi automaton of automaton are $n$ accepted states labelled with stutter events.

**Definition 5 (From N DFA to Büchi Automaton):** our Büchi automaton $\beta$ for an interaction $I$ denoted $\beta(I)$ is a 5-tuple $(S_0, \Sigma = \cup \{e\}, S, \Psi = T \cup T', A_\varepsilon)$ where:
1. $S_0$ is the initial state.
2. $\Omega = \Sigma \cup \{e\}$ the finite alphabet which contains all events send (message, $ob_{ji}$, $ob_{jj}$), receive (message, $ob_{ji}, ob_{jj}$) and the stutter event $\varepsilon$.
3. $S$ is the finite set of States.
4. $T \subseteq S_i \times \Omega \times S_j$ is the set of transitions, for $i \neq j$.
5. $T' \subseteq S_i \times \{e\} \times S_j$ is the set of transitions for $i=j$.
6. $A_\varepsilon \subseteq S$ is the set of accepted states.

**Definition 6 (strict behavior):** the stutter event with more than one operand creates new automata by merging the states of the operand automata into a completely new set of states. It creates one initial state and merges the second initial state into another state. For two interactions $I_1$ and $I_2$, let $\beta(I_1)$ and $\beta(I_2)$ be corresponding automaton: $\beta(I_1) = (initial_1, \Omega_1, S_1, \Psi_1, A_1, \varepsilon)$ and $\beta(I_2) = (initial_2, \Omega_2, S_2, \Psi_2, A_2, \varepsilon)$. The strict transition $\beta(I_1) \beta(I_2) = (initial_1, \Omega_1 \cup \Omega_2 \cup \{e\}, S_1 \cup S_2, \Psi_1 \cup \Psi_2 \cup T_e, A_1 \cup A_2, \varepsilon)$ where $T_e$ is the transition which links the state of the last occurrence specification $S_j$ in the interaction $I_j$ to the initial state $initial_2$ of the interaction $I_2$. Formally denoted as follows: $T_e \rightarrow_{\varepsilon} initial_2$.

**Definition 7 (alt operator):** Given a combined fragment with $n$ operands. Each operand possess its deterministic fired condition denoted respectively $condition_1, condition_2, ... condition_n$. The alt($condition_1, I_1; condition_2, I_2; ...; condition_n, I_n$) = alt($\beta(I_1), T_1; \beta(I_2), T_2; ...; \beta(I_n), T_n$) where $T_n$ are transitions formally defined as follows: $T_{i=1,n}$: $S_{i=1,n} \rightarrow S_{j=1,n}$ where $S_r \in S$ which will be considered as the first state for the operand. So, $T_i$ links the first occurrence specification $S_i$ in the interaction $I_i$ for a Büchi automaton $\beta(I_i)$ and will be fired when the respective condition is evaluated to true.

**Definition 8 (loop operator):** as sequence diagrams are set of interactions represented with set of operands. We consider each area of combined fragments as an area of a new behavior. So, we define a begin state of combined fragment and an end state of combined fragment. As result, if our diagram contains $n$ combined fragments, it covers $n$ begin states and $n$ end states. We only take into consideration the behavior repletion for a deterministic condition: $Loop(condition_i, I_i) \equiv loop (T_{begin, \beta(I_i)}, T_{exit}, T_e)$ where:
Definition 9 (Negative behaviours): to implement negative behaviours (forbidden behaviour); we just remove the neg operator and its content\(^{13}\)). For a Büchi automaton, the application of the operator neg has no behaviour and will be interpreted as: \(\text{neg}(\beta(I)) \equiv \phi\).

4. Our approach

To illustrate the use of Operational semantics cited above through model transformations, (1) we propose two metamodels one for UML2SD and another for Büchi automaton,(2) a graph grammar that performs the transformation.

4.1. Büchi Automaton metamodel

Fig. 1 shows our metamodel for Büchi automaton. The automaton possesses an initial state identified by the attribute \(\text{IsInitial}\) having a true value, a set of available states and a set of transitions. The attribute \(\text{Kind}\) associates to a class transition which is chosen to identify the nature of a transition. A single name is affected for each state to distinguish between states. The purpose of using the class \(\text{BüchiAutomaton}\) is to capture each interaction into an abstract Büchi automaton.

Fig. 1. Büchi Automaton Metamodel.

4.2. UML2SD metamodel

Our metamodel for UML 2 sequence diagrams is depicted in Fig.2. A sequence diagram is not of other than a set of interactions, where each interaction is covered by a set of lifelines. The occurrences are ordered top-down according to each lifeline. A message consists of send event and receive event, which are linked to two different occurrence specification. A combined fragment consists of a set of operands. Each operand is considered as an interaction. A guard relation indicates the interaction constraint of the operand. Each combined fragment has a start occurrence specification and end occurrence specification. The attribute \(\text{Kind}\) of Combined fragment indicates the type of operator.
4.3. Productions for model transformations

Figures of transformation rules have not been illustrated due to the limited number of pages.

**Rule 1: InitInteractionToAutomaton.** This rule transforms each interaction to an instance of abstract Büchi automaton having the same name as in the interaction. The NAC guarantee that each interaction is transformed only on time.

**Rule 2: InitOccurrenceSpecificationToState.** This rule transforms each occurrence specification to an instance of state whose name is given by the formula $S^{+”-”+}Integer.toString(n)$.

**Rule 3: InitMessageToState.** This rule transforms each message to an instance of state whose name is given by the name of message.

**Rule 4: Association between the source and the target.** This transformation rule creates the instance Transition to link the 4:state and 5:state. We link the instance OccurrenceSpecification to its corresponding state.

**Rule 5: Assigns a type send and receive message.** To obtain all participants in communication (parameters of transitions send and receive), we use the relation Covered to determine what are the concerned lifelines by this communication. In LHS, we have no name of transitions when we apply this one, we obtain the name of transitions.

**Rule 6: Relation between two consecutive combined fragments.** This rule links two consecutive combined fragments by using the Stutter event. If occurrence specification is of kind EndCfg and the next occurrence specification is of kind StartCfg, we create the relation Stutter to link the consecutive occurrence specification.

**Rule 7: Generation of Stutter transition between states.** This rule links each state whose accepted attribute is true and the consecutive state whose attribute initial state is true. Two states are consecutive if their corresponding occurrence specification are consecutive; this condition is given by the formula $(y-x = 1)$ where y and x represent the order of the consecutive occurrences specifications.

**Rule 8: Linking States for Büchi automaton.** These rules link each state to their own abstract Büchi automaton.

**Rule 9: Generation of InSameState for states.** This rule allow us to avoid the premature terminaison of interaction, we transform NDFA to Büchi automaton.

5. Case Study

In order to illustrate our approach, we have used the last version of AGG. It is implemented in JAVA and offers clear and simple concepts to develop transformations, and ensures the use of layer’s transformation which is the core of our approach. The case study in this section is a possible sequence diagrams of the uses case withdraw money where nominal scenario is detailed below.

The sequence diagram of Fig.3 describes the nominal scenario of the use case withdraw money which allows a bearer of credit card to withdraw money if his weekly credit allows that. It defines a principal actor who is the bearer of card, a secondary actor which is the automatic system and the instance ATM.
In the following, we illustrate step by step the generation of the correspondent abstract Büchi automaton. The first one is to insert manually a possible abstract syntax of Figure 3 according to the metamodel which we have defined above. Rule 1 transforms the interaction named sd1 to BuchiAutomaton named sd1. The Rule 2 transforms each OccurrenceSpecification to the corresponding state. The Rule 3 transforms each message to the corresponding state. We have applied the rule 4 to generate transitions between states(transition snd(x,y,m1) and the transition rcv(x,y,m1)). After that, we have applied the rule 8 to generate the relation states for linking each state to its own abstract Büchi automaton. Ultimately, we have applied the Rule 9 to generate the relation InSameState applied to all states of the automaton (see Fig.4).
6. Conclusions and future Work

In this paper we have proposed an operational semantics to transform an abstract UML2SD to its equivalent Büchi automaton. The objective of this approach was to provide UML2SD sequence diagrams with Büchi automaton formal semantics. As a result we can use a model checker to verify some properties. Semantics were supported on Algebraic graph transformation and used AGG tool. So, we have formalized a simple way to capture the operands inside the combined fragments. We have transformed sequence diagrams to a manipulation of operators and operands by analogy with arithmetic expressions, where the operands are Büchi automata. In a future work we plan to extend our approach by transforming the obtained Büchi automaton to concrete PROMELA code.

References