Declarative Approach to Model Checking for Context-Aware Applications

Ammar Alsaig(✉), Vangalur Alagar, and Nematollaah Shiri

Concordia University, Montreal, QC, Canada
{A_alsaig,alagar,shiri}@encs.concordia.ca

Abstract. Systems need to be formally verified to ensure that their claimed properties hold at all times of system operation. Deterministic Finite State Machines (FSM) are widely used as model checkers to verify system properties. However, for context-aware systems that have regular inputs and contextual inputs, FSM models become more complex and less intuitive, and do not precisely represent the system behavior. In this paper we use simple examples to introduce the declarative reasoning framework *Contelog* , a theoretically and practically well grounded work in progress, as a complementary approach that can be used to represent, reason, verify data-centric and contextual properties of context-aware systems.

Keywords: Formal verification · Context-aware modeling · Model checking · Context-based knowledge base systems

1 Background

As technologies progress, their dependence on pervasive, ubiquitous, and Knowledge base system features is increasing in order to meet application demands in different fields. In many of these application domains safety-criticality plays a crucial role in decision making. Some examples are (1) rule-based medical diagnosis systems [8,10], (2) rule-based access control [7] systems to enforce security/privacy, and (3) rule-based expert systems [6] for prediction. Due to this criticality, formal verification is necessary on the modeled system to ensure that safety, security, and privacy properties hold at all times throughout the operation of the system. One method of formal verification is through model checking [9]. Many model checkers that are available for hardware/software checking are also being used to model check context-aware system properties. However, it is pointed out [13] that traditional checking models are insufficient, inefficient, and non-intuitive for verifying properties of context-aware systems. The dynamism, and the rapid change of the behaviour of context-aware systems make them prone to adaptation-faults and unexpected behaviours [12]. This makes context-aware verification a challenging process. It is in this context that we propose *Contelog* reasoner for model checking context-aware systems.

P. C. Vinh and A. Rakib (Eds.): ICCASA 2019/ICTCC 2019, LNICST 298, pp. 3–10, 2019.
https://doi.org/10.1007/978-3-030-34365-1_1

Example in Fig. 1 is to motivate where and how contexts arise and how they can be brought into FSM modeling. The illustrated scenario is that of entry to an emergency room in a hospital. Based on access control model, both patients and doctors are authenticated to enter the room. Whereas a doctor is allowed to enter using the authorized barcode at any time, patients and other staff are allowed to access the room only at specific time periods using their authorized barcodes. Because there might exist many emergency rooms, a person may require to use the same or different bar code to enter many rooms subject to time constraints for entry, the pairs "⟨ times, emergency room locations ⟩" become the set of contexts at which barcode authentication should be certified for valid entry. Because any combination logic can be represented as a deterministic Finite State Machine (FSM) [16], the above example can be represented as an FSM as shown in Fig. 2. However, the FSM representation in Fig. 2B in complicated scenarios becomes complicated and non-intuitive. This is because it is less descriptive as it reduces most of the details to context. Although there is a rich volume of literature [4,14] dedicated to increasing the expressiveness of FSMs, their representation, understandability, and verification become more challenging when a variety of contexts, that are multi-dimensional objects, need to be represented. As observed in [11,12], standard FSM can have states, input, and outputs, but adding context can only be in terms of inputs which does not represent the actual conceptual relevance of context to the system. Thus, we are motivated to introduce our framework *Contelog* to model context-aware systems. *Contelog* is a Datalog program [1] in which context is integrated as a first class citizen in order to enable contextual reasoning. The declarative semantics of *Contelog* can prove theoretically that certain property holds for the entire world of inputs and contexts embedded in it.

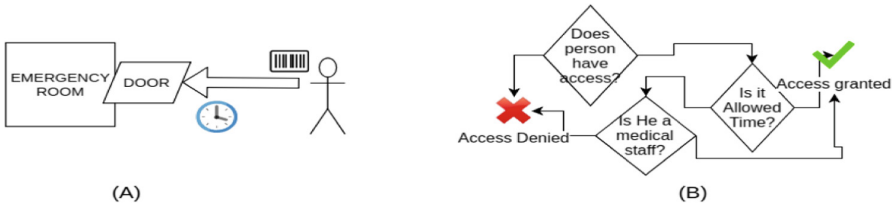

Fig. 1. Illustration (A) gives general overview of the example, illustration (B) gives the scenario in if-statement diagram

1.1 Outline

In Sect. 2 we briefly introduce the theoretical foundation of *Contelog* . In Sect. 3 we discuss the reasoning tool and comment on the Book of Examples [2] that we have implemented using the reasoning tool. In Sect. 4 we describe *Contelog* programs for the Microwave Example, the most often cited first example in model checking, and the access control example Fig. 2. We conclude the paper in Sect. 5 with a brief description of our work in progress.

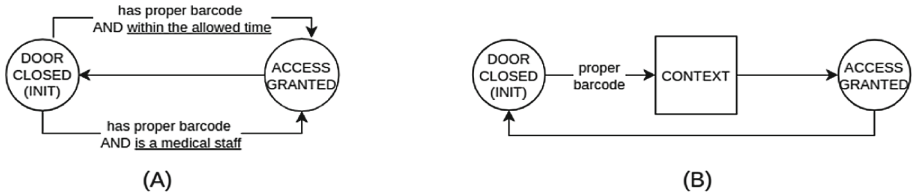

Fig. 2. Illustration (A) represents Fig. 1 in FSM (B) provides an equivalent diagram using context

2 Formalization of Context, and *Contelog*

In this section we briefly summarize our results on context formalization [3] and the work completed so far on *Contelog* creation. Context is a multi-disciplinary concept that has diverse conceptualizations. However, it has been commonly perceived as the "settings of a system" and "a surrounding environment" [5,15]. Our context theory is conceptualized in three layers, in order to allow maximum flexibility for system designers to choose meaningful contexts and at the same time provide sufficient formalism for building a context calculus on which contexts can be manipulated at system level. The layers are respectively Context Schema(CS) layer, Typed Context Schema(TCS) layer, and Context Instances(CI) layer. Contexts are viewed as multi-dimensional objects where a dimension represent a type of the settings being described, and attributes associated with each dimension give the detailed description for a particular dimension.

Definition 1. *Let D be a finite, non-empty set of dimensions, A be a set of attributes, and V^T is a set of values of type T. A context Schema $\mathscr{C}$ is a set of pairs defined as follows:*

$$CS = \mathscr{C} = \{< d, A_d > \mid d \in D \wedge A_d \subseteq A\}$$

$$TCS = \mathscr{C}^T = \{< d, A_d > \mid d \in D \wedge A_d \subseteq A^T\}$$

$$CI = I_i(\mathscr{C}^T) = \{< d, V_d > \mid d \in D \wedge V_d \subseteq V^T\} \quad \blacksquare$$

As described in [3], context operations are Join ($\oplus$) and Meet ($\odot$). The definitions of operations on context schemas lead to a closed lattice structure, thus making our context calculus a complete universe. Datalog also considers closed world under Herbrand structures. Consequently, when we integrate it with the declarative semantics of Datalog we achieve "closed world assumption" for context reasoning. Thus, *Contelog* is a logic-based framework that uses "context" as a first class citizen, extends Datalog's syntax and semantics to reason with contextual knowledge in a declarative fashion. *Contelog* syntax is formally presented as follows:

Definition 2. *A* Contelog *program P ($\mathscr{C}$-program) is a four-tuple $\langle T, S_{\mathscr{C}^*}, F, R\rangle$ whose components are defined as follows:*

1. *T is the set of truth values $\{True, False\}$*
2. *$S_{\mathscr{C}^*}$ is a set of instantiated contexts partially ordered by $\sqsubseteq$. We assume that $< S_{\mathscr{C}^*}, \sqsubseteq, \oplus, \odot >$ is a complete lattice. The least upper bound is denoted by $C_\top$, and the greatest lower bound by $C_\bot$.*
3. *F is a finite set of annotated ground atoms, each of which is in the form $q@c$, where q is a ground atom, and c is a context name in $S_{\mathscr{C}^*}$.*
4. *R is a finite set of rules, each of which is in the form:*

$$r: \quad H :\text{-}\ B_1, B_2, \ldots, B_n$$

where $H, B_1, \ldots, B_n$ are annotated atoms. Standard datalog predicates may be annotated with context C where C is a context variable, e.g., $q(\bar{X})@C$.

As in datalog, we use uppercase letters for variables and lower case letters for constants in the universe. As is customary in database and logic programming frameworks, we restrict the semantics of ($\mathscr{C}$-*program*) to Herbrand structures. We have developed the declarative semantics for ($\mathscr{C}$-*program*) P to be the least model of P, defined by the intersection of all the Herbrand models of P. We also developed a bottom-up, fixpoint semantics of ($\mathscr{C}$-*program*) and show that the fixpoint model of any ($\mathscr{C}$-*program*) P coincides with the least model of P. We have built a running prototype of the Contelog framework which computes the fixpoint semantics of ($\mathscr{C}$-*program*) . The prototype system has been tested with a number of examples and is available for evaluation at the link [2].

3 *Contelog* System

Contelog system is a tool that implements both context calculus and *Contelog* reasoner. This prototype tool is a playground for *Contelog* programs to be able to implement simple and small-sized *Contelog* programs. The system can be accessed online through this link [2]. The system allows input in two modes. These are respectively context and code input modes. The rationale is to allow context calculus to be tried independently from executing a *Contelog* program. In context input mode, context representation, as formalized by us, should be used. The user interface guides the user to input syntactically correct input with correct typed values. In code input mode the syntax of *Contelog* is used to guide the user to create the program. The syntax checker will ensure that only those contexts that have already been constructed are used in the program rules. This is to avoid any inconsistency in reasoning. The reasoning is currently operating using the naive reasoning method. The complexity is polynomial, as for Datalog programs, yet it is an exhaustive approach and not efficient for large-sized applications. The current running version is just a proof of concept, just to demonstrate the different kinds of examples that we have implemented, and give users a forum to use the system and give us feedback.

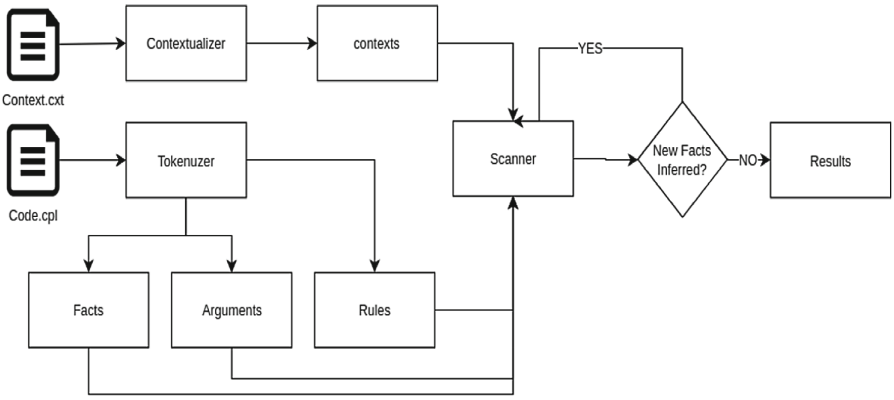

Fig. 3. *Contelog* structure

4 Reasoning with *Contelog* : Access Control and Microwave Examples

We claim that a FSM for context-aware systems can be represented using *Contelog* in three steps. In Step 1, each non-contextual input/edge is represented as a ground predicate (fact) in *Contelog* . For instance, the barcode input in Fig. 1 is represented as $input(barcode)$ in *Contelog* . In Step 2, each contextual input or any input that is dynamic in nature is represented as context using our notation. For instance, staff context is represented as $C_{staff} = \{time\_constraint : [none], type : [staff]\}$, where $time\_constraint$ and $type$ are dimensions. Each state is represented as a rule, while the edge (condition) to move from one state to another is represented in the rule body. For instance, to represent the event of moving from "init state" to "open state" we write $state(\text{open-door})@C : \text{-}input(\text{barcode}), type(\text{staff})@C$. Finally, the initial state is given as a ground fact. Following this transformation rule we get the two contexts $\mathscr{C}_s = \{time : [no], type : [staff]\}$, and patient context $\mathscr{C}_p = \{time : [yes], type : [patient]\}$ for medical staff and patient categories. Inputs, states, and events are represented in terms of facts and rules. The resulting *Contelog* program and its execution in our *Contelog* system are shown below:

Contelog representation for the motivating example

```
input(1,barcode)@cs.
input(2,barcode)@cp.
state(X,open)@C:-input(X,Z)@C,time(no)@C,type(staff)@C.
state(X,open)@C:-input(X,Z)@C,time(yes)@C.
state(X,closed)@C:-state(X,open)@C.
***** RESULTS *****
{input(1,barcode)@cs, input(2,barcode)@cp, **state(1,open)@cs,
**state(2,open)@cp, **state(1,closed)@cs, **state(2,closed)@cp.}
```

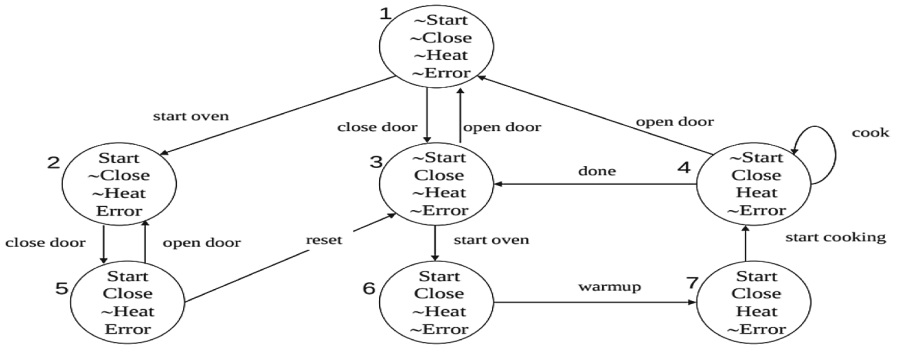

Fig. 4. *Contelog* structure

In the commonly known Microwave example contexts are used to model "from-to relationship" between states in the FSM shown in Fig. 4. Inputs are regular facts, while first state is the initial state. The program recursively fires the rules and generates all output, given the correct set of input. If system is not given the input "startoven" it will not generate any of the other answers as they are all chained together through contexts. The *Contelog* representation for Microwave example is shown below. In order to verify a feature or a property such as "heat does not start with door opened", a query like $features(heat)@c1{*}c2{*}c3$ can be used. The idea is to identify all contexts where door is open and use it in the query. Basically, can heat feature be present in all three contexts at the same time? If the answer to the query is an empty set (no answer) then heat is not present in all the three contexts at the same time, which means that heat while door is closed is satisfied.

Contelog representation for the Microwave example

```
Note: context 4, 41, 42 are edges coming out of the same state.
## CONTEXT ##
c1={'to':['startoven','c2'],
    'features':['~start','~close','~heat','~error']}
c2={'to':['closedoor','c5']},
    'features':['start','~close','~heat','error']}
c5={'to':['opendoor','c2']},
    'features':['start','close','~heat','error']}
c51={'from':['c5'],'to':['reset','c3']},
     'features':['start','close','~heat','error']}
c3={'to':['startoven','c6']},
    'features':['~start','close','~heat','~error']}
c31={'from':['c3'],'to':['opendoor','c1']},
     'features':['~start','close','~heat','~error']}
c6={'to':['warmup','c7']},
    'features':['start','close','~heat','~error']}
c7={'to':['startcooking','c4']},
    'features':['start','close','heat','~error']}
c4={'to':['cook','c4']},
    'features':['~start','close','heat','~error']}
c41={'from':['c4'],'to':['done','c3']},
     'features':['~start','close','heat','~error']}
c42={'from':['c4'],'to':['opendoor','c1']},
     'features':['~start','close','heat','~error']}
```

```
## CODE ##
input(startoven).
input(closedoor).
input(opendoor).
input(reset).
input(warmup).
input(startcooking).
input(done).
input(cook).
state(1)@c1.
state(X)@C:−state(X)@W,input(M),to(M,C)@W.
state(X)@C:−state(X)@W,from(W)@C.
### RESULTS ###
{ input(startoven), input(closedoor), input(opendoor), input(reset),
input(warmup),input(startcooking),input(done),input(cook),state(1)@c1,
**state(1)@c2,**state(1)@c5,**state(1)@c51,**state(1)@c3,**state(1)@c6,
**state(1)@c31,**state(1)@c7,**state(1)@c4,**state(1)@c41,**state(1)@c42 }
```

5 Conclusion

In this paper we have proposed *Contelog* as an approach to model check context-aware system properties through examples. Our study on *Contelog* was originally motivated from the need to provide a formal framework for representing and reasoning about contextual knowledge. With that goal we have completed a context formalism, *Contelog* semantics, and constructed a prototype implementation as a proof concept of what we have achieved. We believe that our approach needs to be fine-tuned with query optimization techniques to deal with the reasoning of large context-aware systems. We are undertaking a deeper study of model checking methods, especially for protocol and contract specifications, investigate how contextual reasoning may be necessary in ubiquitous applications, and find ways to improve our current approach to handle such larger real life practical problems.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases: the Logical Level. Addison-Wesley, Boston (1995)
2. Alsaig, A.: Book of examples: a prototype environment for reasoning with contexts (2017). http://www.contelog.com
3. Alsaig, A., Alagar, V., Shiri, N.: Formal context representation and calculus for context-aware computing. In: Cong Vinh, P., Alagar, V. (eds.) ICCASA/ICTCC -2018. LNICST, vol. 266, pp. 3–13. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-06152-4_1
4. Bresolin, D., El-Fakih, K., Villa, T., Yevtushenko, N.: Deterministic timed finite state machines: equivalence checking and expressive power. arXiv preprint arXiv:1408.5967 (2014)
5. Brézillon, P.: Context in problem solving: a survey. Knowl. Eng. Rev. **14**(1), 47–80 (1999)
6. Buchanan, B.G., Shortliffe, E.H., et al.: Rule-Based Expert Systems, vol. 3. Addison Wesley, Reading (1984)

7. Carminati, B., Ferrari, E., Perego, A.: Rule-based access control for social networks. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006. LNCS, vol. 4278, pp. 1734–1744. Springer, Heidelberg (2006). https://doi.org/10.1007/11915072_80
8. Clancey, W.J.: The epistemology of a rule-based expert system–a framework for explanation. Artif. Intell. **20**(3), 215–251 (1983)
9. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R.: Handbook of Model Checking. Springer, Heidelberg (2018)
10. Lamperti, G., Zanella, M.: Rule-Based Diagnosis, pp. 193–233. Springer, Dordrecht (2003)
11. Le, H.A.: Formal modeling and verification of context-aware systems using event-b. EAI Endorsed Trans. Context-Aware Syst. Appl. **1**, e4 (2014). https://doi.org/10.4108/casa.1.2.e4
12. Liu, Y., Xu, C., Cheung, S.: Afchecker: effective model checking for context-aware adaptive applications. J. Syst. Soft. **86**(3), 854–867 (2013)
13. Schmidtke, H.R., Woo, W.: Towards ontology-based formal verification methods for context aware systems. In: Tokuda, H., Beigl, M., Friday, A., Brush, A.J.B., Tobe, Y. (eds.) Pervasive 2009. LNCS, vol. 5538, pp. 309–326. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01516-8_21
14. Skelin, M., Wognsen, E.R., Olesen, M.C., Hansen, R.R., Larsen, K.G.: Model checking of finite-state machine-based scenario-aware dataflow using timed automata. In: 10th IEEE International Symposium on Industrial Embedded Systems (SIES), pp. 1–10. IEEE (2015)
15. Strang, T., Linnhoff-Popien, C.: A context modeling survey. In: Workshop Proceedings (2004)
16. Wagner, F., Schmuki, R., Wagner, T., Wolstenholme, P.: Modeling Software with Finite State Machines: A Practical Approach. Auerbach Publications (2006)