Online Verification through Model Checking of Medical Critical Intelligent Systems

Joao Martins, Raul Barbosa, Nuno Lourenco, Jacques Robin, Henrique Madeira

Online Verification through Model Checking of Medical Critical Intelligent Systems

João Martins
Centre for Informatics and Systems
University of Coimbra
Coimbra, Portugal
jpdmartins@student.dei.uc.pt

Raul Barbosa
Centre for Informatics and Systems
University of Coimbra
Coimbra, Portugal
rbarbosa@dei.uc.pt

Nuno Lourenço
Centre for Informatics and Systems
University of Coimbra
Coimbra, Portugal
naml@dei.uc.pt

Jacques Robin
Centre for Research in Informatics
University of Paris 1 Panthéon-Sorbonne
Paris, France
jacques.robin@univ-paris1.fr

Henrique Madeira
Centre for Informatics and Systems
University of Coimbra
Coimbra, Portugal
henrique@dei.uc.pt

***Abstract*—Software systems based on Artificial Intelligence (AI) and Machine Learning (ML) are being widely adopted in various scenarios, from online shopping to medical applications. When developing these systems, one needs to take into account that they should be verifiable to make sure that they are in accordance with their requirements. In this work we propose a framework to perform online verification of ML models, through the use of model checking. In order to validate the proposal, we apply it to the medical domain to help qualify medical risk. The results reveal that we can efficiently use the framework to determine if a patient is close to the multidimensional decision boundary of a risk score model. This is particularly relevant since patients in these circumstances are the ones more likely to be misclassified. As such, our framework can be used to help medical teams make better informed decisions.**

***Index Terms*—critical systems, intelligent systems, verification, model checking, medical risk scores**

I. Introduction

Software systems are increasingly relying on Artificial Intelligence (AI) in general and Machine Learning (ML) in particular, in some cases applied to critical domains. Examples include the health and transportation domains, with applications ranging from diagnosis tools [14] to self-driving vehicles [9]. In such domains, verification, certification and explanation are fundamental.

One of the crucial steps in the development process of critical systems is *verification*, with the goal of assuring that the design and implementation of a system fulfill its requirements. Formal methods are a complementary approach to software engineering methodologies and development processes, suitable for rigorously describing and reasoning about complex systems [16]. Based on formal logic and mathematical notions, techniques such as formal proofs [15] and model checking [2] can be used to verify properties of critical systems.

A key challenge of AI and ML components is that often one cannot extract a specification directly from the machine learned models, nor is their training performed according to a verifiable specification. Some research efforts focus on logic-based algorithms that learn reasoning models from data, since their representation is suitable for extracting a specification [12]. Provided a specification, we are therefore able to apply formal methods to perform verification.

Along the several existing approaches that include perceptron techniques, instance based learning, support vector machines, probabilistic and logic-based models, the last two are among the favored choices in the medical field. Decision trees and sets of rules, which are examples of logic-based models, are preferred over other representations as these are regarded as explainable and interpretable [8]. Models for computing *risk scores*, used in medical risk assessment, are a good example and an interesting case-study, given their acceptance and validation within the medical community [1].

This paper addresses the usage of model checking, at runtime, to qualify medical risk scores with information concerning the proximity of a patient to the multidimensional decision boundaries. The proposed framework provides the medical staff with an evaluation of the risk score, as the classic approach, along with a binary indicator of proximity to a decision boundary, based on specific medical input features and ranges. In other words, our frameworks explicitly identifies whether a patient's risk assessment could be different upon small changes to some of the input risk factors variables. This feature increases trust on the calculated risk scores, because the framework brings to the user's attention the cases in which the categorical risk assessment should be questioned.

Next, Section II addresses relevant existing contributions within the scope of our work, Section III provides a detailed explanation of the proposed framework, Section IV and V lay the experimentation, results and discussion done to the framework and Section VI presents the main conclusions.

This is a pre-print of an article published at DSML 2020, © IEEE.

II. Related Work

In this section we present relevant work already conducted in medical software certification, risk scores, verification strategies of intelligent systems and model checking. All these research fields lay bed to our proposed framework.

A. Medical software

Critical medical software must follow safety standards concerning lifecycle management (IEC 62304), risk assessment (ISO 14971) and usability (IEC 62366). However, those standards are based on assumptions that are typically not verified by AI. One of such assumptions is that software is implemented manually by human programmers, specifying at design-time the whole control and data flows of the software. Another one is that this implementation can be tested extensively by running test sets also manually programmed and that verify properties of these control and data flows.

An AI component, in contrast, is often implemented by a persistent declarative knowledge base, that is specific to the medical software to be developed, but it is interpreted by a generic inference that is completely independent of the medical software at hand. It is the run-time interaction between (a) the volatile input data given to the AI to reason about, (b) its declarative persistent knowledge base and (c) its application-independent inference engine that defines control and data flow. Also, the inference engine often performs heuristic and/or non-deterministic approximate search of a very large combinatorial space. This makes extensive design-time and manual writing of tests for such AI unpractical.

This difficulty is made worse by the fact that nowadays part if not all the knowledge base is acquired from data by machine learning instead of manually declared. This learning process includes steps of data selection, transformation, choice of learning algorithms, parameter tuning of these algorithms, learning hypothesis, search space *a priori* pruning, all of which introduce biases that can completely change the resulting knowledge base and hence the reasoning of the AI component. All these questions are only starting to be explored by the ISO/IEC task force JTC-1/SC-42.

B. Medical automated assessment of admitted patients (Risk scores)

Medical Condition Risk Scores (MCRS) were introduced in medical emergencies as tools to help prioritise and differentiate patients treatment, providing the needed balance between saving people's lives while saving on health costs through application of the right treatment.

The MCRS are based on large scale longitudinal medical research studies. They are, however, easy to implement as decision-support software since they generally apply simple weighted combinations of indicators such as blood pressure or known history of a condition. The output is the sum of the weighted values which falls under a risk category.

The various limitations of state-of-the-art MCRS were studied together with attempts to overcome them in two ways [13]: (a) composing several of them or (b) selecting the best of several for a given target population which phenotype may differ significantly from the cohorts used in the studies from which each of the considered MCRS was derived.

C. Verification strategies for intelligent systems

One might achieve verification through explanation. That is why [11] identifies logic-based models as interpretable and therefore verifiable through inspection. Model complexity is an attribute that puts not only manual inspection in check, but also challenges both manual inspection and global automated verification. [10] is an example of the efforts put into making a complex black-box model locally explainable. Our work falls within this trend, applying a formal method to locally check the vicinity of the risk factors space for a given patient.

D. Model checking

Model Checking (MC) is a formal method used to verify hardware and software systems [4]. It consists in building a formal model of what the system should do and then automatically verify properties of the model by state-space search techniques detecting the reachability of an error state.

MC has historical relevance in critical systems development typically following a waterfall process, meaning that it was a verification step before implementation. Nowadays, with new development processes, one can see source code as a specification itself [4]. The argument is that code is a static representation that has yet to be compiled, converted into binary and executed, so it is actually valid to see it as a requirements specification.

Model properties are often specified in a temporal logic language [3], with primitives like *eventually* and *always*, and can be used to avoid for instance deadlocks, contradictions or confirm correctness of the system output.

Many model checkers are composed of two main components [17]: (a) an executor that, for a model and entry-states, outputs reachable states and (b) a verifier that, for a given specification and state along the state exploration, checks if the specification is satisfied, returning the state and explored path as a counter-example when the specification fails.

III. Methodology

We propose a framework for verifying the evaluation made by an intelligent MCRS. It determines whether a risk assessment falls within a gray area of the decision space. As shown in Fig. 1, our approach feeds a model checker with both the patient data passed as input to the MCRS and the risk assessment output by the MCRS. This output is used as an assertion property to be verified by the model checker, which will search in the vicinity of the patient input risk factors space for different assessment outputs. If the model checker does not identify counter examples to the baseline assertion, the framework outputs a confident classification indication.

To implement our approach we started by programming the MCRS in Python for fast prototyping. We then manually translated the logic of the Python program together with assumptions and search ranges into a semantically equivalent

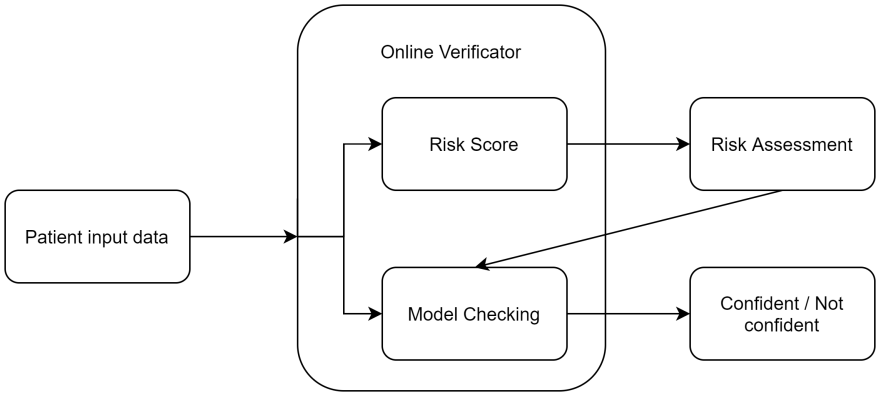

Fig. 1: High level overview of the proposed framework

Promela modeling language accepted as input by the popular model checker SPIN. We are then able to perform online verification through the model checker, for each new patient data, completing the framework pipeline. We explain each of the steps in detail in the following subsections.

A. Implementation

Listing 1 shows the implementation of a MCRS assessment function, considering a single risk factor. We can add further risk factors, following a similar approach. The risk assessment rules are implemented by conditional expressions and statements. *Age* is a common risk factor among risk scores and often includes several ranges of discrimination of the risk, so it is a good example that we shall use across the explanation of our approach. For a given patient, if the accrued risk sum falls above a pre-determined threshold, it outputs that the patient is in risk.

Listing 1: Implementation of a risk score in Python

```
def grace(patient_data, risk_threshold):

    risk=0

    age = patient_data['Age']
    if age >= 40 and age <= 49:
        risk += 15
    elif age >= 50 and age <= 59:
        risk += 29
    #Other age ranges...
    elif age >= 90:
        risk += 80

    #Other risk factors...

    if risk >= risk_threshold:
        patient_in_risk=True
    else:
        patient_in_risk=False

    return patient_in_risk
```

B. Translation

At this stage we have an executable MCRS assessment function. We now need to translate the code onto a specification language.

A few considerations must be taken into account for this step. Since Python has no assigned types, one needs to decide the necessary allocation space for the variables used in the code. Usually, type *int* is suitable for the majority of the risk factors, since it can both store categorical and numerical variables. Also, conditional statements are blocking-state in Promela, so one needs to add an *else* dummy state for each conditional statement in the specification model.

Listing 2 presents the specification for the implemented risk score in the previous subsection. Besides the straight forward translation, where we just added the assigned types and *else* states, the specification considers three extra elements: (a) *AGE* and *RISK* are constants that specify the input parameters concerning the patient risk assessment, (b) the *assert* statement set the property we want to verify and (c) a *select* command specifies the search space for the risk factor.

Listing 2: Specification of a risk score using Promela

```
#define AGE
#define RISK

active proctype Grace() {
    int age, risk;
    bool patient_in_risk;

    select(age : (AGE-3) .. (AGE+3) );
    risk=0;

    if
    :: age >= 40 && age <= 49 -> risk = risk +
        15
    :: age >= 50 && age <= 59 -> risk = risk +
        29
    //other age ranges...
    :: age >= 90 -> risk = risk + 80
    :: else -> skip
    fi;

    //other risk factors...

    if
    :: risk >= 145 -> patient_in_risk = true
    :: else -> patient_in_risk = false
    fi;

    assert(patient_in_risk == RISK)
}
```

C. Search space parameters

One needs to specify the bounds on which each input variable (risk factor) is going to be verified for the assumption of falling within the same risk assessment. As in Listing 2, by using constants to specify risk factors like *AGE*, we can limit the verification space through the *select* command, creating an upper and lower interval around the patient risk factor.

An important consideration is related to which risk factors we want to vary in our verification process. We do not perform ranging for the risk factors that are categorical since different values for those features represent patients that are too far apart, considering the notion of vicinity of medical risk factors search space.

D. Online verification

Each time a risk assessment is performed, one uses the implemented MCRS assessment function to obtain a classification. We then provide the model checking tool with the specification, previously translated, which is set according the input data of the patient and the risk assessment output. We use Spin as our model checking tool. Through the execution of the tool we check for counter examples of the expected risk assessment output within the risk factors search space bounded by the range parameters. If no counter example is found, we provide the user with an indication of confidence on the risk assessment.

IV. Experimental Evaluation

In this section we present relevant experimentation of our framework. We focused on implementing the GRACE risk score, translating it into a specification and applying the framework to a pool of patients, extracting relevant metrics.

A. GRACE

Global Registry of Acute Coronary Events (GRACE) is an international database that allowed for the supervised learning of a risk score model [6], [7], based on eight risk factors listed in Fig. 2. GRACE risk model was built for short term assessment, based on events of death or myocardial infarction (heart attack) on patients with coronary artery disease. GRACE risk score is widely accepted and was subject to many validation studies, identifying strengths and weaknesses [1], [5].

The novelty of our work arises from locally identifying regions where the risk assessment may vary for small patient risk factors changes.

B. Data

A collaboration with a medical team which applies the GRACE risk score allowed us to gather medical records from patients data and test and validate our framework.

The dataset used is composed of 460 patients admitted at Santa Cruz Hospital (Oeiras, Portugal) between March 1999 and July 2001, with the specific condition of acute coronary syndrome with non-ST segment elevation - ACS-NSTEMI [13].

TABLE I: Range variation parameters used in the GRACE model specification

Risk factor	**Range variation**
Age	+/- 4 (years)
Heart rate	+/- 5 (beats/min)
Systolic blood pressure	+/- 3 (mm Hg)
Creatinine	+/- 0.1 (mg/dL)

Within the dataset we have a positive event rate of 7.2%, comprising 33 events of death or myocardial infarction after 30 days of admission. We directly map those events with our target class, the risk indication.

C. Range parameters

Table I details the considered ranges for each of the varied risk factors. For a given patient (data entry),the model checker will verify the vicinity of the risk factors space, according to the defined bounds.

Since the remaining risk factors are categorical, we decide not to vary them through the model checker analysis as that could lead to generating a verification space too far apart from each patient data. We recognise that this choice might limit the application of the proposed framework to simpler risk scores that only use categorical risk factors. Nevertheless, the idea of our framework is to provide local verification of complex models, meaning that for the simpler ones we can use an approach of global verification.

V. Results and discussion

Experimental evaluation of the framework was done on a Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz machine. Relevant results are presented in Table II and Spin execution statistics are shown in Table III.

Our framework considers the patients in two groups. The first one (Group 1 - G1) contains the patients that were identified as being far from the decision boundary. The second one (Group 2 - G2) contains the patients that are very close to the decision boundary, which are the ones that are prone to misclassification. This division allows for our confidence metric to be a binary output for each patient, as depicted in Fig.2.

The majority of the patients fall within G1, i.e., the risk assessment falls within an unambiguous classification, given our set of proximity parameters. This is an important result that confirms the relevance of the proposed framework. If G2 was to have more patients than Group 1, medical teams could easily disregard the application of the risk score and our framework.

The risk assessment accuracy is improved for G1 and degrades G2, when comparing with the GRACE overall application. This justifies the relevance of our framework, as patients that fall close to the decision boundaries tend to be misclassified. Results under the F1 measure, the weighted average of precision and recall, provide further evidence of the clear distinction within both groups.

Looking at the sensitivity and specificity metrics it is possible to see that the GRACE model tends to classify as high

Input data								Output data	
Risk Factors								Risk	Confident
AGE	SBP	CAA	HR	CR	STD	ECE	KIL	High / Low	Yes / No

AGE: age, SBP: systolic blood pressure, CAA cardiac arrest at admission, HR heart rate, CR: creatinine, STD: ST segment depression, ECE: elevated cardiac enzymes, KIL: Killip class I-IV

Fig. 2: Characterisation of data entries and our framework outputs.

TABLE II: Results from the overall GRACE risk score and for the presented framework.

GRACE overall									
Accuracy	0.5522	F1 Measure	0.2077	Sensitivity	0.7941	Specificity	0.5329	Patients	460
GRACE with online verification									
Group 1 - Confident assessed									
Accuracy	0.5730	F1 Measure	0.2336	Sensitivity	0.8065	Specificity	0.5524	Patients	384
Group 2 - Not confident assessed									
Accuracy	0.4605	F1 Measure	0.0889	Sensitivity	1.0000	Specificity	0.4459	Patients	76

TABLE III: Spin execution statistics: number of states stored, memory usage and execution time.

States			
Average	61036	Max	73012
Memory (Megabytes)			
Average	131.50	Max	132.05
Time (seconds)			
Average	0.03	Max	0.05

risk (positive event) in the vicinity of the decision boundary. This will increase the value of the sensitivity to its maximum, but will ruin the specificity score. This is interesting evidence that can be taken into account when setting the thresholds of risk discrimination.

The model checker execution time, memory used and states explored show that the framework can be used together with a medical software tool. We know that time is an important factor for medical teams, since they need to make decisions and analysis in a timely manner. Thus, and taking into account the results presented in Table. III, it is possible to see that the usage of a model checker does not increase significantly the execution time. Another important result is concerned with the fact that we only explore a small portion of the state space, which allows for the application of the framework to larger and more complex models.

These results also help validate why the GRACE risk score is widely used.

VI. Conclusion

This paper applies the formal verification technique of model checking at runtime to increase the confidence in the application of Medical Condition Risk Scores. The proposed technique consists in verifying whether an input to the MCRS model is in proximity to its multidimensional decision boundary. This principle increases trust in the risk score tool and identifies cases in which categorical risk assessment should be further examined.

With our approach we perform a tailored analysis of the decision space. Common approaches disregard problem information that is relevant for that analysis. For instance, medics are not interested on a distance verification that is based on all the features (risk factors), because some of them are binary or categorical and patients with different values for those features are not considered ”close” from a medical perspective.

The verification complexity of the model checker is sufficiently small to allow for online verification results to be produced efficiently. This is, in part, a consequence of the model checking technique itself, which expands the graph of reachable states and is therefore more efficient than both testing and sensitivity analysis.

The paper describes the practical implementation of the proposed framework, starting with a Python model for risk score computation that is translated into the Promela language. The Promela model examines input values in close proximity to the input point (the actual patient case) and the verifier generated by Spin may be executed during runtime, providing a form of online verification. Using this principle it is possible to determine if a case is close to a boundary and, to some extent, explain for which reason that occurs. The expected impact for practice is a greater ability to achieve online verification of machine learning models.

Looking forward, it might be pertinent to partially automate the specification translation step, for instance by adding a compiler component to the framework, in order to deal with scalability challenges that may arise in more complex models.

Acknowledgements

This work is partially funded by Portuguese Foundation for Science and Technology, I.P., within the project CISUC - UID/CEC/00326/2020, by European Social Fund, through the Regional Operational Program Centro 2020 and by the project AI4EU – “A European AI On Demand Platform and Ecosystem”, project funded by the European Commission, H2020-EU.2.1.1., Grant agreement ID: 825619.

REFERENCES

[1] S. Akyuz, S. Yazici, E. Bozbeyoglu, T. Onuk, O. Yildirimturk, D. Karacimen, M. I. Hayiroglu, G. Erdogan, A. O. Oner, A. N. Calik, M. Cagdas, and N. Cam, "Validity of the updated GRACE risk predictor (version 2.0) in patients with non-ST-elevation acute coronary syndrome," *Revista Portuguesa de Cardiologia*, vol. 35, no. 1, pp. 25–31, jan 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0870255115003200

[2] S. Chaki and A. Gurfinkel, *BDD-Based Symbolic Model Checking*. Cham: Springer International Publishing, 2018, pp. 219–245. [Online]. Available: https://doi.org/10.1007/978-3-319-10575-8_8

[3] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 8, no. 2, pp. 244–263, 1986.

[4] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model checking*. MIT press, 2018.

[5] B. Elbarouni, S. G. Goodman, R. T. Yan, R. C. Welsh, J. M. Kornder, J. P. DeYoung, G. C. Wong, B. Rose, F. R. Grondin, R. Gallo, M. Tan, A. Casanova, K. A. Eagle, and A. T. Yan, "Validation of the Global Registry of Acute Coronary Event (GRACE) risk score for in-hospital mortality in patients with acute coronary syndrome in Canada," *American Heart Journal*, vol. 158, no. 3, pp. 392–399, 2009.

[6] K. A. Fox, O. H. Dabbous, R. J. Goldberg, K. S. Pieper, K. A. Eagle, F. Van De Werf, Á. Avezum, S. G. Goodman, M. D. Flather, F. A. Anderson, and C. B. Granger, "Prediction of risk of death and myocardial infarction in the six months after presentation with acute coronary syndrome: Prospective multinational observational study (GRACE)," *British Medical Journal*, vol. 333, no. 7578, pp. 1091–1094, 2006.

[7] K. A. Fox, G. FitzGerald, E. Puymirat, W. Huang, K. Carruthers, T. Simon, P. Coste, J. Monsegu, P. G. Steg, N. Danchin, and F. Anderson, "Should patients with acute coronary disease be stratified for management according to their risk? Derivation, external validation and outcomes using the updated GRACE risk score," *BMJ Open*, vol. 4, no. 2, pp. 1–10, 2014.

[8] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning," *Proceedings - 2018 IEEE 5th International Conference on Data Science and Advanced Analytics, DSAA 2018*, pp. 80–89, 2019.

[9] I. Goodfellow, P. McDaniel, and N. Papernot, "Making machine learning robust against adversarial inputs," *Communications of the ACM*, vol. 61, no. 7, pp. 56–66, 2018.

[10] R. Guidotti, A. Monreale, S. Ruggieri, D. Pedreschi, F. Turini, and F. Giannotti, "Local rule-based explanations of black box decision systems," 2018.

[11] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models," *ACM Computing Surveys*, vol. 51, no. 5, pp. 1–45, 2018.

[12] G. J. Holzmann and M. H. Smith, "Software model checking: extracting verification models from source code," *Software Testing, Verification and Reliability*, vol. 11, no. 2, pp. 65–79, jun 2001. [Online]. Available: http://doi.wiley.com/10.1002/stvr.228

[13] S. Paredes, T. Rocha, P. de Carvalho, J. Henriques, J. Morais, and J. Ferreira, "Integration of Different Risk Assessment Tools to Improve Stratification of Patients with Coronary Artery Disease," *Medical and Biological Engineering and Computing*, vol. 53, no. 10, pp. 1069–1083, 2015.

[14] S. Paredes, T. Rocha, D. Mendes, P. Carvalho, J. Henriques, J. Morais, J. Ferreira, and M. Mendes, "New approaches for improving cardiovascular risk assessment," *Revista Portuguesa de Cardiologia*, vol. 35, no. 1, pp. 5–13, 2016.

[15] J. Souyris, V. Wiels, D. Delmas, and H. Delseny, "Formal Verification of Avionics Software Products," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, vol. 5850 LNCS, pp. 532–546. [Online]. Available: http://link.springer.com/10.1007/978-3-642-05089-3_34

[16] M. Tierney, "Software engineering standards: the 'formal methods debate' in the uk," *Technology Analysis & Strategic Management*, vol. 4, no. 3, pp. 245–278, jan 1992. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/09537329208524097

[17] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda, "Model checking programs," *Automated software engineering*, vol. 10, no. 2, pp. 203–232, 2003.