29th IEEE International Requirements Engineering Conference (RE2021), 20-24 Sept 2021, South Bend, USA

ARF: Automatic Requirements Formalisation Tool

1st Aya Zaki-Ismail *, 2nd Mohamed Osama *, 3rd Mohamed Abdelrazek *, 4th John Grundy †, and 5th Amani Ibrahim *
Information Technology
* Deakin University , † Monash University
Melbourne, Australia
* {amohamedzakiism, mdarweish, mohamed.abdelrazek, amani.ibrahim}@deakin.edu.au , †<john.grundy@monash.edu>

***Abstract*—Formal verification techniques enable the detection of complex quality issues within system specifications. However, the majority of system requirements are usually specified in natural language (NL). Manual formalisation of NL requirements is an error-prone and labor-intensive process requiring strong mathematical expertise and can be infeasible for a large number of requirements. Existing automatic formalisation techniques usually support heavily constrained natural language relying on requirement boilerplates or templates. In this paper, we introduce ARF: An Automatic Requirements Formalisation Tool. ARF can automatically transform free-format natural language requirements into temporal logic based formal notations. This is achieved through two steps: 1) extraction of key requirement attributes into an intermediate representation (we call RCM), and 2) transformation rules that convert requirements in RCM format into formal notations.**

***Index Terms*—Requirements engineering, Requirements Formalisation, Requirements Extraction**

I. Introduction

Errors in requirements affect the quality of the system and increase the time, effort and cost of system development [1]. In addition, they can also lead to catastrophic consequences in safety critical systems. Quality standards typically recommend the requirements to be specified in formal notations to be able to apply formal methods [2]. However, the majority of system requirements are specified in Natural Language (NL) [3]. Thus, it is pivotal to be able to accurately transform NL requirements into formal notations.

Manually formalising the requirements requires strong expertise in mathematics and is an error prone process. Automated formalisation tools encourage organisations to perform formal checking and translation. However, most existing (semi-)automated formalisation approaches are limited to: (1) (very) constrained requirements templates/formats [4] or (2) variations of requirements utilising constrained language [5]. In addition, most of these approaches provide only one formal notation as output although different formal notations have advantages and disadvantages. According to recent work [6], there is still a great need for better automated requirements formalisation approaches for free format textual requirements.

To address this problem we developed ARF: **A**utomatic **R**equirements **F**ormalisation tool. ARF is capable of transforming NL requirements into multiple formal notations (i.e., versions of temporal logic) providing users with more transformation flexibility. ARF can process input requirements with a much wider range of requirement formats compared to existing approaches. This is done by extracting the key requirements properties instead of relying on the rigid structure and semantics of defined templates. ARF supports the transformation into multiple formal notations by isolating the extraction layer from the transformation layer (i.e., utilising a reference model). Thus, only the transformation layer shall be adjusted/customised to support other formal notations. Further details about ARF is available here [1].

II. ARF Architecture

The main goals of ARF are to enable the automatic formalisation of a wide range of requirements formats and structures, and be extensible to multiple formal notations. To achieve this, we designed the tool by separating the natural language processing part (extraction layer) from the formalisation mapping rules (formalisation layer). Between the two layers, the requirements are represented in a semi-formal model – requirement capturing model (RCM) [7]. RCM is a comprehensive model supporting the key properties to be extracted based on analysis of the state of the art requirements representation models. In addition, it is used to enable issues detection in [8]. The extraction layer follows an NLP-based extraction technique that can extract the requirements properties and construct the corresponding RCM [9]. The mapping rules utilised in the formalisation layer are derived from the transformation techniques in [7] that enables the conversion of requirements from the RCM format into metric temporal logic (MTL) and computational tree logic (CTL). The architecture of ARF is shown in Figure 1.

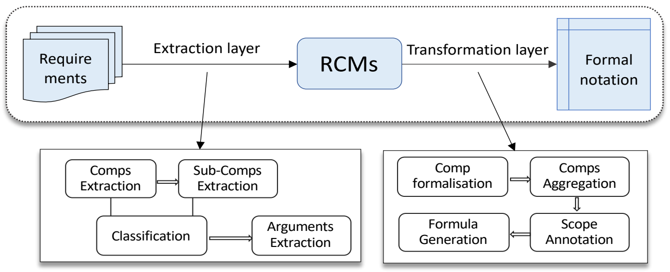
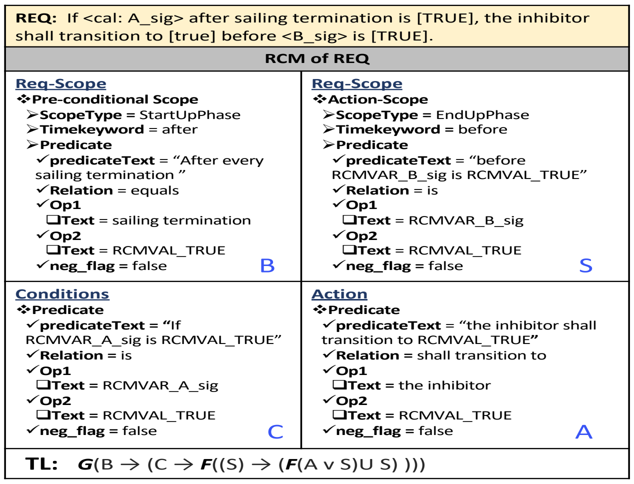


Fig. 1: ARF Processes

[1]ARF UI: https://github.com/ABC-7/ARF/blob/main/ARF_Annex.pdf

A. Intermediate Representation (RCM)

RCM represents requirements in a unified structured format detailing the breakdown of all properties within the requirement. Requirements consisting of more than one sentence are stored together in the same RCM structure to maintain sentences correlation. Each requirement sentence is represented as a primitive requirement that can hold all the extracted properties within the given sentence. Figure 2 shows the RCM representation of a requirement example.



REQ: If <cal: A_sig> after sailing termination is [TRUE], the inhibitor shall transition to [true] before <B_sig> is [TRUE].

RCM of REQ

Req-Scope
❖**Pre-conditional Scope**
➢**ScopeType** = StartUpPhase
➢**Timekeyword** = after
➢**Predicate**
✓**predicateText** = "After every sailing termination "
✓**Relation** = equals
✓**Op1**
❑**Text** = sailing termination
✓**Op2**
❑**Text** = RCMVAL_TRUE
✓**neg_flag** = false
B

Req-Scope
❖**Action-Scope**
➢**ScopeType** = EndUpPhase
➢**Timekeyword** = before
➢**Predicate**
✓**predicateText** = "before RCMVAR_B_sig is RCMVAL_TRUE"
✓**Relation** = is
✓**Op1**
❑**Text** = RCMVAR_B_sig
✓**Op2**
❑**Text** = RCMVAL_TRUE
✓**neg_flag** = false
S

Conditions
❖**Predicate**
✓**predicateText** = "If RCMVAR_A_sig is RCMVAL_TRUE"
✓**Relation** = is
✓**Op1**
❑**Text** = RCMVAR_A_sig
✓**Op2**
❑**Text** = RCMVAL_TRUE
✓**neg_flag** = false
C

Action
❖**Predicate**
✓**predicateText** = "the inhibitor shall transition to RCMVAL_TRUE"
✓**Relation** = shall transition to
✓**Op1**
❑**Text** = the inhibitor
✓**Op2**
❑**Text** = RCMVAL_TRUE
✓**neg_flag** = false
A

TL: $\boldsymbol{G}(B \rightarrow (C \rightarrow \boldsymbol{F}((S) \rightarrow (\boldsymbol{F}(A \vee S) U\ S))))$

Fig. 2: RCM High Level Structure [9]

B. Extraction Layer

In the extraction layer [9], the input requirements are first pre-processed to overcome inherent natural language issues and enable more reliable interpretations (e.g., closed word unification and foreign word substitution). ARF then utilises the Stanford CoreNLP library alongside WordNet and Prolog inference engine to analyse the input sentence. Identified properties of a requirement sentence are extracted along with their breakdowns. A primitive requirement is created in compliance with the extracted properties (i.e., structure adapting to the identified properties in the sentence). First, components of a given sentence are extracted (i.e., each representing one clause). Then, the sub-components within each component are located utilising syntactic and semantic analysis – by identifying (1) the head (has a defined grammatical role) and (2) the body (linked to the head through syntactic/semantic relations) of each sub-component. After that, each extracted component and sub-component are classified into one of the RCM's classes. Finally, the arguments breakdowns of the sub-components are identified.

C. Transformation Layer

Each primitive requirement is converted to the target formal formula of temporal logic (TL) utilising meta-models that map RCM elements to the target formal language. A bottom up approach is utilised to formulate TL-formula with the support of the meta-models mapping. First, a formal predicate is created for each component while attaching the formal notations of its related sub-components (e.g., time notations in temporal logic). Then, components with the same type are aggregated through traversing the coordinating relations. After that, scopes are attached to eligible preconditions/actions (if any) expressing temporal modality in temporal logic. Finally, the entire formula is generated.

III. Evaluation

We evaluated both layers of ARF (extraction and transformation layers) on a set of curated requirements from existing case studies in the literature. The dataset consists of 162 requirement sentences and is available online in [2].

We used precision, recall and F-measure to assess the soundness of the extraction layer. The extraction achieves a promising performance achieving 79% recall 95% precision and 86% F-measures [9]. On the other hand, having a correct RCM ensures the correctness of the generated formulas. To assess the correctness of the transformation layer, we tested it on the correct RCMs of the 162 requirements and the manual assessment of the output shows the reliability of the transformation layer [7]. The output of extraction and formalisation are available in [3] and [4] respectively.

IV. Conclusion and Future Work

We presented ARF, a tool supporting the extraction of key requirements properties and formalisation of NL requirements into both MTL and CTL formal notations. Our planned future work includes: (1) Supporting visual graphs for each primitive requirements. (2) Conducting a formal user evaluation on an industry case study to assess the usability and reliability of the tool. (3) Integrating more formal notations .

References

[1] A. Zaki-Ismail., M. Osama., M. Abdelrazek., J. Grundy., and A. Ibrahim., "Requirements formality levels analysis and transformation of formal notations into semi-formal and informal notations," in *SEKE*, 2021.

[2] I. ISO, "26262: Road vehicles-functional safety," *International Standard ISO/FDIS*, vol. 26262, 2011.

[3] M. Osama, A. Zaki-Ismail, M. Abdelrazek, J. Grundy, and A. Ibrahim, "Score-based automatic detection and resolution of syntactic ambiguity in natural language requirements," in *ICSME*. IEEE, 2020, pp. 651–661.

[4] R. Yan, C.-H. Cheng, and Y. Chai, "Formal consistency checking over specifications in natural languages," in *DATE*. IEEE, 2015.

[5] S. Ghosh, N. Shankar, P. Lincoln, D. Elenius, W. Li, and W. Steiener, "Automatic requirements specification extraction from natural language," SRI INTERNATIONAL MENLO PARK CA, Tech. Rep., 2014.

[6] A. Brunello, A. Montanari, and M. Reynolds, "Synthesis of ltl formulas from natural language texts: State of the art and research directions," in *26th International Symposium on Temporal Representation and Reasoning*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[7] A. Zaki-Ismail., M. Osama., M. Abdelrazek., J. Grundy., and A. Ibrahim., "Rcm: Requirement capturing model for automated requirements formalisation," in *MODELSWARD*, 2021.

[8] M. Osama, A. Zaki-Ismail, M. Abdelrazek, J. Grundy, and A. Ibrahim, "Srcm: A semi formal requirements representation model enabling system visualisation and quality checking," in *MODELSWARD*, 2021.

[9] A. Zaki-Ismail., M. Osama., M. Abdelrazek., J. Grundy., and A. Ibrahim., "Rcm-extractor: Automated extraction of a semi formal representation model from natural language requirements," in *MODELSWARD*, 2021.

[2]Dataset: https://github.com/ABC-7/RCM-Extractor/tree/master/Datasets
[3]Extraction-Output: https://github.com/ABC-7/RCM-Extractor
[4]Formalisation-Output: https://github.com/ABC-7/RCM-Model/tree/master/RCM-Auto-Transformation