Hanfor: Requirements Formalisation and Beyond

Nico Hauff[1,*,†], Elisabeth Henkel[1,†], Tobias Kolzer[1,†], Vincent Langenfeld[1,*,†] and Andreas Podelski[1]

[1]Department of Computer Science, University of Freiburg, Germany

Abstract

Context and motivation. The semantic review, a process that combines the manual inspection and formalisation of requirements with an automatic formal analysis, can help to improve requirements quality and find severe defects. **Problem.** The human needs tool support for inspection, formalisation and review of findings to be able to execute their part of the process effectively and efficiently. **Solution.** The new version of the Hanfor tool combines the existing formalisation editor with new features: reporting, lightweight checks, and the integration of formal analysis. **Conclusion and results.** We report on the addition of the new features to Hanfor and explain how they can help the human to execute their part in the semantic review process.

Keywords

Formal Requirements Analysis, Requirements Formalisation, Tool Supported Review

1. Introduction

Research on the formal analysis of requirements has recently gained increasing attention [1, 2, 3, 4, 5]. The formalisation of requirements is a necessary step for the formal analysis. It is known that humans make errors not only while writing requirements, but also when formalising requirements. With this in mind, we are seeking to improve the requirements formalisation tool Hanfor [6]. The tool helps the human to detect formalisation errors as soon as possible or to prevent them. The tool is able to give immediate feedback as in a code and compile loop. The tool also gives support to help the user understand the interactions between requirements (like stepping through code).

We have reported on the original version of Hanfor four years ago [6]. Since then, we have carried out a series of industrial cooperations where we used the *semantic review* process [7]. We experienced the need for a number of new features on which we report in this paper.

The basic functionality of Hanfor has stayed stable. We have moved the software to a modular design, which enables us to add new support features in a modular fashion. We have accompanied the ongoing refactoring by the continuous evaluation and extension of Hanfor. We have added several new features motivated by the needs that appeared during the industrial cooperations. We have used Hanfor in 41 projects with a total of about $42,000$ items (including not only requirements, but headings, info texts, ...) resulting in more than $10,000$ formalisations (formal requirements).

In the following, we give a quick overview of the semantic review process in which Hanfor is embedded (Section 2). We present the overall architecture and the new features in Section 3. New features are the extended form of tags which have become the central means for communication (Subsection 3.1), a simulator to explore the interaction between requirements (Subsection 3.2), lightweight checks on the use of variable domains (Subsection 3.3), the seamless connection to our formal

In: A. Hess, A. Susi, E. C. Groen, M. Ruiz, M. Abbas, F. B. Aydemir, M. Daneva, R. Guizzardi, J. Gulden, A. Herrmann, J. Horkoff, S. Kopczyńska, P. Mennig, M. Oriol Hilari, E. Paja, A. Perini, A. Rachmann, K. Schneider, L. Semini, P. Spoletini, A. Vogelsang. Joint Proceedings of REFSQ-2025 Workshops, Doctoral Symposium, Posters & Tools Track, and Education and Training Track. Co-located with REFSQ 2025. Barcelona, Spain, April 7, 2025.

*Corresponding author.

†These authors contributed equally.

hauffn@informatik.uni-freiburg.de (N. Hauff); langenfv@informatik.uni-freiburg.de (V. Langenfeld)

0000-0002-8972-2776 (N. Hauff); 0000-0003-3844-8292 (E. Henkel); 0009-0006-9474-0529 (T. Kolzer); 0000-0001-9835-6790 (V. Langenfeld); 0000-0003-2540-9489 (A. Podelski)

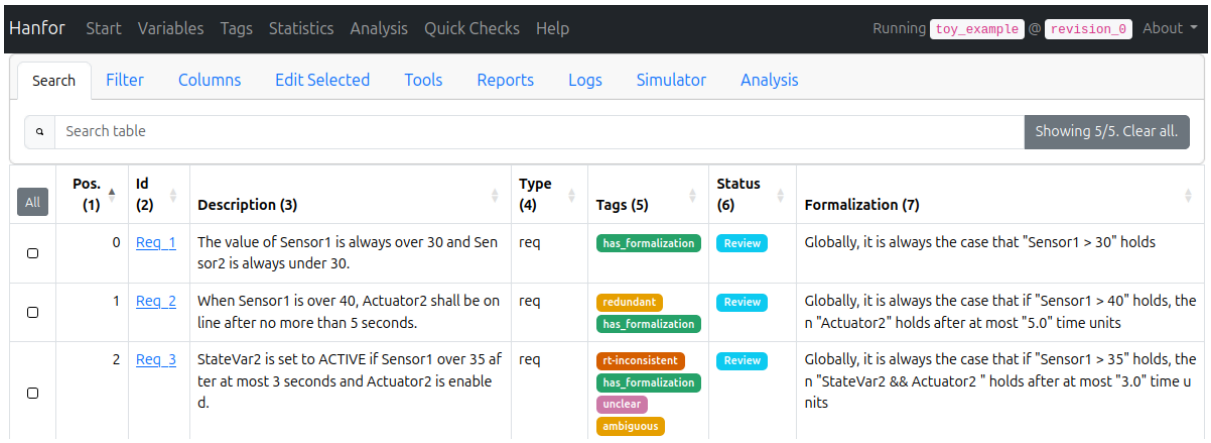

Figure 2: *Start* page of HANFOR within the *Search* tab: tabular view of imported requirements with their requirement ID and type, their process status, as well as added tags and formalisations.

analysis tool (Subsection 3.4), and telemetry which allows us to measure how the human effort distributes over the single steps in the formalisation and inspection process (Subsection 3.5).

2. HANFOR in the Semantic Review Process

We call a requirements review a semantic review if it integrates the inspection of requirements by reproduction of the requirements as a formal artefact and the (automatic) formal analysis of said formal artefacts in conjunction. More precisely, in a semantic review, inspection for impactful defects (e.g. ambiguity or illegibility) is supported by deciding on a formalisation while detection of defects in the complex behaviour of the requirements (e.g. inconsistencies) is solved by an automatic (exhaustive) analysis.

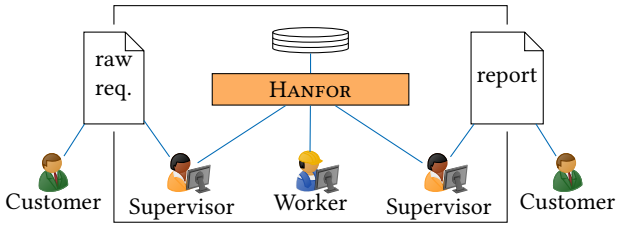

Figure 1: HANFOR in the semantic review process.

Here, we will include a short summary of the semantic review process that we apply (for a comprehensive overview, see [7]): A service provider offers the semantic review to a *customer*. The semantic review runs outside of the usual development process, performing the semantic review parallel to, e.g., elicitation and review activities performed during system development. After receiving the requirements document from the customer, a *supervisor* (a role knowledgeable in formal analysis and requirements) performs an initial check for road blocks for the formalisation. The requirements are then imported into HANFOR[1] to be formalised by *workers*. People performing the worker role are required to have experience in requirements quality and foundations in formal logics in order to detect defects and irregularities in the requirements, and to formalise the requirements in a formal language. In the case of HANFOR the requirements pattern language HANFORPL [8]) is used to express formal requirements in an accessible way. The formalised requirements are then input into a formal analysis tool (such as ULTIMATE REQCHECK [2]). The accumulated results are reviewed by a supervisor (optionally in cooperation with the worker) and then reported to the customer to act upon the findings.

The work in HANFOR is mainly performed by the worker role, thus we will from here on mainly focus on their interaction with HANFOR. In a HANFOR session, the worker is shown a table of requirements (Fig. 2). By clicking on a requirement, the formalisation dialogue is opened; a formalisation can be entered by selecting a *scope* (e.g. *Globally*) and a pattern (e.g. *it is always the case that if '**R**' holds then '**S**' holds after at most '**T**' time units*) and filling the placeholders with according expressions (Fig. 3). During formalisation, the worker sets tags (Sec. 3.1) for violations of requirements quality attributes and other impediments to formalisation. For each tag, a detailed explanation can be added in the comment

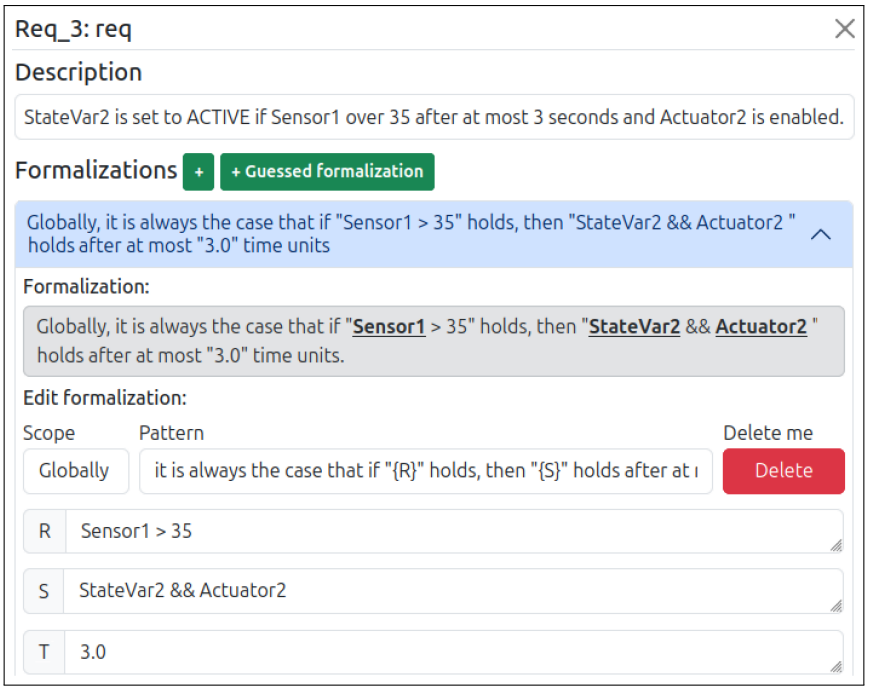

Figure 3: Pattern instantiation dialogue for *Req_3*.

field (Fig. 4). The formalisation is accompanied by supporting tools, such as, variable autocompletion, type-checking and -inference [6]. The HANFOR simulator (see Sec. 3.2) helps the worker grasp complex patterns and their interactions more easily not only during the formalisation but also during the review of requirements.

After a larger set of requirements has been formalised, the worker can decide to run fast internal checks (Sec. 3.3) and the exhaustive formal analysis in ULTIMATE REQCHECK[1] (Sec. 3.4). The worker evaluates the results produced by the analysis and decides if the formalisation has to be adapted, if the finding is an artefact of the formalisation and has to be ignored, or if the finding is a true positive.

At this point, the worker has done their part and the supervisor reviews the findings. The final report is then presented to the customer. Both the formalisation process and the entire semantic review process may be iterated based on resolved findings on the supervisor and customer side, respectively.

3. Architecture and Inner Workings

The implementation of HANFOR is based on Python 3.12 and the web application framework *flask* on the server side, as well as JavaScript on the customer side. It relies on a number of third party tools, being either directly integrated (e.g. the SMT-solver *Z3* via *pysmt*), connecting via remote API (e.g., the ULTIMATE-framework) or only suggested for combined usage (e.g. the version control system *git*).

The architecture (Fig. 5) of HANFOR is based on modules sharing a commonly accessible data storage and that supply their own web/API endpoint handling complex interactions (in a loose MVC fashion).

The session storage relates to the data of a single formalisation project, with revisions (snapshots of the project) that relate to each update of the requirements document by the customer. To be able to delegate any further version handling to git, the storage is based on JSON-files with versioning friendly (i.e. deterministic) updating. To enable rapid development, any module can register classes at the session storage, which are automatically saved and restored.

Functionality that is directly interacting with tags, variables, requirements and formalisations resides in the core-library to enable complex operations on this data. Parts of modules that are to be commonly

[1]HANFOR and ULTIMATE REQCHECK are available under https://github.com/ultimate-pa.

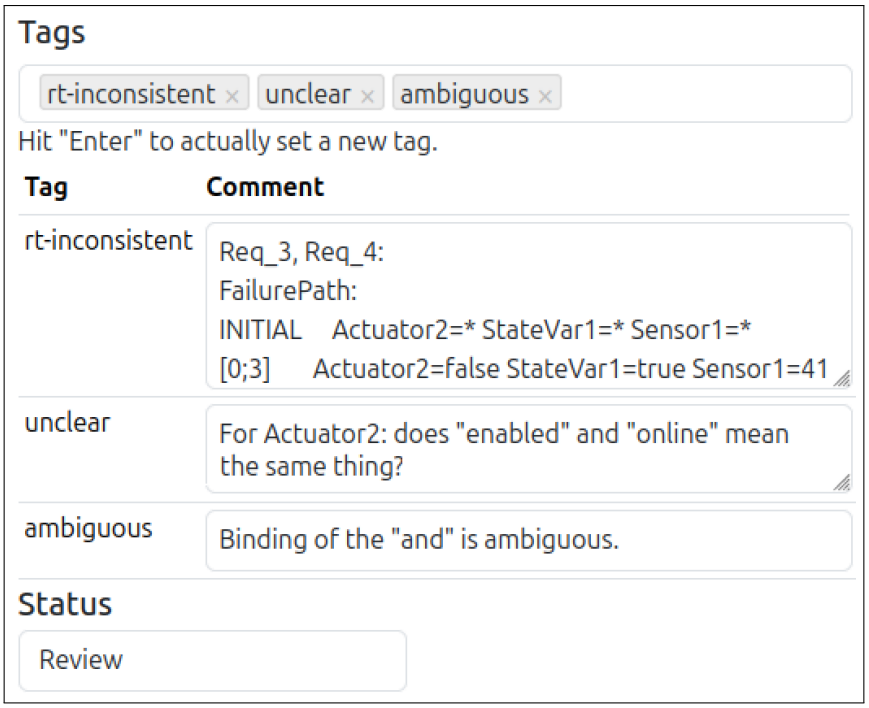

Figure 4: Tags and comments added to the formalisation of *Req_3*.

used are also moved to the core library (e.g., *Req to Pea* started as part of the simulator but is now being used in some quick checks and thus was recently moved).

On the web-facing side, we make heavy use of flask-blueprints to separate the user-facing functionality (server side as well as customer side) from the core functions.

Regarding our previous publication on Hanfor [6], the following subsections will provide an overview of the modules that were added or that changed substantially.

3.1. Tags

During a semantic review, information about defects has to be communicated to the customer as well as shared between worker and supervisor. Documenting these findings should not impede the thinking process during formalisation. Furthermore, this documentation should be structured in such a way that it facilitates the review by allowing to sort for defect classes and filter out process information. To achieve this, we implemented tags [6] that can be attached to requirements in the formalisation editor. Figure 4 shows several tags added during formalisation of *Req_3* (Fig. 3), e.g., the tag *unclear* to signify questions to the supervisor or customer and, e.g., the tag *ambiguous* to mark that the requirement has the according defect. Tags added to a requirement are also shown in the main view (Fig. 2).

A tag can have a general description to document the intention of this particular tag, and can be marked as *process only* in order to be excluded in any reports generated (as process tags are usually of no value for the customer). As soon as a tag has been defined by adding it to a requirement, it is listed on the *Tags* page, where it can also be edited. Hanfor also comes with a number of generally applicable standard tags for defects and communication.

During our case studies, tags turned out to be useful, but additional documentation and communication was required to convey the reasoning behind a large fraction of individual instances of tags. To allow this documentation to happen directly in Hanfor and also for it to be included in the report to the customer (except for comments on process tags), Hanfor shows a text field for each tag added (Fig. 4). This text field can be used to elaborate on the tag.

Recently, we started to use the tag-comment system to also show and store findings of tools such as the type checker and even the formal analysis (see Sec. 3.4). Experience in several industry projects

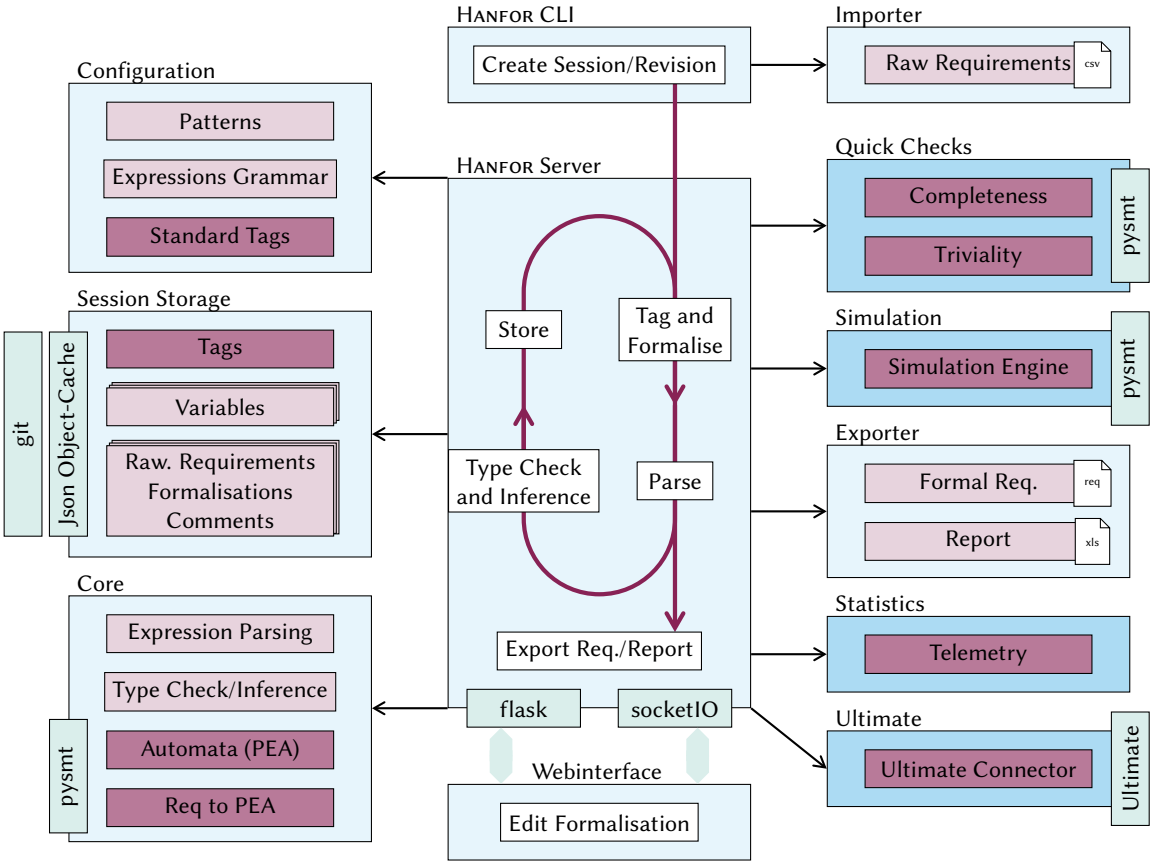

Figure 5: A basic overview of the Hanfor architecture. Columns show different aspects of Hanfor, boxes within columns show different modules (blue) and submodules (red), with relevant third-party modules (green)[1]. Novel (sub)modules are highlighted with darker colours.

shows that the tags are useful for a quick overview and structuring of findings, while the comments have replaced any additional documentation outside of Hanfor as used in older semantic review projects.

3.2. Simulator

Analysing the interaction of a small set of formal requirements or even of an edge case of a single formal requirement by hand is complex and time-consuming. The *simulator* offers a graphical interface (Fig. 6) to simulate the behaviour of a selected set of requirements directly in Hanfor.

During simulation, the values of all relevant variables are shown in a timing diagram. To control the simulation, the user can enter concrete values that should be set in the next step and the duration of the next step. After clicking the *Check*-button, the simulator offers a number of possible assignments for all free variables. The *Next*-button advances the simulation accordingly, while *Back* reverts the last selected interval (repeatedly).

In the bottom of the simulator, the formalised requirements are additionally shown as their logical formulas (for details regarding the underlying logic, see [10]). Intuitively, each formula describes a sequence of behaviours called a phase (and optionally a time bound), with each phase being separated by a semicolon. In a system conforming to the requirements, none of the sequences may never be observed fully (i.e. from the left-most to the right-most phase) as they describe what not should happen. The simulator marks phases that are currently the right-most reachable phase in green. In general, behaviour can be split and assigned to phases in different ways, several phases are marked green. Phases with time bound are coloured yellow if their time bound is not yet reached. In case of an

[1] pysmt [9] (pypi.org/project/PySMT/), Ultimate [7] (github.com/ultimate-pa/ultimate) and flask (pypi.org/project/Flask/) with SocketIO (pypi.org/project/Flask-SocketIO/)

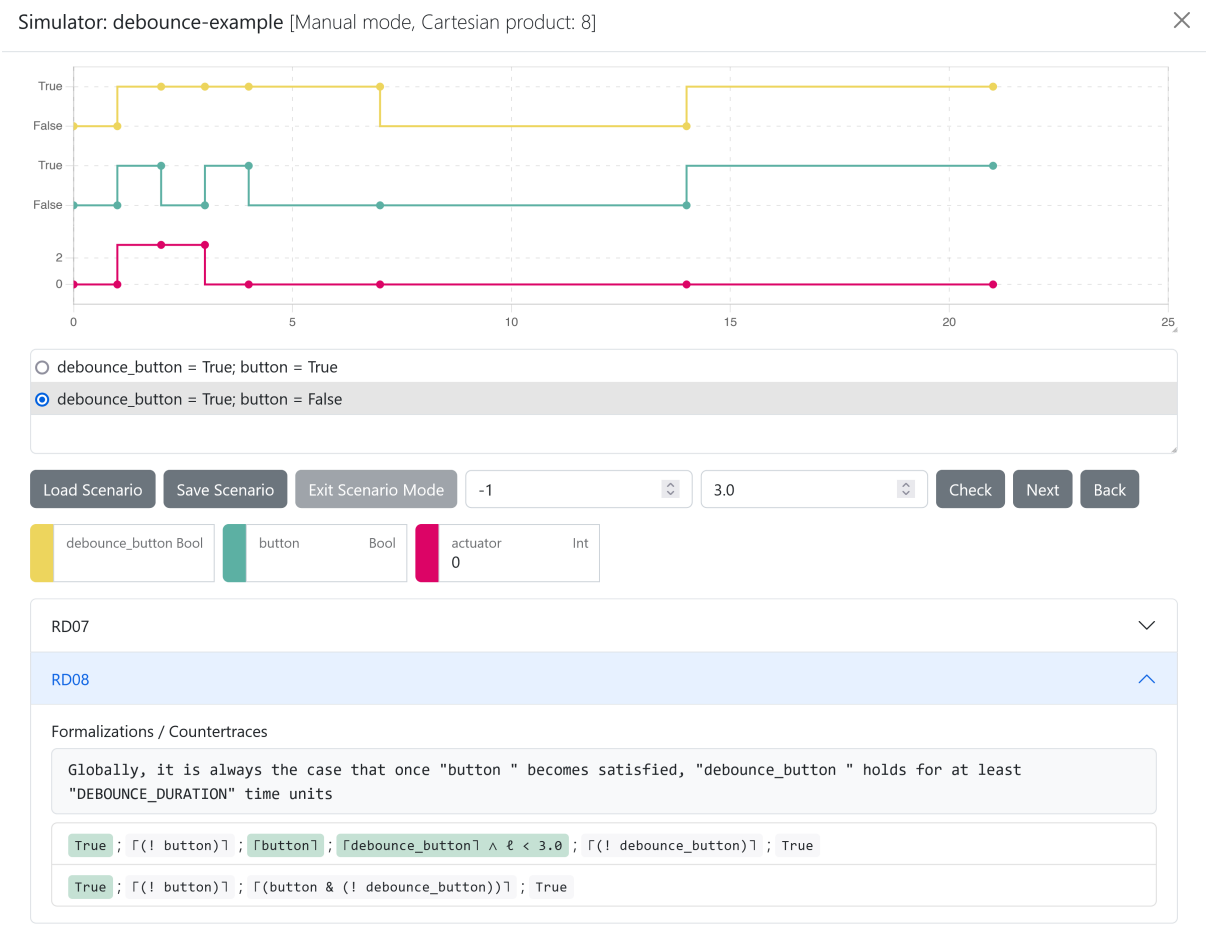

Figure 6: HANFOR simulator modal.

imminent violation of a phase's time bound, the phase is marked red. This representation is not as intuitive as showing the underlying automata that the simulation is using internally. However, it is the most compact representation and provides valuable information to guide the simulation to interesting behaviour.

Experience shows that the simulator is used sparingly, as inspecting the behaviour is relatively time-consuming. In the occasion the simulator is used, it is an effective tool to inspect if combined requirements behave the way intended in the formalisation, or if, e.g., an unwanted deadlock is encountered when the formalisations are triggered. It is equally effective in refreshing one's memory of the exact semantics of a requirements pattern.

3.3. Quick Checks

HANFOR supports the worker by providing feedback on the quality of formalisations as early as possible. While we perform exhaustive formal analyses in an external tool (see Sec. 3.4), all checks that can possibly be performed in line with the formalisation are directly executed in HANFOR. This started with simple checks like syntactical correctness and type checking/inference [6], and is now extended to additional lightweight checks:

Requirements sets may define admissible values or value ranges for variables mentioned in the requirements. Intuitively, to be complete, each value admissible should also be at least element of one observable in the requirements. Or, in other words, the requirement should at least mention each of the admissible values at least once (possibly also as part of a range). We call a failure to do so **domain incompleteness**, which can hint at an incompleteness in the raw-requirements or at a formalisation error. To check for domain incompleteness, we compute the union of all values I_v in all observables

(for each variable v), and check if $I_v \supseteq E_v$ with E_v being the set of all admissible values. If this is not the case, a value missing in I_v as well as both the sets I_v and E_v are displayed, to guide investigation of the finding.

For example, in a set of requirements, the permissible values of `speed` were set to $0 \leq \texttt{speed} \leq 100$. The two requirements containing the `speed` variable contain the observables $\texttt{speed} < 30$ and $\texttt{speed} > 30$. The check for domain incompleteness reported the value 30 not being referred to anywhere, which seems to be an omission in one of the observables.

Equally, in any requirement, no observable for a variable v, should have only elements outside the admissible range. We call this failure **domain incompatibility**, which has to be either a defect in the raw-requirements or in the formalisation. To check for domain incompatibility, we check for each observable P_v of variable v if the observable value is wholly outside the environment $P_v \cap E_v \neq \emptyset$. If this is the case, the observable P_v as well as the permissible values E_v are reported.

With the same variable as from the example above, a requirement contained the observable $\texttt{speed} > \texttt{FULLSPEED}$ with the constant $\texttt{FULLSPEED} = 100$ being defined elsewhere. A *domain incompatibility* was reported by HANFOR because the aforementioned requirement ties to exclusively set `speed` outside the permissible range. We were able to track this defect back to the worker using the wrong comparative symbol[2].

3.4. Ultimate Access

While the quick checks allow for immediate feedback, a semantic review in HANFOR takes long to detect complex defects (e.g. redundancy, rt-inconsistency and vacuity [1, 2, 4]). These analyses are implemented as part of the program analysis framework ULTIMATE REQCHECK due to their computational demands. The *Ultimate connector* (Fig. 5) connects HANFOR seamlessly to ULTIMATE, handling all data transfer and result integration in the background, so that running formal analysis regularly can be performed by the workers themselves, e.g., as an overnight test cycle.

Technically, the *Ultimate connector* connects to the web-API of an instance of ULTIMATE REQCHECK running on a dedicated (usually more powerful) machine. As visible in the main view of HANFOR (Fig. 2), there is an *Analysis* tab in which the analysis process can be started by selecting a configuration (setting the respective parameters within ULTIMATE REQCHECK) and the set of requirements to be analysed. The analyses usually take between some minutes to several hours. Running analyses tasks can be seen by switching to the *Analysis* page.

After analysis is finished, findings (i.e. a defect was found) are directly integrated into HANFOR as tags. Each tag refers to the kind of analysis, with additional information being supplied in the form of a comment containing the tool output.

3.5. Telemetry

Unfortunately, scientific investigations into the formalisation process itself are rare, as the formalisation step is usually only seen as a necessary step to perform formal analyses and not as a valuable review activity. With the telemetry feature of HANFOR, we intend to get a more nuanced view of the requirements formalisation process as executed by our workers, i.e., which requirements are finished in no-time versus which requirements take long or are finally disregarded for formalisation. The telemetry will also give insight into HANFOR itself, as it tracks what features are used during the formalisation.

On the technical side, HANFOR was enabled to run a *socketIO* connection in combination with its *REST*-API (see Fig. 5), that is used to transmit events generated by the user interface. Events are emitted by e.g. opening or closing a page and opening a formalisation dialogue. Data generated this way is written into a server side time series database.

We are currently in the process of collecting data of five different workers tasked with each formalising a slightly altered version of the pseudo real-world requirements published by Houdek and Raschke [11].

[2]Both examples are simplified versions of defects happening during the formalisation of pseudo-realistic automotive requirements published by Houdek and Raschke [11].

Data will be analysed for the formalisation times and interaction timelines with Hanfor.

4. Conclusion

Originally, the requirements formalisation tool Hanfor was developed out of the necessity to formalise requirements as precondition to formal analysis. Emergence of our semantic review process during industrial case studies, however, showed that the formalisation step itself is of far greater importance as an individual component i.e. leading to a thorough inspection even with little to no domain knowledge.

In this paper, we gave an overview of new features we added to Hanfor to support the inspection by enabling fast documentation as well as the quality of resulting formalisations. Furthermore, we presented the move of Hanfor to support research on requirements formalisation.

The development of Hanfor continues as part of an industrial validation project.

Acknowledgments

Authors N. Hauff, E. Henkel, V. Langenfeld, and A. Podelski were supported by the Bundesministerium für Bildung und Forschung (BMBF, Federal Ministry of Education and Research, Germany) under reference no. 03VP11880 *SystemValid*.

References

[1] E. Henkel, N. Hauff, L. Funk, V. Langenfeld, A. Podelski, Scalable redundancy detection for real-time requirements, in: RE, IEEE, 2024, pp. 193–204.

[2] V. Langenfeld, D. Dietsch, B. Westphal, J. Hoenicke, A. Post, Scalable analysis of real-time requirements, in: RE, IEEE, 2019, pp. 234–244.

[3] J. Frattini, L. Montgomery, J. Fischbach, D. Méndez, D. Fucci, M. Unterkalmsteiner, Requirements quality research: a harmonized theory, evaluation, and roadmap, Requir. Eng. 28 (2023) 507–520.

[4] A. Post, J. Hoenicke, A. Podelski, Vacuous real-time requirements, in: RE, IEEE Computer Society, 2011, pp. 153–162.

[5] A. Katis, A. Mavridou, D. Giannakopoulou, T. Pressburger, J. Schumann, Capture, analyze, diagnose: Realizability checking of requirements in FRET, in: CAV (2), volume 13372 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 490–504.

[6] S. Becker, D. Dietsch, N. Hauff, E. Henkel, V. Langenfeld, A. Podelski, B. Westphal, Hanfor: Semantic requirements review at scale, in: REFSQ Workshops, volume 2857 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021.

[7] D. Dietsch, V. Langenfeld, B. Westphal, Formal requirements in an informal world, in: 2020 IEEE workshop on formal requirements (FORMREQ), IEEE, 2020, pp. 14–20.

[8] E. Henkel, N. Hauff, V. Langenfeld, L. Eber, A. Podelski, Systematic adaptation and investigation of the understandability of a formal pattern language, Requir. Eng. 29 (2024) 3–23.

[9] M. Gario, A. Micheli, Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms, in: SMT Workshop 2015, 2015.

[10] J. Hoenicke, Combination of processes, data, and time, Ph.D. thesis, Carl von Ossietzky University of Oldenburg, 2006.

[11] F. Houdek, A. Raschke, Adaptive exterior light and speed control system, in: ABZ, volume 12071 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 281–301.