

Trusta: Reasoning about Assurance Cases with Formal Methods and Large Language Models

Zezhong Chen^a, Yuxin Deng^{a,*} and Wenjie Du^b

^aShanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, 200062, China

^bShanghai Normal University, Shanghai, 200233, China

ARTICLE INFO

Keywords:

Assurance cases
Trustworthiness derivation trees
Large language models
Formal methods
Constraint solving

ABSTRACT

Assurance cases can be used to argue for the safety of products in safety engineering. In safety-critical areas, the construction of assurance cases is indispensable. We introduce the Trustworthiness Derivation Tree Analyzer (Trusta), a tool designed to enhance the development and evaluation of assurance cases by integrating formal methods and large language models (LLMs). The tool incorporates a Prolog interpreter and solvers like Z3 and MONA to handle various constraint types, enhancing the precision and efficiency of assurance case assessment. Beyond traditional formal methods, Trusta harnesses the power of LLMs including ChatGPT-3.5, ChatGPT-4, and PaLM 2, assisting humans in the development of assurance cases and the writing of formal constraints. Our evaluation, through qualitative and quantitative analyses, shows Trusta's impact on improving assurance case quality and efficiency. Trusta enables junior engineers to reach the skill level of experienced safety experts, narrowing the expertise gap and greatly benefiting those with limited experience. Case studies, including automated guided vehicles (AGVs), demonstrate Trusta's effectiveness in identifying subtle issues and improving the overall trustworthiness of complex systems.

1. Introduction

In critical safety domains such as healthcare, automotive, and aviation, longstanding adoption of assurance cases has proven essential for aligning system reliability and safety with industry standards [1–3]. An assurance case comprises a systematically documented collection of evidences supporting a set of claims about a product's features, ensuring their adequacy for specific applications in designated environments. It is often visualized as a tree diagram, with the root node symbolizing the primary system claim and leaf nodes representing evidence. Assurance cases are pivotal in risk management, identifying system risks and demonstrating effective mitigations to uphold safety performance. Additionally, they facilitate consensus among diverse stakeholders regarding the system's requisite attributes, serving as a vital communication tool. Internationally recognized safety standards, including ISO 26262 [4] for automotive safety and DO-178C [5] for aviation software, advocate for the use of assurance cases. Their application spans various sectors [6, 7]—nuclear, healthcare, defense, and beyond—underscoring their adaptability.

Creating and assessing assurance cases for complex systems is labor-intensive and time-consuming, demanding significant manual effort from detailed documentation to comprehensive evaluation by safety assessors [8]. This process is not only prone to human error but also grapples with scalability and adaptability amid evolving system requirements or regulatory changes. These challenges highlight the need for automation and artificial intelligence integration, combined with human expertise for more efficient assurance case development.

To enhance reasoning in assurance cases, we introduced Trustworthiness Derivation Trees (TDTs) [9], utilizing formal methods for a streamlined approach. TDTs distill assurance cases to their essence: claims and evidences. Conversion from traditional assurance cases to TDTs involves: (i) Translating auxiliary components of Goal Structuring Notation (GSN) [10, 11] and Claim-Argument-Evidence (CAE) [12] into node descriptions while maintaining core components (goals and solutions). (ii) Enhancing principal components with formal expressions and parameters for precise representation. Formal expressions enable the automation of reasoning, advancing the evaluation of system properties through TDTs. Figure 1(a) shows the widely recognized GSN representation of assurance cases, while Figure 1(b) shows the CAE notation. In contrast, Figure 1(c) gives a TDT representation. The unique aspect of the

*Corresponding author

Email address: yxdeng@sei.ecnu.edu.cn (Y. Deng)

TDT, distinct from the GSN and CAE notations, is the incorporation of formal expressions. This makes it possible to perform automatic reasoning from bottom to top.

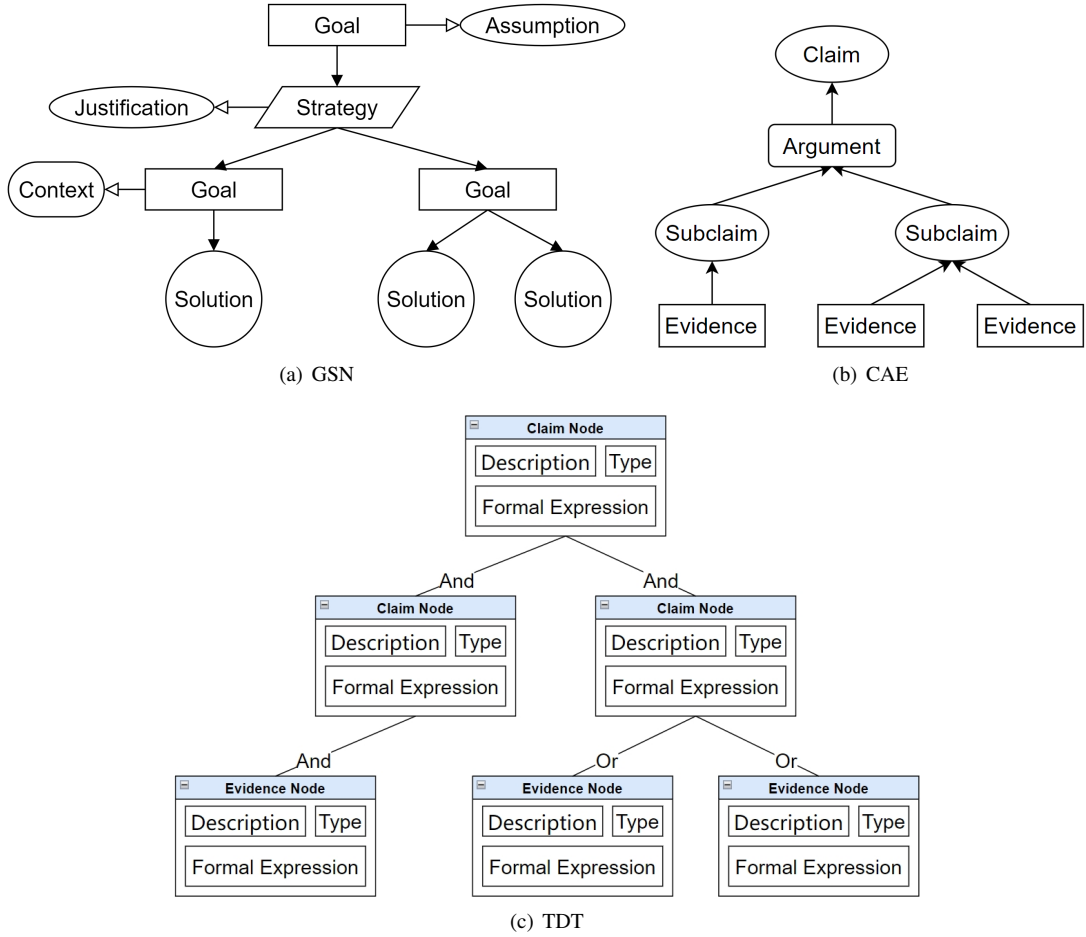


Figure 1: Three graphical representations of assurance cases.

We present Trustworthiness Derivation Tree Analyzer (Trusta), a desktop application designed for the automatic construction and verification of TDTs. Trusta offers a graphical interface for user-friendly interaction with TDTs and incorporates a lightweight Prolog interpreter and solvers like Z3 [13] and MONA [14] for addressing formal constraints. Furthermore, supported by large language models (LLMs), Trusta aids in decomposing complex claims into sub-claims and converting natural language-formulated goals into constraint-based expressions¹. This dual-step approach streamlines node creation and the translation of textual goals into formal constraints, marrying machine intelligence with human expertise to overcome traditional challenges in assurance case development.

Through case studies, including automated guided vehicles, Trusta has proven effective in identifying subtle issues, demonstrating its utility in error analysis and risk identification. The tool not only simplifies the creation of assurance cases but also enhances their expressiveness and accuracy by leveraging formal methods and automated reasoning.

The main contributions of this work include the following:

1. We introduce Trusta, integrating formal methods and large language models to innovate assurance case creation and verification. It leverages the rationality of formal methods to mitigate the hallucinations of LLMs, while utilizing the intelligence of LLMs to reduce the high usage costs of formal methods.

¹Obvious or abstract claims do not need to be transformed into detailed constraint expressions; they can simply be represented by Boolean values.

2. We conduct a cross-domain evaluation of language models (ChatGPT-3.5 [15], ChatGPT-4 [16], PaLM 2 [17]) and find that AI-generated assurance cases effectively guide humans in creating and optimizing them.
3. Through real-world applications, we validate Trusta’s effectiveness in identifying potential risks and streamlining the assurance case development process.

The rest of the article is structured as follows. Section 2 delves into the theoretical background. Section 3 introduces the architecture and functionalities of Trusta. Section 4 explores strategies for developing prompts in assurance case creation and verification. Section 5 demonstrates how Trusta expands from an initial claim node to generate the entire TDT. Section 6 describes an experiment validating Trusta’s benefits and showcases a real-world application of Trusta. Section 7 offers a comparative analysis of Trusta with existing methodologies. Finally, Section 8 summarizes the key contributions and discusses possible future work.

Our tool Trusta is available at <https://gitlab.com/TysonSir/Trusta>.

2. Background

In this section, we review some background knowledge about assurance cases, trustworthiness derivation trees, large language models, and constraint solvers.

2.1. Assurance Cases

The assurance case [18], also known as safety case, is an essential construct within safety-critical systems for demonstrating the safety and reliability of a system within specific operational contexts. These cases typically encompass aspects of system design, development, and maintenance, with an ultimate aim to ensure that the system meets safety and reliability criteria to achieve expected performance in real-world operation. The theoretical origin of assurance cases is traced to the domain of logical reasoning, notably introduced by the British philosopher Stephen Toulmin in 1958 [19]. The concept gained prominence with the rapid development in complex industries and the wide use of novel automation technologies, as humans faced unprecedented technological risks [20]. The evolution and widespread practical application of the assurance case were notably influenced by the 1988 Piper Alpha oil platform disaster [21], underscoring the vital role of systematic, structured argumentation in assessing and establishing system safety in increasingly intricate and risk-prone technological landscapes.

Today, assurance cases, or safety cases, play a crucial role across various domains, particularly in industries that demand high standards of safety, reliability, and compliance. Representative application fields include:

- **Aerospace industry** [22, 23]: Due to stringent safety requirements, aerospace engineering employs assurance cases to verify and assure the safety and reliability of airplanes [24], satellites [25], and spacecraft systems [26].
- **Railway industry** [27, 28]: Assurance cases are used to substantiate the safety and reliability of railway systems, such as signaling, train control, and operating equipment, reducing accident risk and ensuring passenger and staff safety.
- **Automotive industry** [29, 30]: With the advent of autonomous driving [31], assurance cases are deployed to argue for the safety and reliability of self-driving systems.
- **Medical devices** [32]: Medical device manufacturers (e.g., infusion pumps [33], pacemakers [34]) utilize assurance cases to demonstrate the safety and compliance of the design, manufacturing, and usage processes of their products.
- **Nuclear energy industry** [2, 35, 36]: Given stringent demands for safety and compliance, assurance cases are employed to assess the safety of nuclear power stations, facilities, and nuclear material management systems.
- **Oil and chemical industry** [37–39]: In the oil, gas, and chemical sectors, assurance cases are utilized to evaluate and ensure safety and reliability throughout the process, preventing major accidents, averting environmental disasters, and safeguarding workers and environmental safety.
- **Military and defense** [40]: In the highly security-sensitive military and defense sector, assurance cases are used to evaluate the safety and reliability of weapon systems, communication systems, and defensive mechanisms.

- **Finance and banking** [41]: Financial and banking industries leverage assurance cases to verify the security and compliance of financial transaction systems, safeguarding financial data and transactions.
- **Safety management and regulation development** [42]: In shaping safety management and regulations, such as cybersecurity regulation [42], school disaster prevention [43], and pandemic control policies [44], assurance cases play a role in risk assessment, design, and confirmation of control measures, provision of safety evidence, and promoting continuous improvement, thereby ensuring system safety and effective risk management.

The purpose of an assurance case is to articulate a clear, comprehensive, and dependable argument that a system's operation meets acceptable safety within a specific environment [18]. An assurance case serves as a tool for communicating ideas and information, often conveying content to a third party such as regulatory authorities. To achieve this convincingly, it must be as *clear* as possible. The *system* referred to by an assurance case can be any object, such as a pipeline network, software configuration, or a set of operating procedures; the concept is not confined to considerations of traditional engineering "design". Absolute safety is an unattainable goal, and the existence of an assurance case is to persuade others that the system is sufficiently safe, embodying *acceptable safety* with tolerable risks. Safety argumentation must take premises into consideration, as nearly any system might be unsafe if used improperly or unexpectedly, such as arguing for the safety of conventional house bricks [45]. Therefore, part of the work of an assurance case is defining the context or specific environment of safety. An assurance case consists of three main elements, namely goals, argumentation, and evidence, and the relationship between these three elements is depicted in Figure 2 (a).

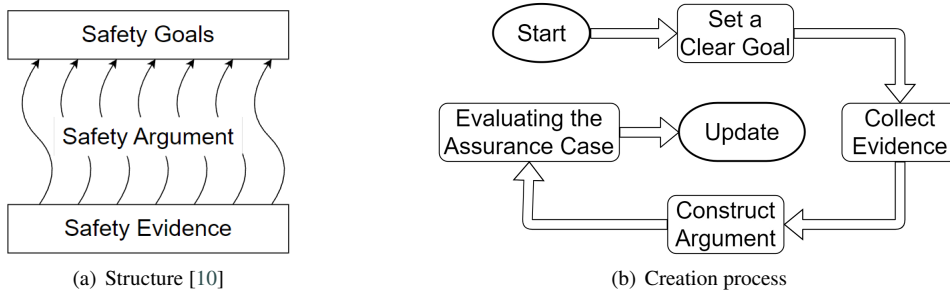


Figure 2: A overview of assurance cases, illustrating their structural components and creation process.

The process of creating an assurance case consists of four basic steps: identifying goals, gathering evidence, constructing arguments, and evaluating the assurance case [46]. As shown in Figure 2 (b), these steps build the fundamental framework of the assurance case, providing directions for safety engineers and project managers. This structured approach ensures a coherent and transparent connection between the goals, argumentation, and evidence, facilitating a clear and persuasive presentation of the system's safety and reliability. It is noteworthy that these four steps are not completed all at once but are iteratively performed throughout the project development process. As the project evolves and requirements change, the assurance case may need to be updated and modified. Furthermore, to ensure the quality and effectiveness of the assurance case, these four steps require good collaboration among the team members. This iterative and collaborative approach ensures that the assurance case remains aligned with the project's ongoing development and continues to reflect an accurate and robust representation of the system's safety and reliability.

2.2. Trustworthiness Derivation Trees

In our previous work, we introduced Trustworthiness Derivation Trees (TDTs) [9], an evidence-based model designed to analyze and enhance the trustworthiness of complex software systems. TDTs systematically refine overarching trustworthiness properties into key components and further decompose them until basic evidences are attained. This hierarchical structuring facilitates effective communication among stakeholders and rigorous verification of system properties.

2.2.1. Application in Software Development Life Cycle

TDTs serve as a robust framework for ensuring system trustworthiness throughout the software development life cycle. During the requirement analysis phase, TDTs provide a clear communication medium for stakeholders to agree

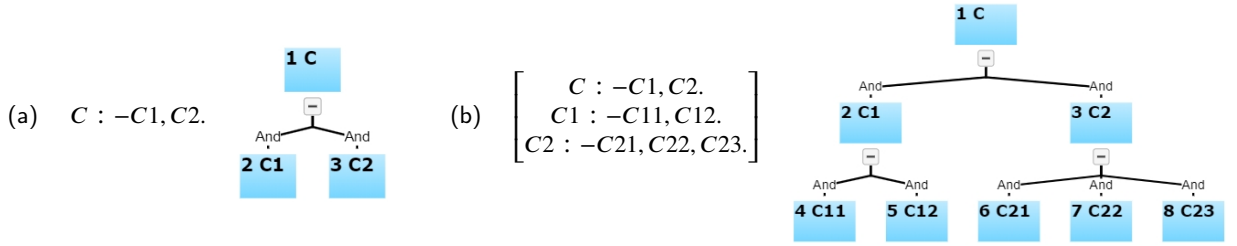


Figure 3: Two examples of rule texts and TDT skeletons

on the system’s trustworthiness properties. In the product validation phase, TDTs enable deductive reasoning to verify that the system meets the specified trustworthiness criteria. This dual functionality not only enhances the clarity and consistency of requirements but also supports rigorous validation processes.

Our approach has been piloted in over a dozen real-world software development scenarios, where TDTs have proven effective in identifying and resolving subtle issues related to system trustworthiness. The ability to systematically decompose trustworthiness into verifiable sub-properties and evidence has demonstrated significant benefits in improving both the development process and the final system quality.

2.2.2. Construction of TDTs

TDTs can be constructed in two primary ways:

1. **Interactive Editing:** Users can create and edit TDTs using our visualization tool which provides a graphical user interface for manipulating the tree structure. This approach allows for intuitive construction and modification of TDTs through actions such as adding or deleting nodes, modifying node content, and rearranging the tree layout.
2. **Prolog Rule Import:** Alternatively, TDTs can be generated by importing Prolog rules into a tool. These rules, written in Prolog’s Horn clause syntax, define the relationships between nodes and can automatically construct the tree skeleton based on predefined implications. A single rule can define a two-level subtree, while multiple rules can construct more complex multi-level trees, as illustrated in Figure 3.

This flexibility ensures that TDTs are accessible to users with varying preferences and expertise, whether they prefer manual construction or automated rule-based generation.

2.2.3. Automated Reasoning

TDTs leverage first-order predicate logic to specify properties and the implications between nodes. By utilizing Prolog for rule definition and inference, TDTs benefit from automated reasoning capabilities, ensuring that each derivation step maintains logical soundness. This formal foundation allows for rigorous validation of trustworthiness properties, making TDTs a powerful tool in the arsenal of software engineering methodologies focused on system assurance and reliability.

2.3. Large Language Models

Large language models [47] have their origins in the progressive evolution of machine learning algorithms and natural language processing techniques. They mark a significant advancement from traditional rule-based systems, employing deep learning architectures such as Transformers [48], introduced by Vaswani et al. in 2017. Application domains for these models are diverse, encompassing machine translation, text generation, sentiment analysis, summarization, and more. The implementation rationale of large language models lies in their ability to process and generate human-like text by learning from vast amounts of textual data, capturing intricate patterns and dependencies in languages. Advantages of these models include their high versatility and adaptability across various tasks, often outperforming task-specific models. However, they are not without disadvantages; their large-scale nature demands extensive computational resources for both training and inference. Additionally, concerns regarding ethical considerations, biases embedded within the training data, and the potential lack of interpretability and transparency make the deployment and use of large language models a complex consideration.

Large language models are capable of accomplishing a wide range of tasks. Their utilization is straightforward, necessitating only an input box through which “prompts” are sent to guide the model’s responses. However, truly

harnessing the full potential of these models is less straightforward. It requires a certain expertise in crafting these prompts. We have categorized several techniques for making effective use of large language models, as summarized in Section 4.

2.4. Constraint Solvers

Constraint solvers [49] originated from the field of artificial intelligence and mathematical programming in the latter half of the 20th century, becoming an essential tool for solving problems expressed through constraints. The application fields of constraint solvers are manifold, including scheduling, planning, resource allocation, and various optimization problems. The implementation principle relies on techniques such as backtracking, consistency checking, and local search, often coupled with heuristics, to explore the solution space systematically and efficiently. Advantages of constraint solvers include their flexibility in modeling complex relationships and the ability to find optimal or near-optimal solutions. However, their disadvantages may involve high computational costs for large or complex problems and difficulty in modeling some real-world scenarios.

For example, constraint solvers are widely used in airline scheduling [50], where constraints like the maximum number of working hours for pilots, mandatory rest periods, and aircraft maintenance schedules must be simultaneously satisfied. In this application, constraint solvers enable the creation of feasible schedules that adhere to all necessary regulations, though the complexity and size of the problem may present computational challenges.

3. Tool Architecture and Implementation

Figure 4 provides an overview of Trusta’s execution flow and architecture. Developed using Python’s PyQt library [51], Trusta serves as a desktop application, functioning as an IDE for constructing TDTs, graphical representations of assurance cases, with support for various graphical transformations. Trusta comprises three key modules: TDT Creator, TDT Evaluator, and Report Generator, each detailed below.

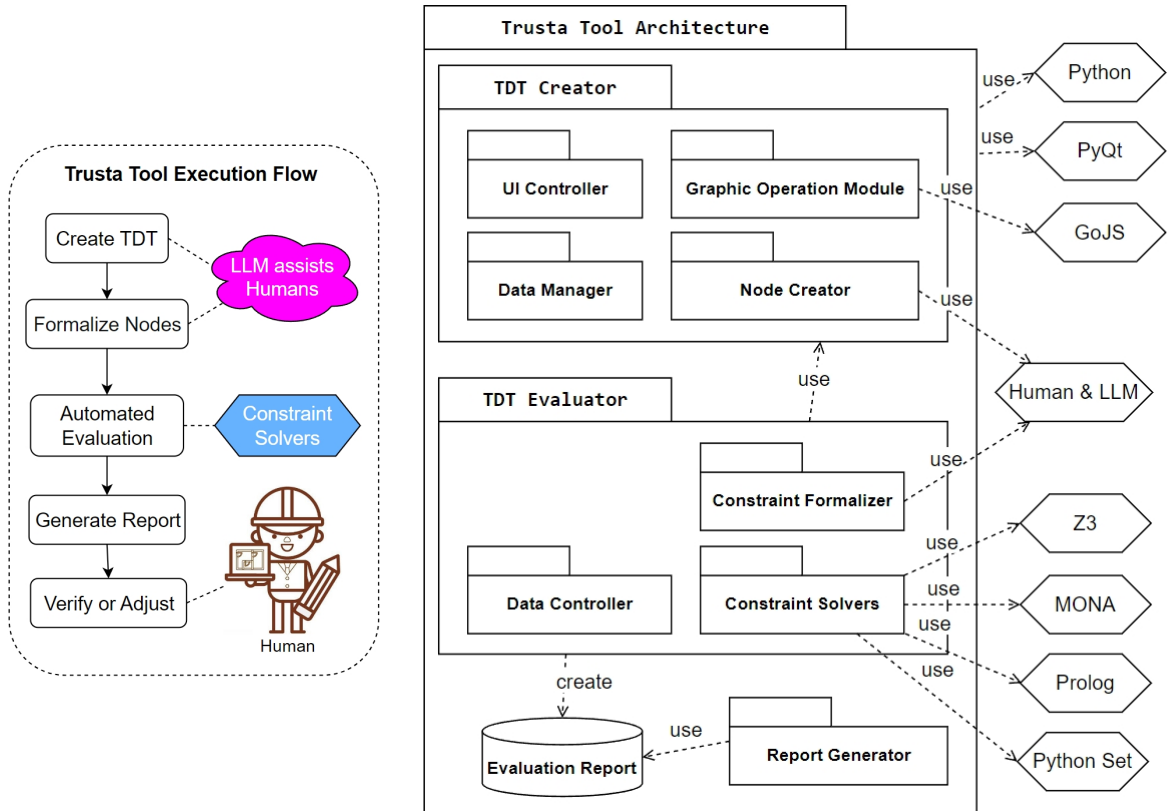


Figure 4: Execution flow and architecture of Trusta.

3.1. TDT Creator

The TDT Creator consists of four sub-modules: (1) a UI controller is in charge of responding to users' actions, (2) a node creator utilizes a large language model to derive child nodes from the upper layer, (3) a data manager can modify the data in a tree, (4) a graphic operation module uses the data of a tree to render TDT graphics and interactively modify the tree.

UI Controller Figure 5 showcases the TDT creation interface in Trusta. It features dynamic rendering of TDTs, with functionalities for node manipulation, color adjustment, tree rotation, and subtree management. Selected node information is editable in the panel's lower section, accompanied by a project explorer and node information outline on the sides.

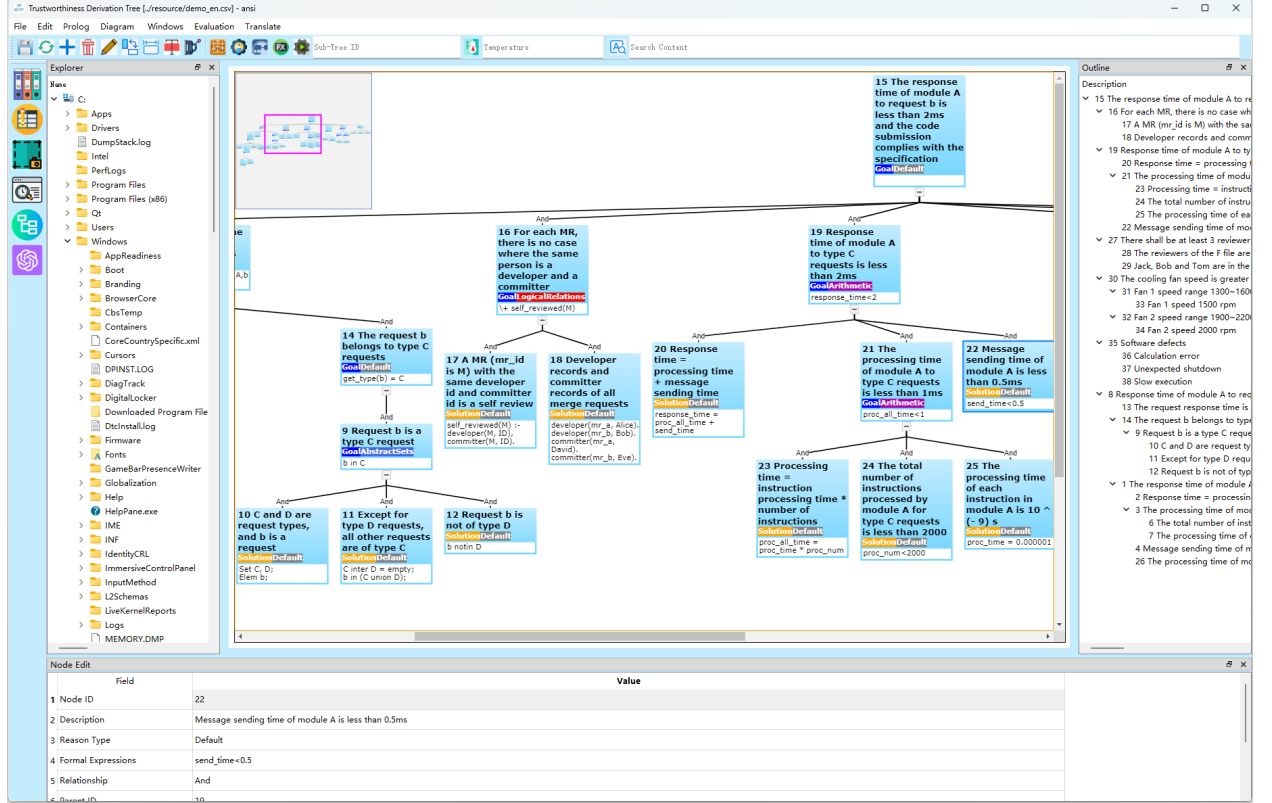


Figure 5: Main interface of Trusta with example nodes demonstrating three types of constraints.

Node Creator Utilizing Prolog's syntax for rules and axioms [9], this sub-module enables structured TDT construction through large language model integration, simplifying node division with immediate feedback and adjustability for user precision. The invocation of a large language model, particularly for complex tasks like assurance case generation, requires carefully crafted prompts. The specifics of these prompts are outlined in Section 4.

Data Manager Responsible for storing and editing TDTs, this module facilitates node addition, deletion, and modification, supporting users in refining node content with descriptions, types, and formal expressions for comprehensive TDT creation.

Graphic Operation Module Transforming TDT data into visual diagrams, this module employs the GoJS library [52] for interactive chart creation, embedded within a PyQt framework for enhanced user interaction.

3.2. TDT Evaluator

This is the module where formal methods are used for automatic reasoning about TDTs. We use three constraint solvers [53] to check the validity of the properties specified by the formal expression in each node of a TDT. Since different solvers are good at different types of reasoning, we use the *Type* field in every node to indicate the evaluation type. For example, the type “AbstractSet” in a node means that the formal expression in the node involves set operations about abstract sets, so we are going to employ MONA to solve the constraints. The process involves the translation of the natural language descriptions within nodes into formalized constraints, a task that can be undertaken through manual translation or through interactive translation with the assistance of a large language model [47].

Constraint Formalizer According to our experience with industrial case studies, we have summarized four types of constraints commonly encountered: logical relations, arithmetic, abstract sets, and concrete sets. The first three require the use of constraint solvers, whereas the last can be managed using Python’s built-in set syntax. To design prompts for the LLM that generate constraint expressions for the first three types, it is essential to understand the formal languages supported by Trusta. Below, we present the formal syntax of each language.

- Logical relations capture the dependencies and conditions between different expressions. They are used to model the logical flow and conditions within assurance cases. The syntax definition of logical relations is as follows:

```
constraint ::= predicate | \ + predicate
rule ::= predicate : - body.
body ::= predicate | \ + predicate | predicate, body
predicate ::= atom (term_list)
term_list ::= term | term, term_list
term ::= atom | variable | number | structure | list
atom ::= lower_case_atom | quoted_atom | special_symbol
```

- Arithmetic constraints involve mathematical relationships between variables and constants. They are evaluated to ensure numerical properties hold true. The syntax definition of arithmetic constraints is as follows:

```
constraint ::= expr | expr operator expr
expr ::= variable | constant | expr operator expr
operator ::= + | - | * | / | // | % | ** | > | < | == | >= | <= | != | And | Or | Not
```

- Abstract sets represent collections of entities. Operations like union, intersection, and subset are used to manipulate these sets. The syntax definition of abstract sets is as follows:

```
constraint ::= element operator set | set operator set
set ::= {element, ...} | empty
element ::= any_identifiable_entity
operator ::= union | inter | \ | sub | in | notin | = | ~=
```

Interactive Translation To facilitate the conversion of natural languages into constraint expressions, the interactive translation interface within Trusta is illustrated in Figure 6. The underlying conceptual framework draws inspiration from Cosler’s work (nl2spec) [47] on translating natural language into temporal logics. We referenced the workflow of nl2spec and have adapted the original prompt words designed for translating temporal logics, as shown in Listing 8 in Appendix A. These adjustments facilitate the seamless transition from descriptive languages to formal constraints, enhancing the applicability and efficacy of the translation process.

In a TDT, the generated formulas are not standalone; they need to be reasoned about jointly with sibling and child nodes. Therefore, we have incorporated information from these related nodes into the prompt input for the large model

responsible for generating the formulas. This information consists of sub-translations of sibling and child nodes [47], which are fragments of formulas output by previous node translations, as shown in Figure 7. As a result, the nodes in a TDT require a bottom-up approach to formal translation, where leaf nodes only need to include the information from their sibling nodes, and the translation of the first node does not require any input from other nodes. The information from child and sibling nodes is automatically added to the entries under “Adding subtranslations” when opening the interactive translation interface (Figure 6), eliminating the need for manual operations. This augmentation of the translation mechanism contributes to the broader goal of enhancing automated reasoning within the Trusta framework.

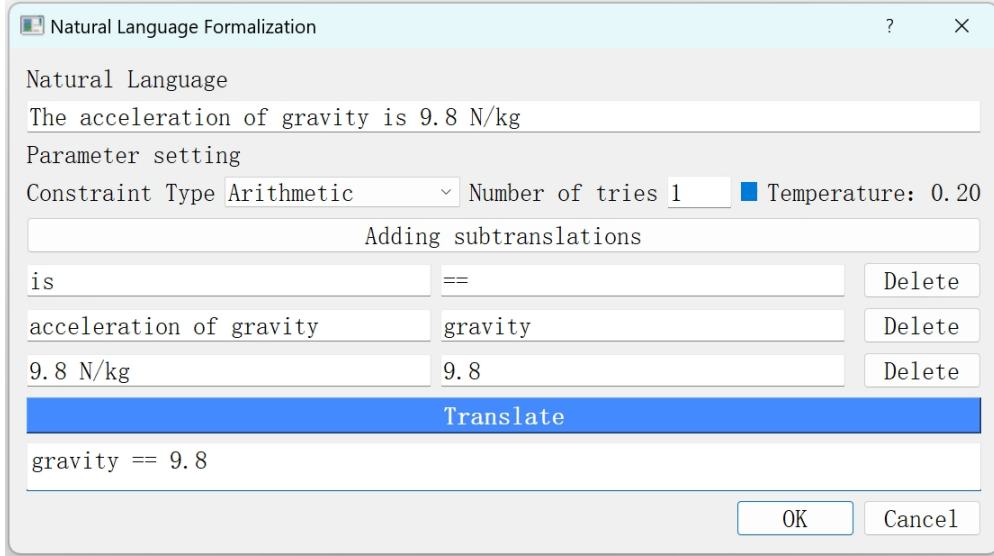


Figure 6: Interactive translation interface within Trusta powered by LLMs.

Data Controller In order to verify that the whole TDT is sound, it suffices to show the soundness of each two-level subtree in the TDT. A two-level subtree consists of a parent node and several child nodes. These child nodes represent the premises, and the parent node stands for the conclusion of the rule. Suppose $F_1, F_2, \dots, F_n$ are the formal expressions of premises, and F is the formal expression of the conclusion. In addition, we allow two types of logical relations between the child nodes and their parent node. The “And” relation means that all the premises need to be combined to lead to the conclusion. In this case, we check if the formula $F_1 \wedge F_2 \wedge \dots \wedge F_n \wedge \neg F$ is satisfiable. If it is unsatisfiable then the rule is sound. Otherwise, a solution exists and witnesses the unsoundness of the rule. The “Or” relation means that any one of the premises can lead to the conclusion. In that case, we need to check the satisfiability of the formula $(F_1 \vee F_2 \vee \dots \vee F_n) \wedge \neg F$.

Constraint Solvers The satisfiability of the formulas given above is determined by constraint solvers. According to our experience with industrial case studies, we have summarized four types of constraints commonly encountered: logical relations, arithmetic, abstract sets, and concrete sets. Unfortunately, there exists no single solver that can solve all those types of constraints. Therefore, we have to call different solvers for different constraints. If the constraints are about logical relations, we resort to a lightweight Prolog built in Trusta. For arithmetic related to first-order theories, we take advantage of Z3. For some reasoning about abstract sets, i.e. unassigned sets whose elements are not explicitly known, we make use of MONA. For concrete sets whose elements are given in terms of arrays or lists, we use Python to deal with set operations. Below we examine the first three constraint types through examples within the context of a TDT depicted in Figure 5.

The node IDs from the set $\{16, 17, 18\}$ correspond to a two-level subtree. The constraint for this subtree is captured by the expression $E_{Logical}$ in (1). It is the conjunction of three parts: the first part says that a merge request with the same developer and committer is called self-reviewed; the second part is an evidence, a list of records showing the developers and committers of some merge requests; the third part is the negation of the property in the parent node,

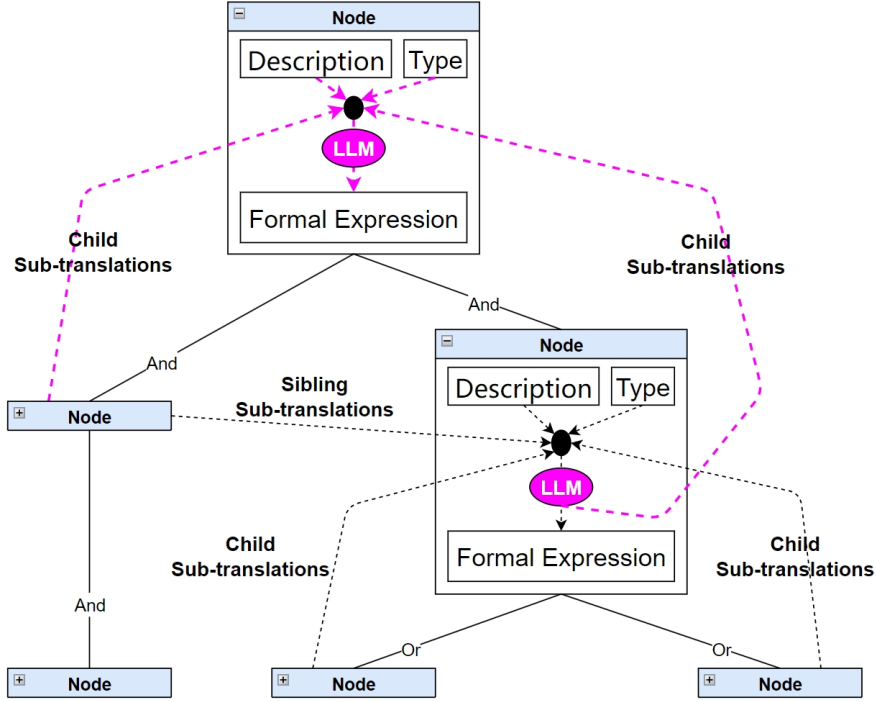


Figure 7: Sibling and child node sub-translations are integrated into the prompt inputs to generate cohesive formalized formulas.

concerning about the absence of self-reviewed merge request, where the symbol ‘\+’ is the Prolog syntax for negation. The satisfiability of the formula $E_{Logical}$ can be checked by the lightweight Prolog built in Trusta.

$$\begin{aligned}
 E_{Logical} = & \text{“self_reviewed}(M) : - \text{developer}(M, ID), \text{committer}(M, ID).”} \\
 & \wedge \text{“developer}(mr_a, Alice). \quad \text{developer}(mr_b, Bob). \\
 & \quad \text{committer}(mr_a, David). \quad \text{committer}(mr_b, Eve).”} \\
 & \wedge \neg \text{“} \backslash + \text{self_reviewed}(M).”
 \end{aligned} \tag{1}$$

Now consider the node IDs from the set $\{19, 20, 21, 22\}$. They correspond to a two-level subtree whose constraints are about arithmetics and captured by the formula $E_{Arithmetic}$ in (2). The formula is a conjunction of four parts: the first part defines the relationship between the variables *response_time*, *proc_all_time*, and *send_time*; the second and third parts define the constraints on the last two variables; the last part is again the negation of the property in the parent node. The satisfiability of the formula $E_{Arithmetic}$ can be checked by Z3.

$$\begin{aligned}
 E_{Arithmetic} = & \text{“response_time} = \text{proc_all_time} + \text{send_time”} \\
 & \wedge \text{“proc_all_time} < 1” \wedge \text{“send_time} < 0.5” \\
 & \wedge \neg \text{“response_time} < 2”
 \end{aligned} \tag{2}$$

Then we consider the node IDs from the set $\{9, 10, 11, 12\}$. They correspond to a two-level subtree that talks about abstract sets. Their constraints are captured by the formula $E_{AbstractSet}$ in (3). The formula is a conjunction of four parts: the first part defines the sets *C* and *D* together with an element *b*; the second and third parts define the constraints between *C*, *D*, and *b*. The last part is the negation of the property in the parent node. We can employ MONA to check the satisfiability of the formula $E_{AbstractSet}$.

$$\begin{aligned}
 E_{AbstractSet} = & \text{“Set } C, D; \text{Elem } b;” \wedge \text{“} b \text{ notin } D;” \\
 & \wedge \text{“} C \text{ inter } D = \text{empty}; b \text{ in } (C \text{ union } D);” \\
 & \wedge \neg \text{“} b \text{ in } C;”
 \end{aligned} \tag{3}$$

To evaluate the formal languages supported by Trusta, we analyzed their use in real-world assurance cases. The primary languages include: (1) abstract sets, used for set operations and membership criteria; (2) arithmetic constraints, prevalent in performance metrics, timing constraints, and resource allocation; and (3) logical relations, capturing dependencies and conditions for safety requirements and fault tree analysis. It is possible to increase the expressiveness of Trusta with more languages in the future.

3.3. Report Generator

Based on the results of constraint solving, Trusta reports on the vulnerabilities in the systems modeled by TDTs. More specifically, if a property is invalid, the constraint solvers generate counterexamples to witness the invalidity of the property. For example, if we change the third part of the formula $E_{Arithmetic}$ into $send_time < 1.5$, then that formula is satisfiable. One solution is ($proc_all_time = 0.9, send_time = 1.4, response_time = 2.3$). In that case, the goal $response_time < 2$ does not hold, so the TDT is unsound. This kind of feedback from the constraint solvers provides TDT developers with more explicit information about the unsafe scenarios so they can quickly fix the problems.

4. Prompt Design

Large language models have shown remarkable versatility across a wide range of tasks. However, unlocking their full potential requires a nuanced understanding of prompt engineering. This section delves into the strategic development of prompts that enhance LLMs' utility in the creation and verification of assurance cases.

Based on various prompt engineering literature [54–57] and our experience with LLMs usage, we have summarized 18 prompt techniques. They are detailed in Table 1, ranging from improving instruction quality to systematic testing. Techniques like Being Specific (T1) refine query relevance, while Solution Strategy (T12) encourages diverse solution exploration. Systematic testing, including comparison to gold standard answers (T17) and A/B testing (T18), is crucial for assessing model performance. These strategies, whether standalone or combined, aim to elicit meaningful output from LLMs based on task complexity and user goals. Additionally, Chen et al. [58] highlighted that aggregating results from multiple runs significantly enhances outcomes compared to those from a single run.

Table 1
Classification and summary of usage techniques for LLMs

Category	Technique	Technique ID
Optimizing Instruction Quality	Being Specific	T1
	Role-play	T2
	Instruction Segmentation	T3
	Specifying Steps	T4
	Providing Examples	T5
	Setting Length	T6
Leveraging Reference Text	Answer Reference	T7
	Citation Reference	T8
Task Decomposition	Intent Classification	T9
	Information Filtering	T10
	Paragraph Summarization	T11
Making the Model “Think”	Solution Strategy	T12
	Simulate Thinking Process	T13
	Asking for Omissions	T14
Combining External Tools	Embedding-based Search	T15
	Code Execution	T16
Systematic Testing	Comparing to Gold Standard Answers	T17
	Conducting A/B Tests	T18

- T1 (Being Specific): Make queries more targeted by providing the model with detailed information for more relevant answers.
- T2 (Role-play): Assign a role to the model within the query for more creative answers.

- T3 (Instruction Segmentation): Use delimiters to distinguish different parts in the query.
- T4 (Specifying Steps): List out the steps needed to complete the task to help the model generate accurate answers.
- T5 (Providing Examples): Assist the model in understanding requirements through examples.
- T6 (Setting Length): Specify the desired length of output in the query.
- T7 (Answer Reference): Allow the model to generate more accurate answers by referring to a specific text.
- T8 (Citation Reference): Instruct the model to quote specific parts from the reference text for more in-depth answers.
- T9 (Intent Classification): Decompose complex queries by analyzing the main objective in user queries.
- T10 (Information Filtering): For applications requiring long conversations, summarize or filter out previous dialogue, keeping only the key information.
- T11 (Paragraph Summarization): If dealing with long documents, split them into multiple paragraphs for summarization, and then combine these summaries.
- T12 (Solution Strategy): Make the model generate possible solutions before producing the final answer.
- T13 (Simulate Thinking Process): Allow the model to conduct an internal monologue, simulating a “thinking” process.
- T14 (Asking for Omissions): Ask the model if it has omitted important information in the problem-solving process.
- T15 (Embedding-based Search): Use embedding-based search for effective knowledge retrieval.
- T16 (Code Execution): Leverage the model’s code generation capability to perform calculations or call APIs.
- T17 (Comparing to Gold Standard Answers): Evaluate the quality of the model output by comparing it with preset gold standard answers.
- T18 (Conducting A/B Tests): Compare the effects of different prompts on the model output to find the most effective prompting strategy.

4.1. Prompt for Claim Decomposition

The prompts for claim decomposition are divided into four segments: requirement description, theoretical introduction, case studies, and content pending. Each segment is thoroughly detailed below. The complete content of these prompts can be found in Listing 7, Appendix A.

The first segment, detailed in Listing 1, focuses on requirement description. This part sets the context and defines the role of the language model as an expert in assurance cases. It provides a general format that the model’s output should follow and instructs the model to break down a given goal into various sub-goals. This section also asks the model to provide explanations for the breakdown as well as potential solutions for the sub-goals, setting up the stage for structured assurance case generation. This segment utilizes techniques T1, T2, and T4, as detailed in Table 1.

Listing 1: A prompt segment with requirement description.

```

1 You are an expert proficient in the Assurance Case.
2 Your answers always need to follow the following output format and you always have to try to provide a
  set of sub-goals. You may repeat your answers.
3 Break down the following goal into several sub-goals, these sub-goals should be able to support the
  parent goal, and explain the reasoning behind the breakdown. Finally, provide solutions that support
  these sub-goals.

```

Listing 2: A prompt segment with theoretical introduction.

```

1 Goal(claim): A goal is a claim in the argument, usually supported by sub-goals(sub-claims), strategies(
arguments) or solutions(evidences). Goals describe assertions about system characteristics, performance,
safety, etc.
2 Strategy(argument): A strategy describes ... ..
3 Solution(evidence): ... ..
4
5 The five basic CAE(claim-argument-evidence) building blocks that we have identified are:
6 1. Decomposition: partitions some aspect of the claim.
7 2. Substitution: ... ..

```

The second segment, detailed in Listing 2, focuses on theoretical introduction. This part provides an in-depth look at the definitions and terminologies employed in assurance cases. This section not only defines what a “Goal”, “Strategy”, and “Solution” are but also outlines the five basic CAE (Claim-Argument-Evidence) building blocks [12] essential for creating assurance cases. These blocks are Decomposition, Substitution, Concretion, Calculation or Proof, and Evidence Incorporation. By introducing these conceptual tools, this segment equips the model with the necessary framework to understand and generate assurance cases more effectively. This segment utilizes technique T7.

The third segment, detailed in Listing 3, focuses on case studies. This part offers multiple examples that individually highlight the use of each of the five building blocks: Decomposition, Substitution, Concretion, Calculation or Proof, and Evidence Incorporation. These examples cover various domains and goals such as self-driving cars, medical devices, and data encryption. For each example, the section details the building blocks employed, the breakdown strategy, the sub-goals, and solutions. Additionally, it provides explanations on how these elements are interconnected. These examples serve as both a comprehensive guide and a template for the model, aiding it in understanding how to structure and approach different types of assurance cases. This segment utilizes techniques T5 and T12.

Listing 3: A prompt segment with case studies.

```

1 Goal G1: The self-driving car is safe to operate on public roads.
2 Building Blocks: Decomposition
3 Break down Strategy: {"strategy": "The safety of the self-driving car can be determined by examining its
hardware and software components."}
4 Sub-goals dictionary: {"G1.1": "The sensor system is reliable.", "G1.2": "The navigation algorithm is
accurate.", "G1.3": "The emergency systems function correctly."}
5 Solutions dictionary: {"Sn1.1": "Manufacturer test reports, third-party evaluations.", "Sn1.2": "Code
audits, simulation results.", "Sn1.3": "Test scenarios, independent assessments."}
6 Explanation: G1.1, G1.2 and G1.3 can support G1, Sn1.1 can support G1.1, Sn1.2 can support G1.2, Sn1.3
can support G1.3.
7 FINISH
8
9 Goal G1: The new version of the medical device is safe.
10 Building Blocks: Substitution
11 ... ..

```

The fourth and final segment, detailed in Listing 4, focuses on content pending. This part presents an incomplete example that consists solely of a placeholder for a goal, denoted as $\langle A_NEW_GOAL \rangle$, which is intended to be decomposed. This incomplete example follows the same format as the examples in the third segment and is designed for completion by a large language model. When invoking the model, $\langle A_NEW_GOAL \rangle$ is replaced with a specific goal, as illustrated in the first line of Listing 5. This segment utilizes technique T3.

Listing 4: A prompt segment with content pending.

```

1 Goal G1: <A_NEW_GOAL>
2 Building Blocks: Break down Strategy:
3 Solutions dictionary: Explanation:
4 FINISH

```

Listing 5: LLM output when splitting nodes.

```

1 Goal G1: The automatic guided vehicle can stop safely when encountering obstacles.
2 Building Blocks: Decomposition
3 Break down Strategy: {"strategy": "The safe stopping capability of the automatic guided vehicle can be
determined by examining its perception system, control system, and braking system."}
4 Sub-goals dictionary: {"G1.1": "The perception system can accurately detect obstacles.", "G1.2": "The
control system can make timely and appropriate decisions to stop the vehicle.", "G1.3": "The braking
system can reliably bring the vehicle to a stop."}
5 Solutions dictionary: {"Sn1.1": "Sensor test reports, performance evaluations.", "Sn1.2": "Algorithm
validations, simulations.", "Sn1.3": "Brake performance tests, safety certifications."}
6 Explanation: G1.1, G1.2, and G1.3 can support G1, Sn1.1 can support G1.1, Sn1.2 can support G1.2, Sn1.3
can support G1.3.

```

The model’s output, as shown in Listing 5, is then parsed by the Trusta tool to generate the TDT nodes. This effectively bridges the gap between theoretical modeling and practical implementation, demonstrating that the model’s output is in a format compatible with Trusta for seamless integration into a workflow.

4.2. Prompt for Formalizing Constraints

The approach for converting natural languages into constraint expressions utilizes prompts structurally similar to those for claim decomposition, encompassing sections on requirement description, theoretical introduction, case studies, and content pending. However, due to the need for higher precision in constraint expressions, the theoretical introduction section includes more detailed descriptions, such as symbolic representations and standardized units of measurement, as exemplified in Listing 6.

Listing 6: Prompt segment of LLM translation from natural language to constraint expressions.

```
1 The formula should only contain variables, numbers or operators +, -, *, /, //, %, **, >, <, ==, >=, <=,
   !=, And, Or, Not.
2 Using the International System of Units (SI) to standardize the units of numerical quantities. For
   example, When we describe distance or length, we typically use meters (m) as the unit. When we measure
   mass, we use kilograms (kg). Time is usually measured in seconds (s)... ..
```

The complete prompts and the outputs from a LLM are presented in Listings 8 and 9 in Appendix A. This methodology translates natural languages into constraint expressions and then employs a constraint solver to find the solution, leverages technique T16.

5. Case Study

Together with our industrial partners, we have built TDTs in more than a dozen real-world scenarios, such as checking software build consistency and the credibility of software implementation. In fact, Trusta helped us identify some subtle issues that had previously gone unnoticed. In this section, we present a case demonstration of TDT creation and evaluation for the braking scenario of Automated Guided Vehicles (AGVs) in a warehouse. This case uses the large language model ChatGPT-4 [16].

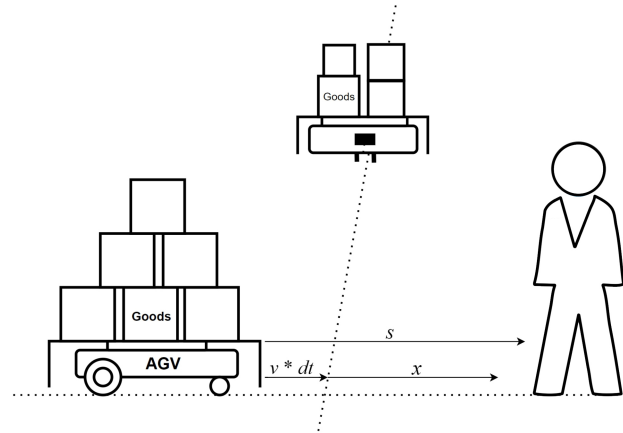


Figure 8: Braking scenario of an AGV in a warehouse.

AGVs autonomously move goods between different areas of a warehouse, as shown in Figure 8. They travel along predetermined routes, carrying several items on top of the vehicle. However, the path of one AGV intersects with another AGV or a pedestrian walkway, creating potential risks. Despite various precautions, it is still necessary to evaluate the safety and reliability of warehouses equipped with AGVs. For this purpose, we have created a TDT. Figure 8 shows the braking scenario of the AGV. The AGV on the left moves to the right at a speed of v , detecting an obstacle at a distance of s meters. After a reaction time of dt seconds, the AGV begins to decelerate and applies brakes within a distance of x meters. To avoid colliding with the obstacle, the AGV on the left must generate sufficient deceleration. However, if the deceleration is too high, it may cause the goods on the AGV to slide or even fall off, posing a safety hazard.

5.1. TDT Creation

Before formally creating the TDT, we used Trusta for a fully automated preliminary decomposition of the top-level goal. This helps users test the usability of the LLM, assess its performance, and also guides them in understanding more details related to safety goals, providing ideas for subsequent formal TDT creation.

5.1.1. Preliminary Decomposition

When building the TDT with Trusta, we first created a top-level goal, as shown in Figure 9. We established a goal node with the objective: "The AGV should be able to stop safely when encountering an obstacle." We instructed Trusta to decompose the goal into three levels, using a language model temperature setting of 0.8. This setting promotes more creativity and can uncover subgoals that might otherwise be overlooked. In the context of large language models [59], the sampling temperature is a value between 0 and 2. Higher values, such as 0.8, produce more random outputs, while lower values, such as 0.2, lead to more focused and deterministic outputs.

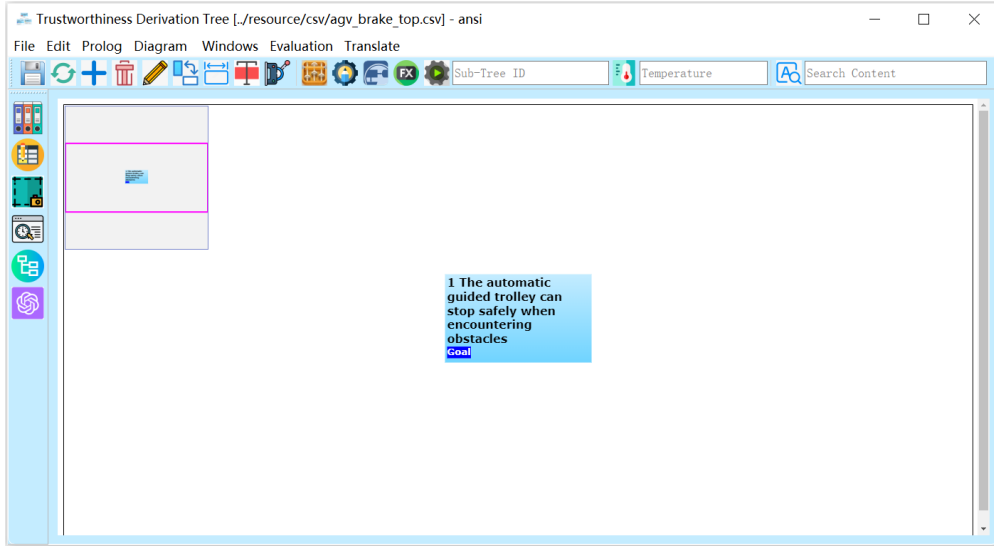


Figure 9: Screenshot of the input for the creation of TDT using Trusta in the context of AGV.

Once the specified input was prepared, Trusta efficiently generated a structured sequence of 36 nodes—23 subgoals and 13 solutions, as shown in Figure 10. The process involved 11 decompositions, each taking about 10 seconds, totaling less than 2 minutes. These decompositions strictly outlined the details that the top-level goal might involve. Figure 11 illustrates the decomposition of the top-level goal: "The AGV can stop safely when encountering obstacles" (Node 1), which was decomposed into three subgoal nodes. These subgoals collectively define the strategy for achieving the main goal, highlighting key functions and safety measures. Specifically, these include: "The AGV's sensors can accurately and timely detect obstacles." (Node 2), "The AGV's braking system can start quickly and safely upon receiving sensor signals." (Node 3), and "The AGV's control system can execute safety strategies, such as deceleration or stopping, after detecting obstacles." (Node 4). Subsequent node decompositions, similar to those in Figure 11, and the leaf node solutions are derived from higher-level nodes. Further decompositions can be found in Figures 23, 24, and 25 in Appendix B.

The above example demonstrates the use of Trusta to generate a multi-layer TDT in one go. However, in practice, we can ask the tool to decompose subgoals layer by layer, allowing users to adjust in real-time and create more granular subgoals. As the decomposition progresses, there are typically two cases that indicate further decomposition may not be necessary: (1) when the generated nodes begin to have the same meaning as their parent nodes or other existing nodes, and (2) when experts believe the current goal node can be supported with evidence. This approach to TDT creation better aligns with user expectations and ensures the process does not consume excessive time.

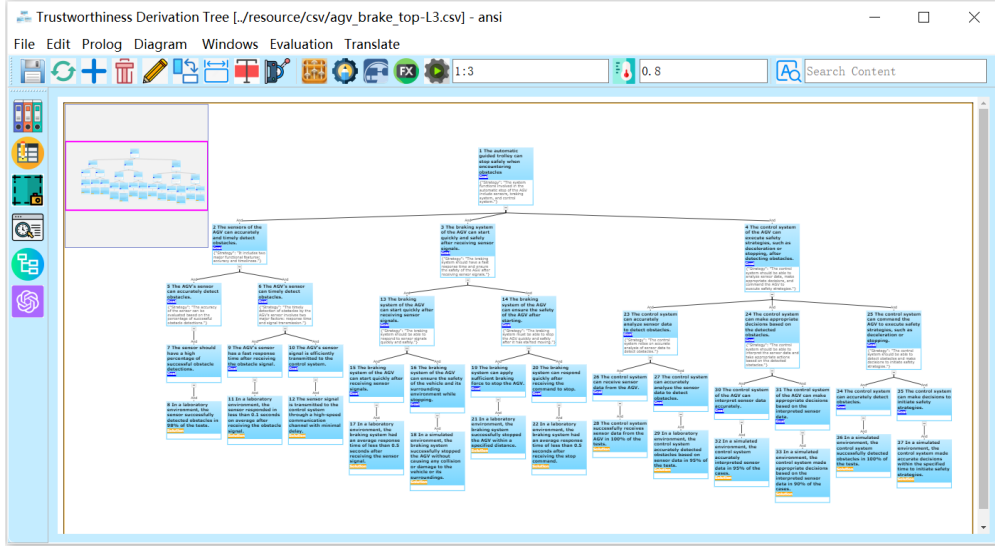


Figure 10: Screenshot of the output for the creation of TDT using Trusta in the context of AGV.

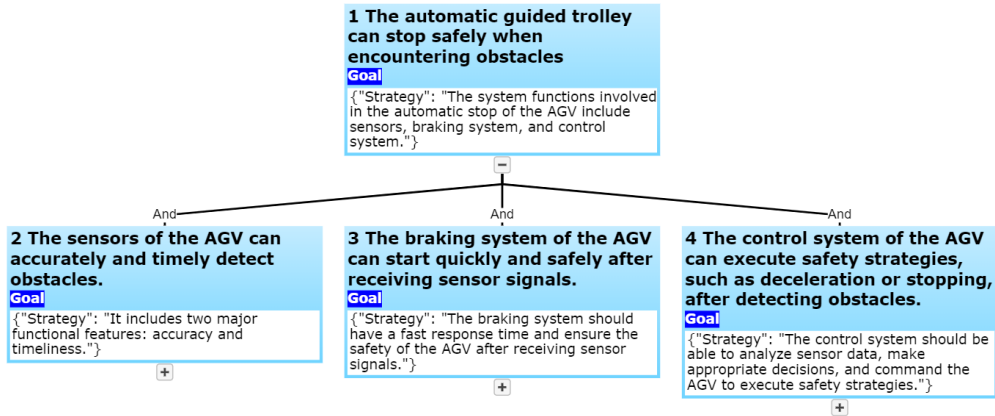


Figure 11: Top-level node decomposition: The AGV can stop safely when encountering obstacles.

5.1.2. Formal Creation

After browsing the preliminary decomposition of the TDT by Trusta, users have gained a preliminary understanding of the factors affecting the safety of AGV braking, such as the AGV's distance sensors, braking reaction time, and the performance of the brakes. However, the LLM did not address the safety of the goods carried on the AGV, which requires human intervention to add corresponding content to the TDT. The reason the LLM overlooked the goods is that the description of the top-level goal did not involve the goods, which is implicit information about the AGV.

At this point, after user intervention with Trusta, the top-level goal is decomposed into "Able to decelerate to a standstill before impact." and "The goods on the shelf will not slide." as shown in Figure 12.

Inevitably, AGVs moving in a warehouse may encounter obstacles ahead, such as people, goods, or other AGVs. The moving AGV should be able to recognize these obstacles and start decelerating and stopping before a collision occurs.

Further, we used Trusta to decompose node 2 "Able to decelerate to a standstill before impact." as shown in Figure 13(a). "Strategy" and "Solution" references were also generated. After manual analysis, redundant information in node 6 was deleted, keeping the other two nodes (nodes 4 and 5). After manual adjustment of the expressions, we obtained the arguments related to braking, i.e., braking reaction time and braking process safety, as shown in Figure 13(b).

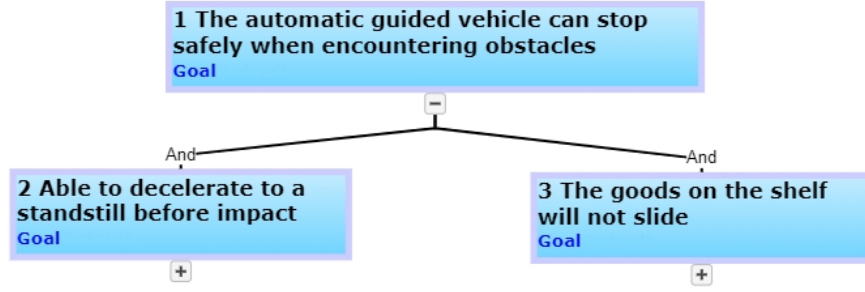


Figure 12: Human-revised top-level goals for AGV braking safety and goods stability.

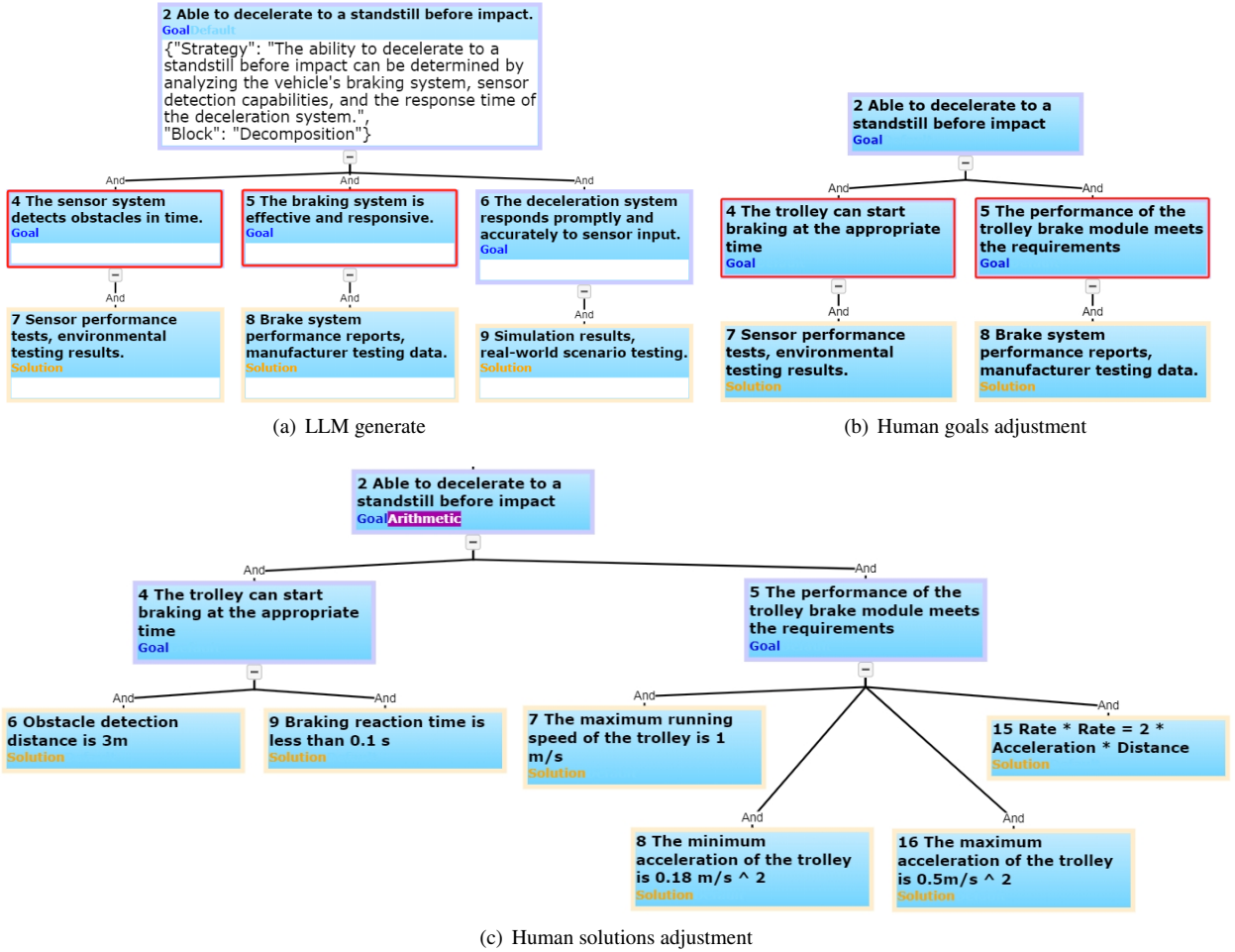


Figure 13: Human intervention in the decomposition and refinement of braking goals and solutions for AGV safety.

After each decomposition step, we need to consider whether the existing evidence can support the decomposed goal nodes. For nodes 4 and 5, Trusta has provided the required evidence descriptions as shown in nodes 7 and 8 of Figure 13(a). Users familiar with the AGV context can determine that the parameters (braking distance, speed, acceleration, time) required in nodes 7 and 8 can be obtained from the AGV's manual. Thus, we can combine actual evidence to refine nodes 7 and 8, as shown in Figure 13(c). If relevant evidence cannot be found, further decomposition of the goal nodes is needed until they can be supported by evidence. Sometimes, we also need to collect more evidence

to satisfy the reasoning. If evidence cannot be collected, it is likely that a system vulnerability has been discovered, requiring system adjustments to make it safer and obtain the evidence.

With evidence and arguments, we use the evidence to infer the validity of nodes 4 and 5, ultimately inferring the validity of node 2. This evaluation process is discussed in the next subsection. Additionally, the goods on the AGV should remain stable and not slide. We use Trusta to decompose node 3 "The goods on the shelf will not slide." and make manual adjustments, similar to the decomposition of node 2.

5.2. TDT Evaluation

To automatically evaluate the TDT using the constraint solver, we first need to create formal constraint expressions for each node. However, creating formal expressions from natural language is a high-difficulty and time-consuming task. Fortunately, the LLM can help us complete part of the task. First, the LLM generates a draft of the constraint expression, which is then adjusted by humans. Trusta then reads the constraint information and calls the constraint solver to provide evaluation conclusions. If each subnode can support its parent node, it gives "No Error"; otherwise, it provides an error prompt and counterexamples. If there are errors, the human user needs to continue adjusting the TDT (including goals, evidence, formal formulas, and even the system being evaluated) until there are no errors in the TDT, at which point the evaluation is complete, as shown in Figure 14.

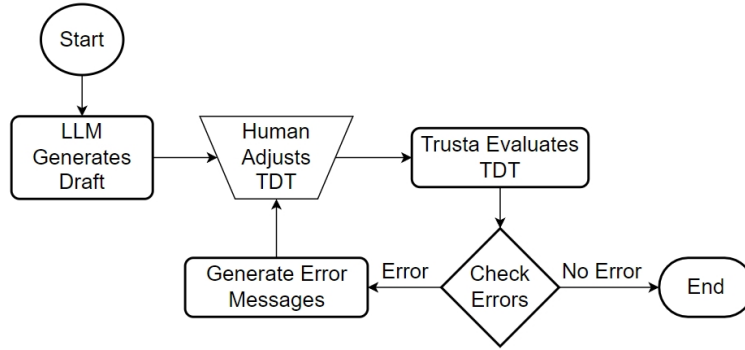


Figure 14: Flowchart of TDT evaluation using constraint solver with human intervention.

Figure 15 shows an example where manual adjustment of constraint expressions is needed. Blue nodes represent normal nodes, checked by Trusta with no logical issues. The yellow node with ID 2 indicates that the subgoals cannot fully support the goal, posing a logical risk. The "Error Report" dialog at the top of the image gives an error prompt. When the variables are as follows, $[braking_reaction_time = 1/16, rate = -2, accel = 1/4, distance = 8, obstacle_distance = 3]$, the expression $distance < obstacle_distance - rate * braking_reaction_time$ of node 2 is not satisfied. At this point, we notice that node 5's formal expression lacks an implicit condition: $rate > 0$. After manually adding this condition (as shown in node 5 of Figure 16), Trusta indicates "No Error". This completes an iteration of automatic checking and manual adjustment, further enhancing this TDT's safety.

After multiple iterations of the above process, the AGV automatic braking case is refined, and the updated TDT is shown in Figure 16. The strategy details generated during node creation are now hidden to prioritize the translation of constraint expressions in the evaluation stage. The TDT fragment shown in Figure 16 aims to prove that the AGV can brake safely when encountering obstacles. The top node has two subtrees: the left subtree argues that the AGV will not collide with obstacles, and the right subtree proves that the goods on the AGV will not slide. The left subtree relies on the equations of uniformly accelerated linear motion, which are given in Table 2. The data for these parameters can be obtained from the AGV's reference manual. The maximum operating speed is $v = 1m/s$, and the maximum deceleration is $a = 0.5m/s^2$. The right subtree uses static friction equations for the argument. Typically, the friction coefficient between the goods and the AGV's top shelf is greater than 0.2.

The development of the TDT revealed some details that need careful consideration. For example, on one hand, we should set the minimum deceleration parameter for the AGV; otherwise, collisions may occur during braking. On the other hand, it is more important to consider the materials of the cargo packaging and the shelf. The static friction coefficient of the corresponding materials should exceed a certain value to ensure the stability of the goods. Trusta has

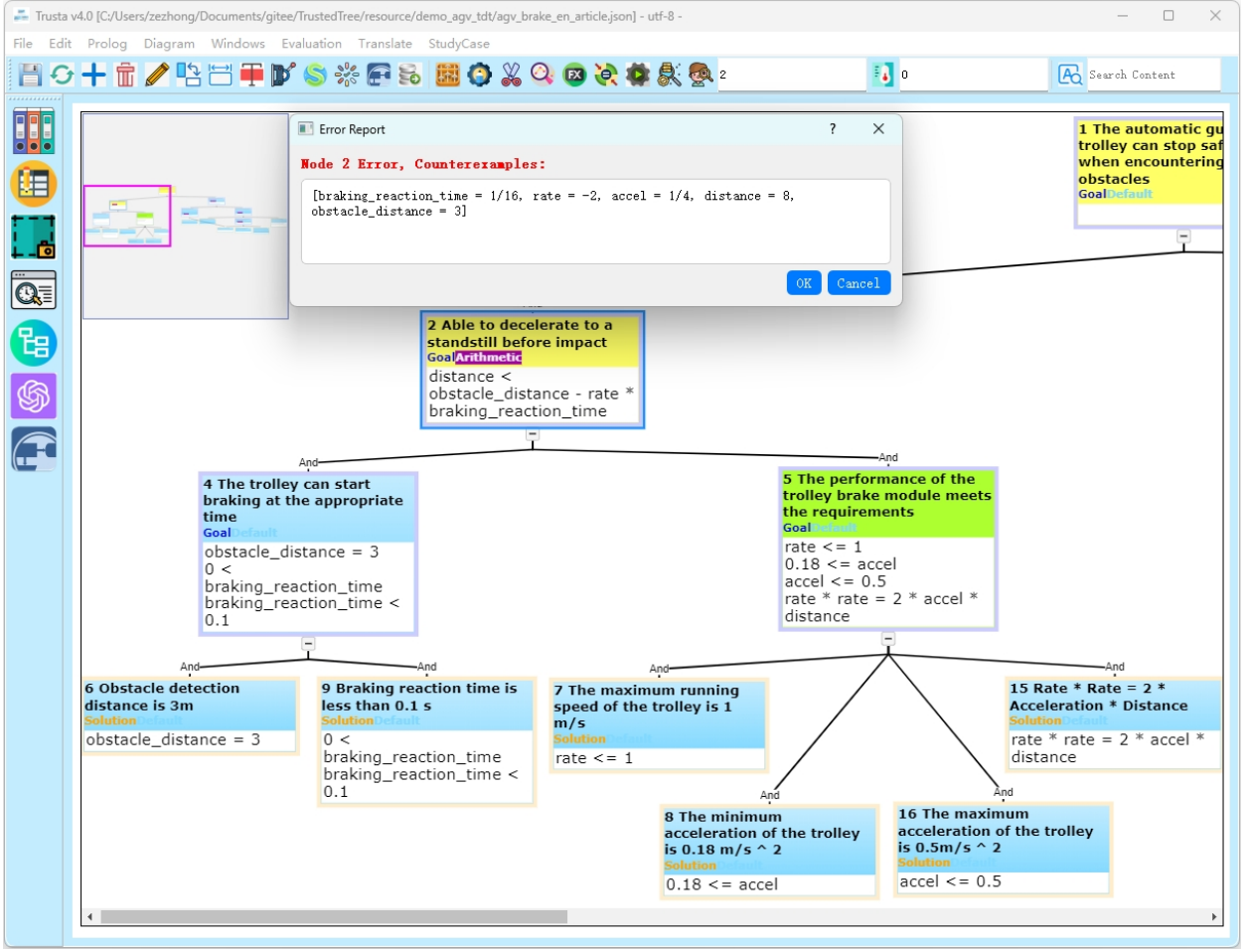


Figure 15: UI display of counterexamples and manual adjustment in TDT evaluation. Each node’s white area is described using formal formulas, constraining the solver to provide counterexamples for reasoning.

Table 2
Physical kinematic equations used in the AGV example

Equation	Detail
$s = vt$	s (displacement), v (velocity), t (time)
$v^2 = 2ax$	v (velocity), a (acceleration), x (displacement)
$F_N = mg$	F_N (normal force), m (mass), g (acceleration of gravity)
$F = uF_N$	F (sliding friction force), u (frictional coefficient)

been very helpful in adjusting these parameters. The construction and automatic evaluation of the TDT in this case study have strengthened our confidence in the safe operation of the AGV.

6. Tool Evaluation

This section evaluates two primary functions of Trusta: 1) Claim Decomposition, and 2) Formalizing and Solving Constraints. Claim decomposition is assessed both qualitatively and quantitatively through human experiments of assurance cases development. Formalizing and solving constraints is qualitatively evaluated through a case study.

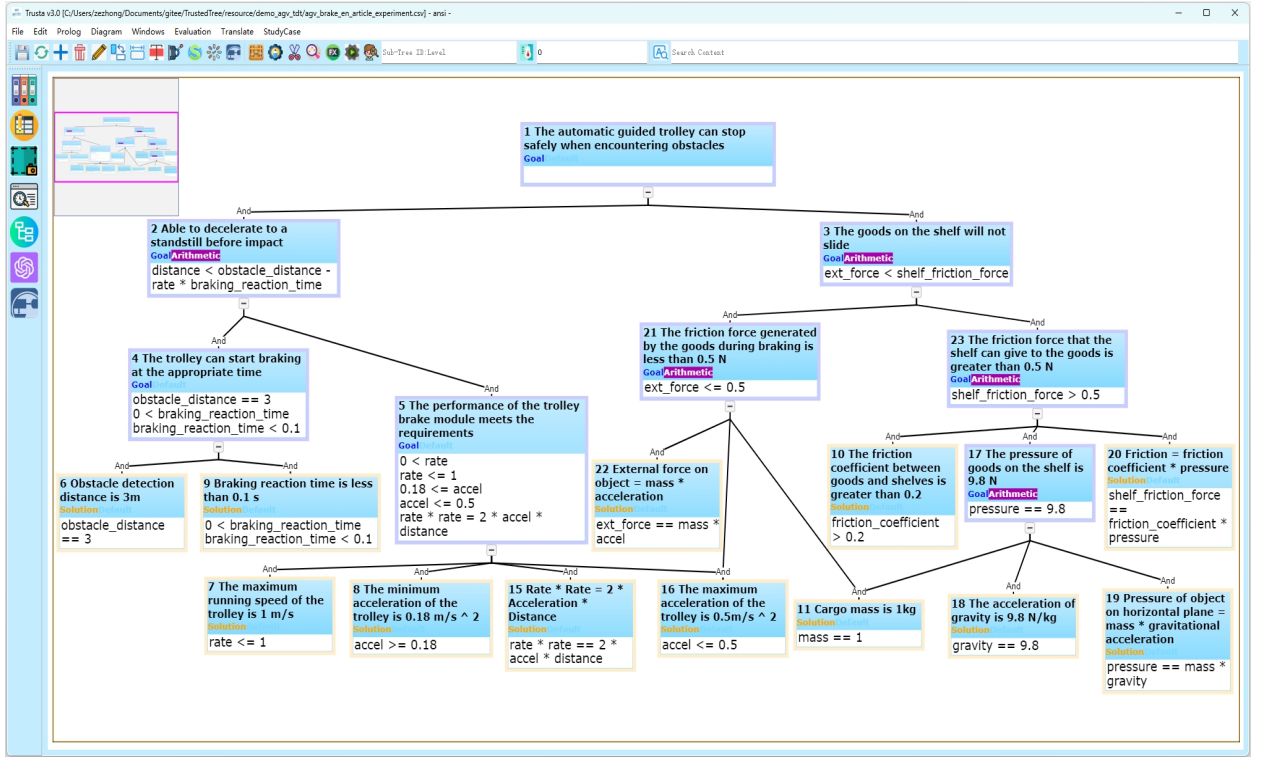


Figure 16: Final TDT for the AGV's automatic braking case: Illustration of the refined structure after adjustments, emphasizing the translation result of constraint expressions in the evaluation stage.

6.1. Evaluation of Claim Decomposition

6.1.1. Experimental Design

Objectives and Motivation This study evaluates Trusta's impact on ease of use, learning curve, training requirements, effort reduction, and issue discovery within safety engineering. We aim to demonstrate how Trusta, by integrating formal methods with large language models, streamlines the development and verification of assurance cases.

Participant Selection Participants were selected to represent Trusta's diverse user base, including safety engineering experts, software developers, and engineering graduate students. Table 3 details the participant demographics.

Table 3
Participant Demographics and Distribution for Trusta Evaluation Study

Participant Group	Description	Total Number	Without Trusta	With Trusta
Safety Engineering Experts	Professionals with extensive experience in safety engineering	20	10	10
Software Developers	Developers experienced in safety-critical applications	20	10	10
Graduate Students	Individuals with knowledge of formal methods but limited practical experience	20	10	10

Training Session Prior to commencing the experimental tasks, participants underwent a training session designed to equip them with the necessary skills to utilize Trusta effectively. This session included a one-hour presentation on Trusta's functionalities followed by a Q&A segment, ensuring all participants were proficient in using the tool.

Table 4

Comparison of semantic similarity between baseline assurance cases and fully automated Trusta-generated assurance case fragments. In the x - y format, the left number x indicates the similarity between the model-generated assurance case fragments and the baseline, while the right number y represents x as a percentage of α (S2S column). Specifically, $y > 80$ serves as the target for the human-based assurance case fragment reconstruction task. The relationship between x and y is given by the formula: $y = \frac{x}{S2S} \times 100$, where y is the percentage similarity, and S2S is the value from the S2S column.

ID	Domain	GPT-3.5	GPT-4	PaLM 2	S2S	ID	Domain	GPT-3.5	GPT-4	PaLM 2	S2S
1	UAV	6-8	22-26	35-42	85	32	Automobile	26-30	0-0	21-24	89
2	UAV	24-31	15-19	13-17	79	33	Automobile	1-2	5-7	7-9	83
3	UAV	0-0	11-14	32-39	83	34	Automobile	37-46	35-44	38-47	81
4	UAV	34-46	26-35	45-60	75	35	Automobile	20-30	41-62	36-54	67
5	UAV	15-25	23-39	41-69	60	36	Automobile	35-52	0-0	18-27	68
6	AutoRobot	46-80	48-83	25-44	58	37	Automobile	12-27	21-46	6-14	46
7	AutoRobot	33-43	37-49	60-78	77	38	Automobile	36-47	19-25	35-45	78
8	AutoRobot	10-24	11-26	0-0	43	39	Automobile	21-36	40-68	27-46	59
9	AutoRobot	21-31	0-0	33-49	68	40	Automobile	18-30	37-60	52-84	62
10	AutoRobot	32-46	38-55	53-76	70	41	Automobile	41-57	36-50	48-67	72
11	CubeSat	51-59	20-23	25-29	87	42	Pacemaker	0-0	0-0	13-17	79
12	CubeSat	2-4	3-5	8-13	63	43	Pacemaker	25-29	36-41	47-54	88
13	CubeSat	24-28	33-39	33-39	86	44	Pacemaker	23-35	15-23	56-84	67
14	CubeSat	14-16	31-36	32-37	88	45	Pacemaker	8-10	32-40	20-25	80
15	CubeSat	14-19	20-27	16-22	75	46	Pacemaker	37-47	56-71	25-32	79
16	CyberSecurity	32-39	15-18	19-23	84	47	Pacemaker	25-27	42-44	25-27	96
17	CyberSecurity	47-64	51-69	79-107	74	48	Pacemaker	33-48	42-60	20-29	70
18	CyberSecurity	40-52	37-48	86-111	78	49	Pacemaker	36-48	30-40	30-40	76
19	CyberSecurity	47-50	48-52	62-66	94	50	Pacemaker	44-71	32-52	31-50	62
20	CyberSecurity	29-44	29-44	22-34	66	51	Pacemaker	44-103	37-87	47-110	43
21	CyberSecurity	14-27	30-58	9-18	52	52	Aircraft	29-45	41-64	27-42	65
22	CyberSecurity	32-73	9-21	40-91	44	53	Aircraft	35-57	11-18	51-83	62
23	CyberSecurity	22-50	14-32	25-57	44	54	Aircraft	29-38	52-67	40-52	78
24	CyberSecurity	43-55	36-46	29-37	79	55	Aircraft	24-34	12-17	45-63	72
25	CyberSecurity	47-79	16-27	47-79	60	56	Aircraft	31-56	29-52	21-38	56
26	CyberSecurity	20-63	18-57	38-119	32	57	Aircraft	29-32	26-29	33-36	92
27	CyberSecurity	11-17	32-49	18-28	66	58	AGV	0-0	8-10	23-27	86
28	CyberSecurity	53-87	25-41	51-84	61	59	AGV	27-30	27-30	24-27	91
29	Automobile	15-20	1-2	0-0	78	60	AGV	22-27	20-25	59-72	82
30	Automobile	6-10	15-25	12-20	62	61	AGV	59-61	72-74	74-76	98
31	Automobile	22-46	6-13	37-78	48		AVERAGE	26-[39]	25-[37]	33-[48]	71

Experimental Tasks Participants were tasked with reconstructing three randomly selected assurance case snippets from a pool of 57 options. These snippets served as the baseline for similarity comparisons. These snippets were collected from the following papers covering seven domains: UAV (Unmanned Aerial Vehicle) [26], AutoRobot [31], CubeSat [25], CyberSecurity [42], Automobile [30], Pacemaker [34], and Aircraft [24].

While there are numerous methods for determining semantic similarity [60], we employ Baidu’s commercial short text similarity service [61], which is based on the ERNIE pre-trained language model [62]. In current similarity algorithms, only sentences that are completely identical, where every word and punctuation mark matches, are assigned a similarity of 100% (omitting the “%” sign for later references). Mathematically, let the baseline sentence be B . Then, we have

$$\begin{cases} \text{Sim}(B, B) = 100, \\ \text{Sim}(B, \text{S2S}(B)) = \alpha < 100, \\ \text{Mean}(B) = \text{Mean}(\text{S2S}(B)), \end{cases}$$

where $\text{S2S}(B)$ is the sentence obtained by applying a Sequence-to-Sequence (S2S) transformation to B . Since neither humans nor LLMs can precisely duplicate B word-for-word (and punctuation-for-punctuation) to achieve $\text{Sim}(B, B) = 100$, we instead employ the S2S approach to generate a sentence with the same meaning $\text{Mean}(B) = \text{Mean}(\text{S2S}(B))$ but a lower similarity score $\text{Sim}(B, \text{S2S}(B)) = \alpha < 100$. We then use α , rather than 100, as the target similarity for reconstructing assurance case snippets.

Specifically, we employ a two-step translation process to generate $S2S(B)$. First, we use the Baidu translation API [63] to translate the English sentence B into a Chinese sentence B' . Then, we translate B' back into English using the Youdao translation API [64], resulting in B'' . This ensures that $B'' \neq B$, as B' and B'' are likely to differ due to the use of two distinct translation systems.

Participants performed these tasks using both traditional methods (Trusta without LLMs—a graphical assurance case editor) and Trusta enhanced with LLMs. The tasks were considered complete when participants achieved at least 80% similarity to the S2S similarity. During the experiment, users edit node content within Trusta, and the tool recalculates similarity scores until they reach the desired threshold. The study focused on comparing the efficiency and quality of traditional methods against those facilitated by Trusta.

Data Collection Methods Quantitative data on task completion times and identified issues were recorded, alongside qualitative data from post-experiment surveys on Trusta’s usability and user satisfaction. This mixed-method approach aims to provide a comprehensive assessment of Trusta’s benefits in safety engineering workflows.

6.1.2. Experimental Results

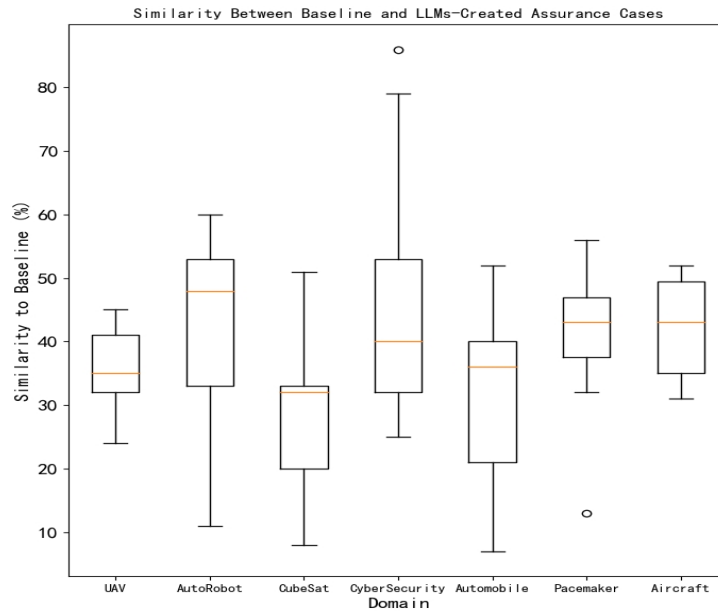


Figure 17: Comparative analysis of similarity between baseline assurance cases and Trusta-generated assurance cases across different domains. The domains examined include UAV (Unmanned Aerial Vehicle) [26], AutoRobot [31], CubeSat [25], CyberSecurity [42], Automobile [30], Pacemaker [34], and Aircraft [24]. The LLMs applied were ChatGPT-3.5, ChatGPT-4 and PaLM 2. Similarity measurement was performed using Baidu’s commercial text similarity service [61].

First, our experiments explored the effectiveness of Trusta and cutting-edge language models, such as ChatGPT-3.5, ChatGPT-4, and PaLM 2, in generating assurance case fragments across seven domains, including UAVs, AutoRobot, and CubeSat, among others. We quantified the semantic similarity between baseline assurance cases and Trusta-generated without human intervention, as shown in Table 4. Each row corresponds to a different assurance case fragment from a particular domain. The values in the cells provide the similarities calculated for each model (GPT-3.5, GPT-4, PaLM 2) for that specific fragment. The final S2S column shows the target percentage of similarity for human-based assurance case reconstruction. In the x - y format, the number x stands for the similarity between the model-generated assurance case fragments and the baseline, and the number y represents x as a percentage of α in the S2S column.

We found an average similarity range of 30%-50% (Figure 17), with instances exceeding 50% of similarity in 12 out of 57 fragments, and a maximum similarity of 86%. These results highlight the LLMs’ potential in producing accurate assurance case content. Although most subgoals initially generated by Trusta using general-purpose LLMs exhibit less than 50% similarity compared to the baseline data, Trusta still provides two key benefits: 1) It guides the direction

for subsequent human intervention. 2) It advances approximately 40% in the proposed decomposition direction, as illustrated in Figure 18. Figure 19 presents examples of assurance case fragments created by different models.

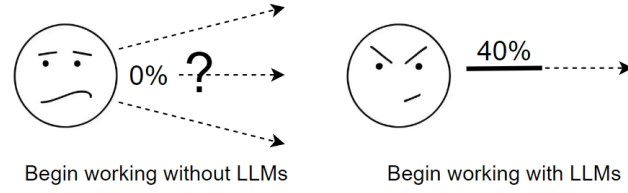


Figure 18: Trusta's role in guiding human intervention and enhancing decomposition progress.

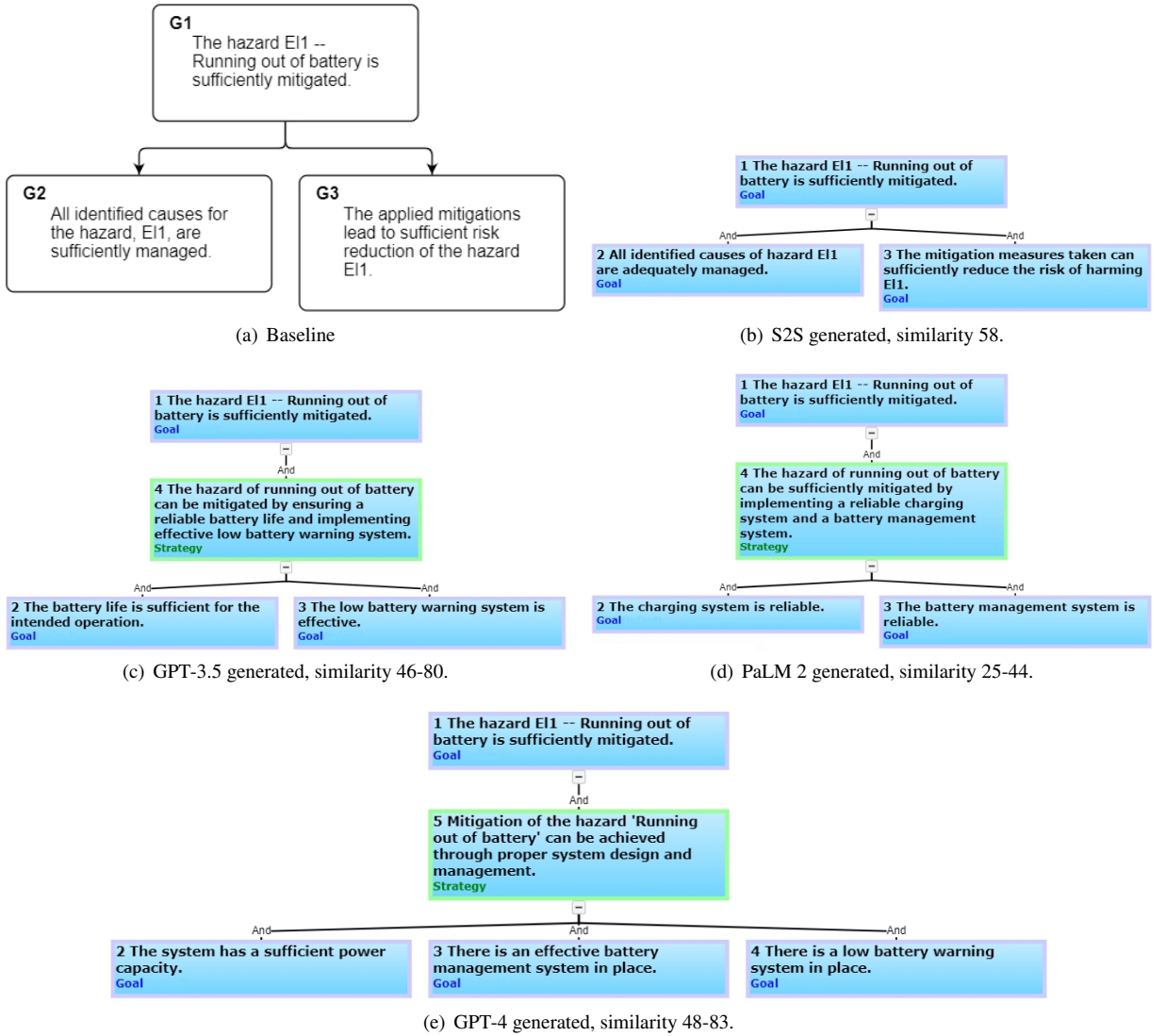


Figure 19: Examples of assurance case fragments generated by different models (ID=6 in Table 4).

Based on the data in Table 4, we present a heatmap of the automatic success rates across different domain-model combinations (Figure 20), which visually demonstrates the percentage of assurance cases generated by each model with $y > 80$ in various domains. Analysis of the success rate heatmap reveals significant performance differences across

models in different domains. PaLM 2 shows a notable advantage in complex system validation scenarios, with a success rate of 38.5% in the CyberSecurity domain, far outperforming other models. It also maintains a relative advantage in the Medical Device (Pacemaker, 20%) and Avionics (Aircraft, 16.7%) domains, while GPT-4 only achieves a near-optimal performance in the Autonomous Driving (AutoRobot, 20%) domain. Notably, 75% of domain-model combinations have an automatic success rate of less than 10%, indicating that automated generation technology still requires human verification in most safety-critical scenarios. This distribution suggests a correlation between model performance and domain complexity, implying that model selection should be tailored to the specific characteristics of the domain.

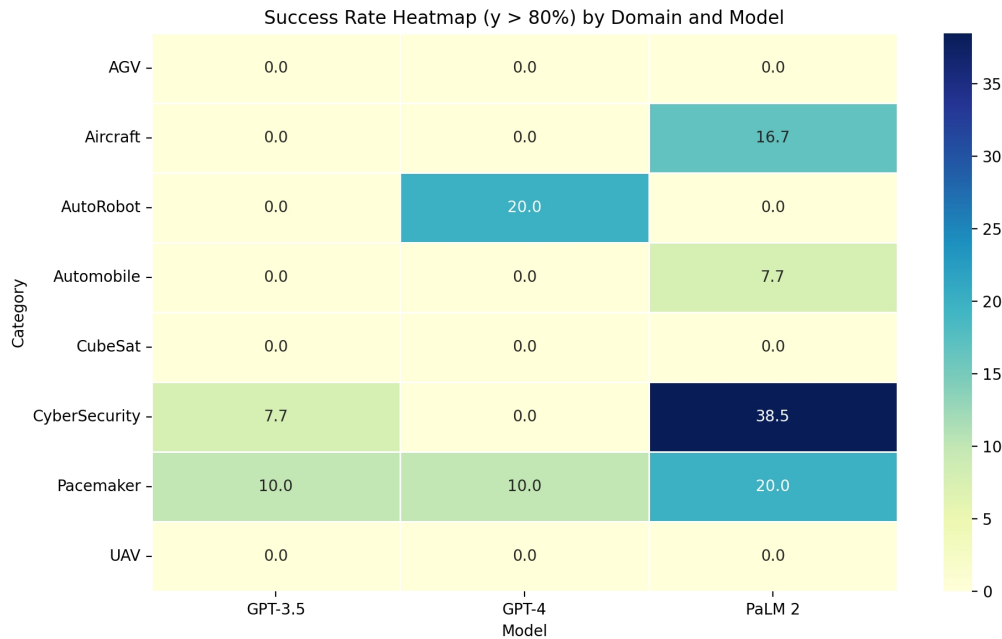


Figure 20: Heatmap of automatic success rates across domain-model combinations.

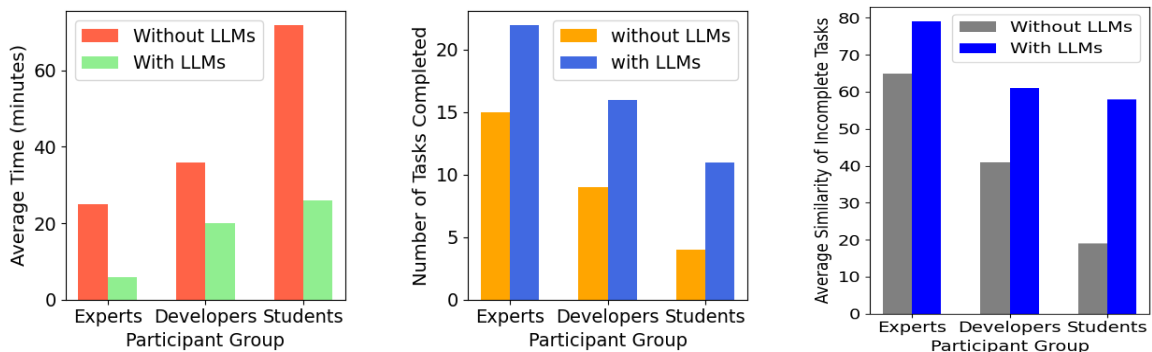


Figure 21: Experimental results demonstrate the impact of using LLMs in Trusta on task completion time, task accuracy, and average similarity scores. The results include those from three LLMs: ChatGPT-3.5, ChatGPT-4, and PaLM 2.

Then, human intervention further refined the initial assurance cases generated by Trusta with LLMs. Human improvements fall into three categories: adding content, modifying content, and deleting content. We have provided examples of human improvements in the AGV case to help readers better understand this step, as shown in Table 5. Assurance cases are represented in Prolog format as “C :- C1, C2.” with similarity provided at the end.

Table 5

Human refinements in assurance case automatic generation using Trusta.

Baseline	Trusta automatic generation	Human adjustment results	Human improvement
"The automatic guided trolley can stop safely when encountering obstacles" :- "Able to decelerate to a standstill before impact", "The goods on the shelf will not slide".	"The automatic guided trolley can stop safely when encountering obstacles" :- "The sensors detecting the presence of an obstacle are reliable", "The collision detection algorithm can process accurate detection data to send out prompt and accurate movement commands to the trolley". (23-27)	"The automatic guided trolley can stop safely when encountering obstacles" :- "The sensors can detect obstacles, allowing the trolley to slow to a standstill before impact", "The goods on the shelf remain stable". (70-81)	Considering the safety of the goods on the shelf. (adding content)
"Able to decelerate to a standstill before impact" :- "The trolley can start braking at the appropriate time", "The performance of the trolley brake module meets the requirements".	"Able to decelerate to a standstill before impact" :- "The braking system is effective and reliable", "Sensors accurately detect obstacles and measure distance", "Control algorithms ensure timely and appropriate deceleration". (27-30)	"Able to decelerate to a standstill before impact" :- "The trolley ensure timely and appropriate deceleration", "The braking module performance meets the requirements". (71-79)	The second and third goals are summarized. (deleting content)
"The goods on the shelf will not slide" :- "The friction force generated by the goods during braking is less than 0.5 N", "The friction force that the shelf can give to the goods is greater than 0.5 N".	"The goods on the shelf will not slide" :- "The coefficient of friction between the goods and the shelf is greater than or equal to 0.5". (59-72)	"The goods on the shelf will not slide" :- "The friction force generated by the goods during braking is greater than the friction force that the shelf can give to the goods". (92-112)	The direct effect of the friction coefficient is refined. (modifying content)
"The friction force that the shelf can give to the goods is greater than 0.5 N" :- "The friction coefficient between goods and shelves is greater than 0.2", "The pressure of goods on the shelf is 9.8 N", "Friction = friction coefficient * pressure".	"The friction force that the shelf can give to the goods is greater than 0.5 N" :- "The coefficient of friction between the shelf and the goods is greater than 0.5", "The normal force acting on the goods is greater than 1 N". (74-76)	"The friction force that the shelf can give to the goods is greater than 0.5 N" :- "The coefficient of friction between the shelf and the goods is greater than 0.5", "The pressure on the goods is greater than 1 N". (87-89)	Only the expression has been changed. (modifying content)

This approach assisted by LLM significantly reduced task completion times across various participant groups, demonstrating Trusta's effectiveness in streamlining safety engineering processes (Figure 21). Furthermore, users of Trusta with LLMs not only completed more assurance case construction tasks but also improved the average similarity of those that remained incomplete. This underscores the tool's ability to enhance both precision and quality. Notably, the performance of the student group using Trusta with LLMs approached that of the expert group working without LLM support.

User satisfaction with Trusta was high (Figure 22), though feedback highlighted areas for improvement, including adaptive scaling, GSN diagram aesthetics, and the need for progress indicators. These insights are crucial for Trusta's future development, emphasizing the importance of user-centric design for the tool.

6.1.3. Findings

Notably, Trusta's impact was most pronounced among Engineering Graduate Students, elevating their performance closer to that of Safety Engineering Experts. This suggests Trusta's potential to bridge expertise gaps, particularly benefiting those with less experience. For Safety Engineering Experts, Trusta provided valuable support by improving efficiency and enhancing capability. Software developers experienced intermediate benefits, reflecting Trusta's broad

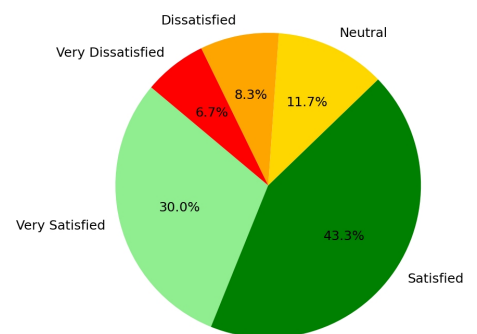


Figure 22: User satisfaction levels with Trusta usability.

applicability. Furthermore, although the content generated by LLMs often requires manual adjustments, using these generated materials as a starting point is far better than beginning from scratch, greatly reducing the mental burden on users.

6.2. Evaluation of Formalizing and Solving Constraints

6.2.1. Experimental Design

Objective Evaluate the effectiveness of the Trusta tool in translating natural language TDT node descriptions into formal constraint expressions.

Experimental Subjects Node contents from the AGV vehicle case study in Section 5, involving natural language descriptions of physical concepts like pressure, gravity, mass, friction, force, and acceleration.

Procedure

1. **Data Collection:** Gather for each node: 1) The initial formal expressions generated by Trusta. 2) The manually adjusted expressions.
2. **Manual Review:** Experts review and record the data before and after manual adjustments for each node using the evaluation metrics (Logical and Variable).
3. **Comparison and Analysis:** Compare Trusta's expressions with the manually corrected versions. Analyze differences in logical correctness and variable naming accuracy.

Evaluation Metrics

- **Logical Correctness:** Whether the expression accurately reflects the relationships described in the original natural language.
- **Variable Naming Accuracy:** Whether variable names are appropriate and consistent.

6.2.2. Experimental Results

Table 6 summarizes the node translations depicted in Figure 16, achieved by using Trusta with LLM (GPT-3.5) to convert natural language into constraint expressions. Human-adjusted expressions are provided for comparison. The check marks in the “Logical” column indicate that the LLM’s translations are logically consistent with human adjustment results. The check marks in the “Variable” column show that the variable names are contextually coherent and suitable for direct input into constraint solvers.

The manual adjustment of formal statements includes “variable adjustment” and “logical adjustment”. This step verifies that the description of each node aligns with its formal expression and ensures that, after running the solver for evaluation, the formal relationship between each node and its child nodes is logical and consistent.

Table 7 presents five typical examples from Table 6, along with suggested human adjustments. For the last two types, the natural language descriptions lack explicit constraint information, leading to translation failures by the LLM. In such cases, Boolean values can serve as formal representations for obvious or abstract claims, or manual modifications are necessary to ensure correct solver execution.

6.2.3. Findings

The study underscored the importance of careful parameter tuning, such as setting minimum deceleration values for AGVs to prevent collisions and considering the materials of goods and shelves to ensure stability. Trusta proved instrumental in this tuning process, enhancing our confidence in the safe operation of AGVs. During the generation of formal statements, safety engineers often lack familiarity with the specific syntax of formal expressions. When Trusta generates a draft, engineers can then make necessary modifications. This not only lowers the barrier for formalizing assurance cases but also improves efficiency. Trusta can also attempt to solve existing constraints, and the generated error reports can guide engineers in modifying the formal content or optimizing deficiencies in the assurance cases.

Table 6

Summary of node translations: comparing automated and manual approaches in constraint expression generation using GPT-3.5.

No.	Natural language	LLM translation	Logical	Variable	Human adjustment results
1	Pressure of object on horizontal plane = mass * gravitational acceleration	$pressure == mass * gravity$	✓	✓	$pressure == mass * gravity$
2	The acceleration of gravity is 9.8 N/kg	$gravity == 9.8$	✓	✓	$gravity == 9.8$
3	Cargo mass is 1kg	$cargo_mass == 1$	✓	✓	$mass == 1$
4	The pressure of goods on the shelf is 9.8 N	$pressure == 9.8$	✓	✓	$pressure == 9.8$
5	Friction = friction coefficient * pressure	$friction == friction_coef * pressure$	✓	✓	$shelf_friction_force == friction_coef * pressure$
6	The friction coefficient between goods and shelves is greater than 0.2	$friction_coef > 0.2$	✓	✓	$friction_coef > 0.2$
7	The friction force that the shelf can give to the goods is greater than 0.5 N	$shelf_friction_force > 0.5$	✓	✓	$shelf_friction_force > 0.5$
8	External force on object = mass * acceleration	$ext_force_obj == mass_obj * accel_obj$	✓	✓	$ext_force == mass * accel$
9	The maximum acceleration of the trolley is $0.5m/s^2$	$trolley_accel == 0.5$	✓	✓	$accel <= 0.5$
10	The friction force generated by the goods during braking is less than 0.5 N	$friction_force < 0.5$	✓	✓	$ext_force <= 0.5$
11	The goods on the shelf will not slide	$ext_force <= shelf_friction_force$	✓	✓	$ext_force < shelf_friction_force$
12	Rate * Rate = 2 * Acceleration * Distance	$rate * rate == 2 * accel * distance$	✓	✓	$rate * rate == 2 * accel * distance$
13	The minimum acceleration of the trolley is $0.18 m/s^2$	$trolley_accel >= 0.18$	✓	✓	$accel >= 0.18$
14	The maximum running speed of the trolley is 1 m/s	$trolley_speed <= 1$	✓	✓	$rate <= 1$
15	The performance of the trolley brake module meets the requirements	$rate <= 1$ and $accel >= 0.18$ and $rate * distance / (2 * accel) <= 0.5$	✓	✓	$0 < rate <= 1; 0.18 <= accel; accel <= 0.5; rate * rate == 2 * accel * distance$
16	Braking reaction time is less than 0.1 s	$braking_reaction_time < 0.1$	✓	✓	$0 < braking_reaction_time; braking_reaction_time < 0.1$
17	Obstacle detection distance is 3m	$obstacle_distance == 3$	✓	✓	$obstacle_distance == 3$
18	The trolley can start braking at the appropriate time	$trolley == start_braking$ and $braking_time < approp_time$	✓	✓	$obstacle_distance == 3; 0 < braking_reaction_time; braking_reaction_time < 0.1$
19	Able to decelerate to a standstill before impact	$final_velocity == 0$ and $deceleration_time < impact_time$	✓	✓	$distance < obstacle_distance - rate * braking_reaction_time$
20	The automatic guided trolley can stop safely when encountering obstacles	$trolley_mass * trolley_acceleration == ext_force + shelf_friction_force$	✓	✓	

Table 7

Representative examples from node translations using GPT-3.5.

Natural Language Statements (NL), LLM Translations, and Human Adjustments	Check	Suggestion
NL: The friction coefficient between goods and shelves is greater than 0.2 LLM: <code>friction_coefficient > 0.2</code> Human: <code>friction_coefficient > 0.2</code>	Logical: ✓ Variable: ✓	No human adjustment required.
NL: Cargo mass is 1kg LLM: <code>cargo_mass == 1</code> Human: <code>mass == 1</code>	Logical: ✓ Variable:	Variable name needs human adjustment.
NL: Braking reaction time is less than 0.1 s LLM: <code>braking_reaction_time < 0.1</code> Human: <code>0 < braking_reaction_time; braking_reaction_time < 0.1</code>	Logical: ✓ Variable: ✓	The logic needs human adjustment, or the natural language description is inaccurate and requires correction.
NL: Able to decelerate to a standstill before impact LLM: <code>final_velocity == 0 and deceleration_time < impact_time</code> Human: <code>distance < obstacle_distance - rate * braking_reaction_time</code>	Logical: ✓ Variable:	The natural language description is too abstract and requires manual creation of the formal expression.
NL: The automatic guided trolley can stop safely when encountering obstacles LLM: <code>trolley_mass * trolley_acceleration == ext_force + shelf_friction_force</code> Human:	Logical: ✓ Variable:	Cannot be precisely represented with formal constraints; use Boolean values to indicate correctness vaguely.

6.3. Threats to Validation

Although LLM-generated content can assist in creating and verifying assurance cases, several factors may affect the validity of our findings:

Hallucination in LLM-Generated Content LLMs may produce text that appears coherent but is factually incorrect, known as “hallucination.” We mitigate this by verifying LLM-generated constraint expressions using a solver, ensuring erroneous or contradictory statements are identified before integrating them into assurance cases.

Limited Scope of Domains and Participants We evaluated our approach using a small set of domains and participants (safety engineering experts, software developers, and graduate students). Future studies involving a broader range of domains (e.g., medical devices, autonomous vehicles) and additional stakeholders (e.g., product managers, system testers) would provide more comprehensive insights.

LLM Integration and Adaptability Trusta currently supports ChatGPT-3.5, ChatGPT-4, and PaLM 2, but its modular design allows integration with other or newer LLMs, including domain-specific ones. As LLM technologies evolve, our findings may change accordingly.

Potential Overlooked Threats Despite quantitative comparisons (Table 4) and constraint-based verification, further threats may remain, such as biases in training data, variations in participants’ domain expertise, or unanticipated limitations when scaling to more complex assurance cases. Addressing these in future work will help enhance the robustness of our approach.

7. Related Work

Several assurance case editors have been developed to support GSN [65–69]. They facilitate the development and maintenance of assurance cases. Some of them offer assurance case patterns for users to reuse existing assurance cases [66, 69]. Luo et al. [70] provided an excellent survey of assurance case tools and summarized a systematic process of assurance case assessment. They also developed a tool to facilitate human evaluation. Chowdhury et al. [71] proposed a set of rules that semi-formally define the structure and content of assurance cases. These rules guide the work of assurance cases developers and reviewers. Assurance cases developers are instructed to use a more rigorous approach to their arguments. External reviewers have a basic checklist that guides them in assessing the rigor of arguments. Maksimov et al. [72] surveyed ten assurance case tools with evaluation capabilities. These tools can examine both the structure and content of assurance cases. Structural checks include structural constraints, correctness, integrity checks, and user queries. Content checks include argument evaluation, evidence evaluation, evaluation tracking, evaluation report, and evaluation interaction. Different tools utilize different approaches for content checks such as type checking, Bayesian belief networks and Dempster-Shafer Theory.

Recent research has focused on integrating formal methods with assurance case development. The integration of formal proof into unified assurance cases using Isabelle/SACM demonstrates how formal methods can enhance the rigor and traceability of assurance cases [73]. The semantic analysis of assurance cases using s(CASP) shows how logical constraints can improve the consistency and integrity of assurance arguments [74]. The ForeMoSt approach leverages Lean theorem proving to ensure the correctness of safety arguments, facilitating the creation of rigorous and verifiable assurance cases [75]. The inspection rover case study illustrates how integrating formal verification with assurance can improve the reliability of both system-level and component-level properties [31]. Similar to Trusta, Resolute [76] is inspired by logic programming and accompanies claims with user-defined logical rules for formal analysis; however, it does not incorporate SMT solvers.

Trusta represents a significant advancement by combining assurance cases with LLMs. This integration automates the extraction and interpretation of formal constraints from natural language texts and facilitates the decomposition of claims into more easily provable sub-claims. Trusta’s real-world application has demonstrated its ability to uncover nuanced issues often overlooked in manual inspections, thereby improving quality, increasing efficiency, and reducing the burden on safety engineers, thus enhancing the overall assurance case development workflow.

8. Conclusion and Future Work

We have presented Trusta, a tool that supports safety modeling and semi-automated validation, along with a detailed report on safety vulnerabilities. The TDTs created by this tool can be adapted from assurance cases by adding formal expressions, which can be used by constraint solvers to perform formal reasoning. With the integration of large language models, Trusta also brings convenience in creating safety cases, and assists users in translating natural language into constraint expressions, streamlining the overall process. In fact, within the Trusta tool, TDT and traditional GSN can be mutually converted. It can be observed that, without losing any information, the TDT representation is more compact, emphasizing key points, making it more easily readable. In assessing Trusta’s usability, it markedly improves assurance case creation for all users, especially aiding novices by improving efficiency and quality, highlighting its value in addressing the expertise gap. Our experiments with more than a dozen industrial cases show that Trusta is helpful to identify issues that are easily overlooked by manual inspection. In summary, a comprehensive analysis of the experimental results indicates that Trusta can reduce human costs. By using Trusta to complete tasks related to creating assurance cases, the overall human time costs were reduced by over 50%. Moreover, hiring senior safety experts is challenging; Trusta enables lower-level safety engineers to accomplish higher-level tasks. Trusta proves to be a potent tool in the construction and evaluation of assurance cases, offering substantial improvements in accuracy, efficiency, and quality across diverse user groups within safety-critical engineering domains.

Looking forward to the future development of Trusta, several promising directions emerge. First, there is an opportunity to trial and compare various large language models to discern the most effective ones for specific tasks among a few assurance cases. Such comparative studies may pave the way for nuanced insights and enhanced efficiencies. Second, by integrating more theoretical knowledge, we can optimize prompt words to guide the models more effectively, harnessing their potential in a more targeted manner. Third, the fine-tuning of these large language models to tailor their performance in specialized tasks is an exciting avenue for research. By customizing these models to the unique requirements of the safety domain, we anticipate significant advancements in their applicability and accuracy. Finally, the integration and development of additional formal languages within Trusta will broaden the

horizons of automatic reasoning within TDTs, making it more versatile and universally applicable. These future endeavors signal a robust pathway towards more comprehensive, adaptable, and intelligent safety modeling and validation.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grant Nos. 62072176 and 62472175, Shanghai Trusted Industry Internet Software Collaborative Innovation Center, and the “Digital Silk Road” Shanghai International Joint Lab of Trustworthy Intelligent Software under Grant No. 22510750100.

References

- [1] Peter Bishop and Robin Bloomfield. A methodology for safety case development. In *In Proceedings of the sixth safety-critical systems symposium*, 1998.
- [2] Robin Bloomfield and Peter Bishop. Safety and assurance cases: Past, present and possible future—an adelard perspective. In *In Proceedings of the Making Systems Safer*, pages 51–67. Springer London, 2009.
- [3] ISO/IEC 15026. Systems and software engineering-systems and software assurance-Part 2: Assurance case, 2011. <https://www.iso.org/standard/52926.html>.
- [4] ISO 26262-1:2018. Road vehicles-functional safety part 1: Vocabulary, 2018. <https://www.iso.org/standard/68383.html>.
- [5] DO-178C. Software considerations in airborne systems and equipment certification, 2011. <https://www.do178.org/>.
- [6] David J Rinehart, John C Knight, and Jonathan Rowanhill. *Current practices in constructing and evaluating assurance cases with applications to aviation*. National Aeronautics and Space Administration, Langley Research Center, 2015.
- [7] David J Rinehart, John C Knight, and Jonathan Rowanhill. Understanding what it means for assurance cases to “work”. Technical report, 2017.
- [8] Robert Lewis. Safety case development as an information modelling problem. In *Safety-Critical Systems: Problems, Process and Practice*, pages 183–193. Springer, 2009.
- [9] Yuxin Deng, Zezhong Chen, Wenjie Du, Bifei Mao, Zhizhang Liang, Qiushi Lin, and Jinghui Li. Trustworthiness derivation tree: A model of evidence-based software trustworthiness. In *In Proceedings of the 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 487–493. IEEE, 2021.
- [10] Tim Kelly and Rob Weaver. The goal structuring notation—a safety argument notation. In *In Proceedings of the Dependable Systems and Networks 2004 Workshop on Assurance Cases*. Citeseer, 2004.
- [11] The Assurance Case Working Group. Goal structuring notation community standard version 3, 2021. <https://scsc.uk/r141C:1?t=1>.
- [12] Kateryna Netkachova, Oleksandr Netkachov, and Robin Bloomfield. Tool support for assurance case building blocks. In *In Proceedings of the International Conference on Computer Safety, Reliability, and Security*, pages 62–71. Springer, 2014.
- [13] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *In Proceedings of the International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.
- [14] Nils Klarlund and Anders Møller. *Mona version 1.4: User manual*. BRICS, Department of Computer Science, University of Aarhus Denmark, 2001.
- [15] OpenAI. GPT-3.5 documentation, 2023. <https://platform.openai.com/docs/models/gpt-3-5>.
- [16] OpenAI. GPT-4 documentation, 2023. <https://platform.openai.com/docs/models/gpt-4>.
- [17] Google. Introducing PaLM 2, 2023. <https://ai.google/discover/palm2/>.
- [18] Tim Kelly. *Arguing safety: a systematic approach to managing safety cases*. Phd thesis, University of York, Heslington, York, England, 1999.
- [19] Stephen Toulmin. *The Uses of Argument*. Cambridge university press, England, 2003.
- [20] George Cleland, Mark-Alexander Sujun, Ibrahim Habli, and John Medhurst. *Evidence: using safety cases in industry and healthcare*. The Health Foundation, 2012.
- [21] Vladimir Sklyar and Vyacheslav Kharchenko. Assurance case for safety and security implementation: a survey of applications. *International Journal of Computing*, 19(4):610–619, 2020.
- [22] Tim Kelly, Iain Bate, John McDermid, and Alan Burns. Building a preliminary safety case: An example from aerospace. In *In Proceedings of the Australian Workshop on Industrial Experience with Safety Critical Systems and Software*, pages 1–10. Not available, 1997.
- [23] John Rushby, Xidong Xu, Murali Rangarajan, and Thomas Weaver. Understanding and evaluating assurance cases. Technical report, NASA Langley Research Center, 2015.
- [24] Patrick Graydon, John Knight, and Elisabeth Strunk. Assurance based development of critical systems. In *In Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 347–357. IEEE, 2007.
- [25] Rebekah Austin, Nagabhushan Mahadevan, Brian Sierawski, Gabor Karsai, Arthur Witulski, and John Evans. A cubesat-payload radiation-reliability assurance case using goal structuring notation. In *In Proceedings of the 2017 Annual Reliability and Maintainability Symposium*, pages 1–8. IEEE, 2017.
- [26] Michael Vierhauser, Sean Bayley, Jane Wyngaard, Wandi Xiong, Jinghui Cheng, Joshua Huseman, Robyn Lutz, and Jane Cleland-Huang. Interlocking safety cases for unmanned autonomous systems in shared airspaces. *IEEE transactions on software engineering*, 47(5):899–918, 2019.
- [27] John Medhurst and David Embrey. Safety case use in the railway industry. In *Supplements to: Using safety cases in industry and healthcare*, pages 65–74. The Health Foundation, London, 2012.

- [28] Julie Beugin, Cyril Legrand, Juliette Marais, Marion Berbineau, and El-Miloudi El-Koursi. Safety appraisal of gnss-based localization systems used in train spacing control. *IEEE Access*, 6:9898–9916, 2018.
- [29] Gerhard Griessnig and Adam Schnellbach. Development of the 2nd edition of the iso 26262. In *In Proceedings of the Systems, Software and Services Process Improvement*, pages 535–546. Springer International Publishing, 2017.
- [30] Robert Palin and Ibrahim Habli. Assurance of automotive safety—a safety case approach. In *In Proceedings of the Computer Safety, Reliability, and Security*, pages 82–96. Springer Berlin Heidelberg, 2010.
- [31] Hamza Bourbough, Marie Farrell, Anastasia Mavridou, Irfan Slijivo, Guillaume Brat, Louise Dennis, and Michael Fisher. Integrating formal verification and assurance: an inspection rover case study. In *In Proceedings of the NASA Formal Methods*, pages 53–71. Springer International Publishing, 2021.
- [32] Robin Bloomfield, Nick Chozos, George Cleland, and LLP Adelard. Safety case use within the medical devices industry. In *Supplements to: Using safety cases in industry and healthcare*, pages 75–91. The Health Foundation, London, 2012.
- [33] Brian Larson, John Hatcliff, and Patrice Chalin. Open source patient-controlled analgesic pump requirements documentation. In *In Proceedings of the 5th International Workshop on Software Engineering in Health Care*, pages 28–34. IEEE, 2013.
- [34] Eunkyong Jee, Insup Lee, and Oleg Sokolsky. Assurance cases in model-driven development of the pacemaker software. In *In Proceedings of the Leveraging Applications of Formal Methods, Verification, and Validation*, pages 343–356. Springer Berlin Heidelberg, 2010.
- [35] Nancy Leveson. The use of safety cases in certification and regulation. Technical Report Technical Report ESD-WP-2011-13, Massachusetts Institute of Technology Engineering Systems Division, 2011.
- [36] Alan Wassyng, Tom Maibaum, Mark Lawford, and Hans Bherer. Software certification: Is there a case against safety cases? In *In Proceedings of the Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, pages 206–227. Springer Berlin Heidelberg, 2011.
- [37] Jamie Henderson. Safety case use in the petrochemical industry. In *Supplements to: Using safety cases in industry and healthcare*, pages 55–64. The Health Foundation, London, 2012.
- [38] Michael Baram. Preventing accidents in offshore oil and gas operations: the us approach and some contrasting features of the norwegian approach. Technical report, Boston University School of Law, 2010.
- [39] Pietro Mendes, Jeremy Hall, Stelvia Matos, and Bruno Silvestre. Reforming Brazil’s offshore oil and gas safety regulatory framework: Lessons from norway, the united kingdom and the united states. *Energy Policy*, 74:443–453, 2014.
- [40] Tim Kelly. Safety case use in the defence industry. In *Supplements to: Using safety cases in industry and healthcare*, pages 19–23. The Health Foundation, London, 2012.
- [41] Bob Duncan and Mark Whittington. Compliance with standards, assurance and audit: does this equal security? In *In Proceedings of the 7th International Conference on Security of Information and Networks*, pages 77–84. Association for Computing Machinery, 2014.
- [42] Robin Bloomfield, Peter Bishop, Eoin Butler, and Kate Netkachova. Using an assurance case framework to develop security strategy and policies. In *In Proceedings of the Computer Safety, Reliability, and Security*, pages 27–38. Springer International Publishing, 2017.
- [43] Evi Widowati, Adi Sutomo, and Wahyudi Istiono. Are elementary schools ready for disaster preparedness and safety? *E3S Web Conf.*, 317:1–13, 2021.
- [44] Ibrahim Habli, Rob Alexander, Richard Hawkins, Mark Sujan, John McDermid, Chiara Picardi, and Tom Lawton. Enhancing covid-19 decision-making by creating an assurance case for simulation models. *arXiv preprint arXiv:2005.08381*, 2020.
- [45] Tim Kelly. A systematic approach to safety case management. *Journal of Passenger Cars: Electronic and Electrical Systems*, 113(7):257–266, 2004.
- [46] Robin Bloomfield and John Rushby. Assurance 2.0: A manifesto. *arXiv preprint arXiv:2004.10474*, 2020.
- [47] Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In *In Proceedings of the International Conference on Computer Aided Verification*. Springer, 2023.
- [48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [49] Joxan Jaffar and Michael J Maher. Constraint logic programming: A survey. *The journal of logic programming*, 19:503–581, 1994.
- [50] Atoosa Kasirzadeh, Mohammed Saddoune, and François Soumis. Airline crew scheduling: models, algorithms, and data sets. *EURO Journal on Transportation and Logistics*, 6(2):111–137, 2017.
- [51] Joshua Willman. Overview of PyQt5. In *Modern PyQt*, pages 1–42. Springer, 2021.
- [52] Farrukh Shahzad, Tarek R Sheltami, Elhadi M Shakshuki, and Omar Shaikh. A review of latest web tools and libraries for state-of-the-art visualization. *Procedia Computer Science*, 98:100–106, 2016.
- [53] Francesca Rossi, Peter Van Beek, and Toby Walsh. Constraint programming. *Foundations of Artificial Intelligence*, 3:181–211, 2008.
- [54] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*, 2023.
- [55] Sabit Ekin. Prompt engineering for chatgpt: a quick guide to techniques, tips, and best practices. *Authorea Preprints*, 2023.
- [56] Ggaliwango Marvin, Nakayiza Hellen, Daudi Jjingo, and Joyce Nakatumba-Nabende. Prompt engineering in large language models. In *In Proceedings of the International Conference on Data Intelligence and Cognitive Informatics*, pages 387–402. Springer, 2023.
- [57] Michael Desmond and Michelle Brachman. Exploring prompt engineering practices in the enterprise. *arXiv preprint arXiv:2403.08950*, 2024.
- [58] Boqi Chen, Kua Chen, Shabnam Hassani, Yujing Yang, Daniel Amyot, Lysanne Lessard, Gunter Mussbacher, Mehrdad Sabetzadeh, and Dániel Varró. On the use of gpt-4 for creating goal models: an exploratory study. In *In Proceedings of the 31st International Requirements Engineering Conference Workshops*, pages 262–271. IEEE, 2023.
- [59] OpenAI. Create chat completion, 2023. <https://platform.openai.com/docs/api-reference/chat>.
- [60] Dhivyaa Chandrasekaran and Vijay Mago. Evolution of semantic similarity—a survey. *ACM Computing Surveys (CSUR)*, 54(2):1–37, 2021.
- [61] Baidu. Short text similarity algorithm - baidu ai open platform, 2024. https://ai.baidu.com/tech/nlp_basic/simnet.

- [62] Yu Sun, Shuohuan Wang, Shikun Feng, Siyu Ding, Chao Pang, Junyuan Shang, Jiaxiang Liu, Xuyi Chen, Yanbin Zhao, Yuxiang Lu, et al. Ernie 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation. *arXiv preprint arXiv:2107.02137*, 2021.
- [63] Baidu. Baidu text translation api, 2024.
- [64] Youdao. Youdao translation api, 2024.
- [65] Ewen Denney, Ganesh Pai, and Josef Pohl. Advocate: An assurance case automation toolset. In *In Proceedings of the International Conference on Computer Safety, Reliability, and Security*, pages 8–21. Springer, 2012.
- [66] Yutaka Matsuno. D-case editor: A typed assurance case editor. *University of Tokyo*, 2011.
- [67] George Despotou, Aris Apostolakis, and Dimitris Kolovos. Acedit, 2016. <https://code.google.com/p/acedit/>.
- [68] NASA. Certware, 2016. <http://nasa.github.io/CertWare/>.
- [69] Sebastian Voss, Bernhard Schätz, Maged Khalil, and Carmen Carlan. Towards modular certification using integrated model-based safety cases. In *In Proceedings of the Proc. VeriSure: Verification and Assurance Workshop*, 2013.
- [70] Yaping Luo, Mark van den Brand, Zhuoao Li, and Arash Khabbaz Saberi. A systematic approach and tool support for GSN-based safety case assessment. *Journal of Systems Architecture*, 76:1–16, 2017.
- [71] Thomas Chowdhury, Alan Wassyng, Richard F Paige, and Mark Lawford. Systematic evaluation of (safety) assurance cases. In *In Proceedings of the International Conference on Computer Safety, Reliability, and Security*, pages 18–33. Springer, 2020.
- [72] Mike Maksimov, Sahar Kokaly, and Marsha Chechik. A survey of tool-supported assurance case assessment techniques. *ACM Computing Surveys*, 52(5):1–34, 2019.
- [73] Simon Foster, Yakoub Nemouchi, Mario Gleirscher, Ran Wei, and Tim Kelly. Integration of formal proof into unified assurance cases with isabelle/sacm. *Formal Aspects of Computing*, 33(6):855–884, 2021.
- [74] Anitha Murugesan, Isaac Hong Wong, Robert Stroud, Joaquín Arias, Elmer Salazar, Gopal Gupta, Robin Bloomfield, Srivatsan Varadarajan, and John Rushby. Semantic analysis of assurance cases using s (casp). In *In Proceedings of the To appear at Goal Directed Execution of Answer Set Programs (GDE) Workshop in Int'l Conf. on Logic Programming (ICLP)*, 2023.
- [75] Torin Viger, Logan Murphy, Alessio Di Sandro, Claudio Menghi, Ramy Shahin, and Marsha Chechik. The foremost approach to building valid model-based safety arguments. *Software and Systems Modeling*, 22(5):1473–1494, 2023.
- [76] Michael Whalen and Andrew Gacek. Resolute, 2016. <https://github.com/smaccm/smaccm/>.

A. Complete Prompt Words and Explanatory Notes

LLMs excel in a broad spectrum of tasks, requiring simple “prompts” for operation. However, maximizing their potential involves expert prompt crafting.

The invocation of a large language model, particularly for complex tasks like assurance case generation, requires carefully crafted prompts, with the complete content of the prompts shown in Listing 7.

Listing 7: Prompt with domain knowledge of assurance case.

```
1 You are an expert proficient in the Assurance Case.
2 Your answers always need to follow the following output format and you always have to try to provide a
  set of sub-goals. You may repeat your answers.
3 Break down the following goal into several sub-goals, these sub-goals should be able to support the
  parent goal, and explain the reasoning behind the breakdown. Finally, provide solutions that support
  these sub-goals.
4
5 Goal(claim): A goal is a claim in the argument, usually supported by sub-goals(sub-claims), strategies(
  arguments) or solutions(evidences). Goals describe assertions about system characteristics, performance,
  safety, etc.
6 Strategy(argument): A strategy describes the reasoning relationship between a goal and its supporting
  goals. Strategies clarify how to satisfy a higher-level goal through sub-goals, solutions, or other
  evidence.
7 Solution(evidence): A solution provides references to evidence items. Evidence can be experimental data,
  historical records, analytical reports, simulation results, or other materials supporting the argument.
8
9 The five basic CAE(claim-argument-evidence) building blocks that we have identified are:
10 1. Decomposition: partitions some aspect of the claim.
11 2. Substitution: refines a claim about an object into another claim about an equivalent object.
12 3. Concretion: gives a more precise definition to some aspect of the claim.
13 4. Calculation or proof: used when some value of the claim can be computed or proved.
14 5. Evidence incorporation: incorporates evidence that directly supports the claim.
15 In practice, some of the basic blocks are often merged together into composite blocks.
16
17 Goal G1: The self-driving car is safe to operate on public roads.
18 Building Blocks: Decomposition
19 Break down Strategy: {"strategy": "The safety of the self-driving car can be determined by examining its
  hardware and software components."}
20 Sub-goals dictionary: {"G1.1": "The sensor system is reliable.", "G1.2": "The navigation algorithm is
  accurate.", "G1.3": "The emergency systems function correctly."}
21 Solutions dictionary: {"Sn1.1": "Manufacturer test reports, third-party evaluations.", "Sn1.2": "Code
  audits, simulation results.", "Sn1.3": "Test scenarios, independent assessments."}
22 Explanation: G1.1, G1.2 and G1.3 can support G1, Sn1.1 can support G1.1, Sn1.2 can support G1.2, Sn1.3
  can support G1.3.
23 FINISH
24
25 Goal G1: The new version of the medical device is safe.
26 Building Blocks: Substitution
27 Break down Strategy: {"strategy": "The new version is equivalent to the old version in terms of safety
  features."}
28 Sub-goals dictionary: {"G1.1": "The old version of the medical device is safe."}
29 Solutions dictionary: {"Sn1.1": "Prior safety certification for the old version, documentation showing
  equivalence of safety features between old and new versions."}
30 Explanation: G1.1 can support G1, Sn1.1 can support G1.1.
31 FINISH
32
33 Goal G1: The banking app's data encryption is secure.
34 Building Blocks: Evidence Incorporation
35 Break down Strategy: {"strategy": "The encryption algorithm used is up to industry standards."}
36 Sub-goals dictionary: {"G1.1": "The app uses AES-256 encryption."}
37 Solutions dictionary: {"Sn1.1": "Code review reports, security assessments confirming AES-256 encryption
  ."}
38 Explanation: G1.1 can support G1, Sn1.1 can support G1.1.
39 FINISH
40
41 Goal G1: The airline's booking system is highly reliable.
42 Building Blocks: Concretion
43 Break down Strategy: {"strategy": "'High reliability' in this context means 99.99% uptime."}
44 Sub-goals dictionary: {"G1.1": "The system has 99.99% uptime."}
45 Solutions dictionary: {"Sn1.1": "System logs, third-party uptime monitoring reports."}
46 Explanation: G1.1 can support G1, Sn1.1 can support G1.1.
47 FINISH
48
49 Goal G1: The emergency hotline's average response time is less than 30 seconds.
50 Building Blocks: Calculation
51 Break down Strategy: {"strategy": "The average response time can be calculated from the time taken to
  pick up the call and the time taken to dispatch emergency services."}
```

```

52 Sub-goals dictionary: {"G1.1": "The time taken to pick up the call is less than 10 seconds.", "G1.2": "
The time taken to dispatch emergency services is less than 20 seconds."}
53 Solutions dictionary: {"Sn1.1": "Call logs, staff rosters.", "Sn1.2": "Dispatch records, GPS logs from
emergency vehicles."}
54 Explanation: G1.1 and G1.2 can support G1, Sn1.1 can support G1.1, Sn1.2 can support G1.2.
55 FINISH
56
57 Goal G1: <A_NEW_GOAL>
58 Building Blocks:
59 Break down Strategy:
60 Solutions dictionary:
61 Explanation:
62 FINISH

```

We referenced the workflow of nl2spec and have adapted the original prompt words designed for translating temporal logics, as shown in Listing 8, to facilitate the conversion of natural language into constraint expressions.

Listing 8: Prompt of LLM translation from natural language to constraint expressions.

```

1 You are an expert proficient in the Z3 constraint solver and the Python language.
2 Your answers always need to follow the following output format and you always have to try to provide a
constraint formula. You may repeat your answers.
3 Translate the following natural language sentences into a constraint formula and explain your
translation step by step.
4
5 Remember that + means "Addition", - means "Subtraction", * means "Multiplication", / means "Division",
// means "Integer Division", % means "Modulus", ** means "Exponentiation", > means "greater than", <
means "less than", == means "equal to", >= means "greater than or equal to", <= means "less than or
equal to", != means "not equal to", And(x, y) means "x and y", Or(x, y) means "x or y", Not(x) means "
not x".
6
7 The formula should only contain variables, numbers or operators +, -, *, /, //, %, **, >, <, ==, >=, <=,
!=, And, Or, Not.
8
9 Using the International System of Units (SI) to standardize the units of numerical quantities. For
example, When we describe distance or length, we typically use meters (m) as the unit. When we measure
mass, we use kilograms (kg). Time is usually measured in seconds (s). Speed can be described in meters
per second (m/s). When we talk about the magnitude of force, we use newtons (N, defined as kg*m/s^2).
10
11 Natural Language: The maximum running speed of the trolley is 1 m/s.
12 Given translations: {}
13 Explanation: "speed of the trolley" from the input translates to the variable trolley_speed. "maximum
running" from the input translates to the operators <=. "1 m/s" translates to a number 1.
14 Explanation dictionary: {"speed of the trolley" : "trolley_speed", "maximum running" : "<=", "1 m/s" :
"1"}
15 So the final constraint translation is: trolley_speed <= 1.FINISH
16
17 Natural Language: Fan 2 speed range 1900~2200 rpm.
18 Given translations: {}
19 Explanation: "Fan 2 speed" from the input translates to the variable fan_2_speed. "1900~" from the input
translates to the subformula 1900 <=. "~2200 rpm" from the input translates to the subformula <= 2200,
so "range 1900~2200 rpm" translates to a formula And(1900 <= fan_2_speed, fan_2_speed <= 2200).
20 Explanation dictionary: {"Fan 2 speed" : "fan_2_speed", "1900~" : "1900 <=", "~2200 rpm" : "<= 2200", "
range 1900~2200 rpm" : "And(1900 <= fan_2_speed, fan_2_speed <= 2200)"}
21 So the final constraint translation is: And(1900 <= fan_2_speed, fan_2_speed <= 2200).FINISH
22
23 Natural Language: Response time = processing time + message sending time.
24 Given translations: {}
25 Explanation: "Response time" from the input translates to the variable response_time. "=" from the input
translates to the operator ==. "processing time" from the input translates to the variable
proc_all_time. "message sending time" translates to a variable send_time.
26 Explanation dictionary: {"Response time" : "response_time", "=" : "==", "processing time" : "
proc_all_time", "message sending time" : "send_time"}
27 So the final constraint translation is: response_time == proc_all_time + send_time.FINISH
28
29 Natural Language: The pressure of goods on the shelf is 9.8 N.
30 Given translations: {"9.8 N": "9.8", "The pressure of goods on the shelf": "pressure", "is": "="}
31 Explanation:

```

Listing 9: Output of large language model translation from natural language to constraint expressions.

```

1 "The pressure of goods on the shelf" from the input translates to the variable pressure. "is" from the
input translates to the operator ==. "9.8 N" translates to a number 9.8.
2 Explanation dictionary: {"The pressure of goods on the shelf" : "pressure", "is" : "=", "9.8 N" :
"9.8"}
3 So the final constraint translation is: pressure == 9.8.

```

We have revised the introduction of the problem context to focus on constraint expression considerations (lines 1-3). Symbol conventions have been adjusted to align with comprehensible notations for constraint solvers (lines 5-7). A novel provision regarding numeric units has been introduced, mandating a standardized adoption of international units (line 9). Furthermore, we present three illustrative examples of constraint translation challenges (lines 11-27). Conclusively, we furnish pending translations that encompass both natural language and manually generated sub-translation cues (lines 29-31). This framework is seamlessly extended by a large language model, adhering to the format of the provided examples, as demonstrated in Listing 9. The approach of using large language models to translate natural language into constraint expressions, and then solving the answer with a constraint solver, employed technique T16.

B. Detailed Case Study

In Appendix B, Figures 23, 24, and 25 provide a deeper view of the decomposition process for the AGV's obstacle detection and safety control mechanisms. Figure 23 presents further breakdowns of the sensor system's capabilities, ensuring timely and accurate obstacle detection. Figure 24 focuses on the braking system, detailing how it can respond to sensor inputs and initiate braking actions swiftly and safely. Finally, Figure 25 illustrates the control system's decision-making process, showing how it implements safety strategies, including deceleration and stopping, to mitigate collision risks. These figures collectively offer a comprehensive view of the AGV's goal decomposition and the solutions needed to ensure safe and efficient operations in complex environments.

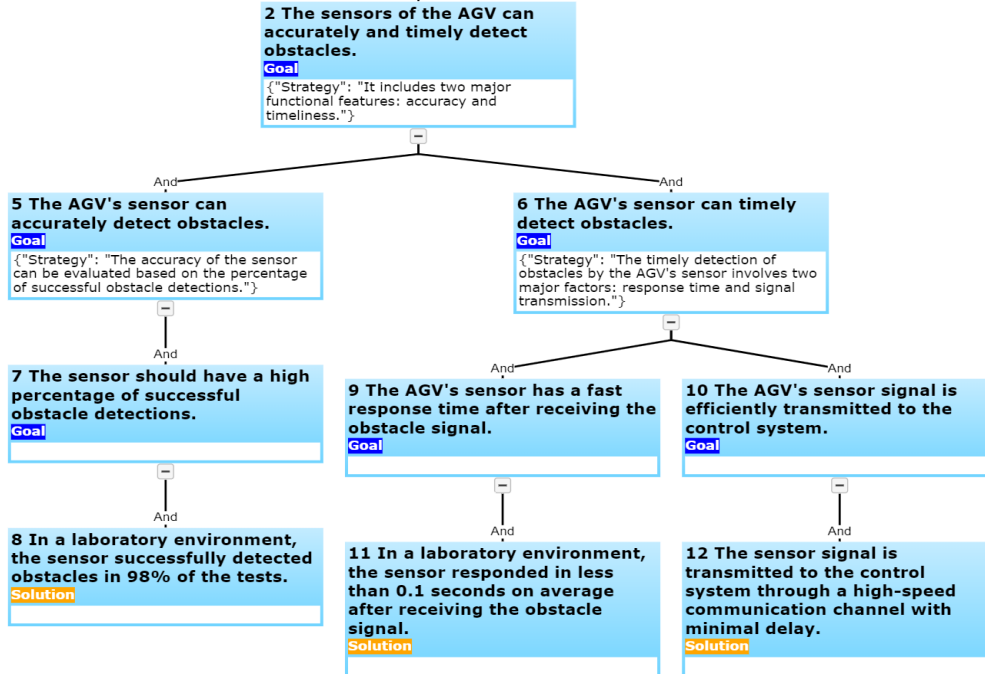


Figure 23: Second-level node decomposition: The sensors of the AGV can accurately and timely detect obstacles.

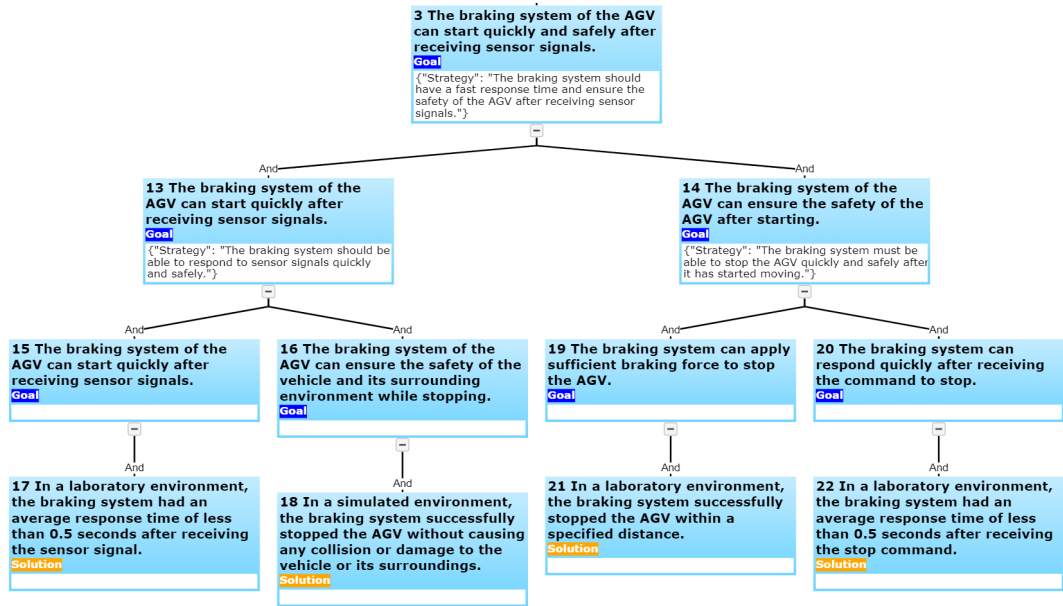


Figure 24: Second-level node decomposition: The braking system of the AGV can start quickly and safely after receiving sensor signals.

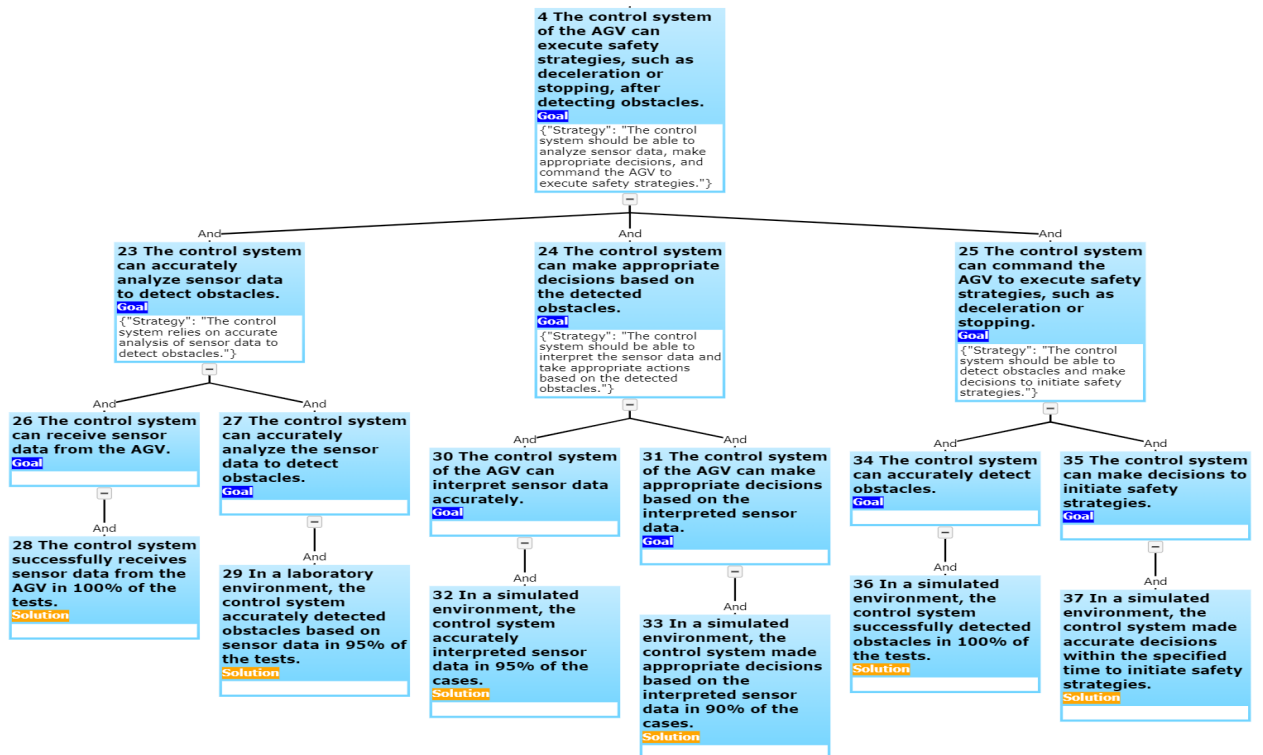


Figure 25: Second-level node decomposition: The control system of the AGV can execute safety strategies, such as deceleration or stopping, after detecting obstacles.