Automatically Detecting Fallacies in System Safety Arguments

Tangming Yuan[1], Suresh Manandhar[2], Tim Kelly[3], and Simon Wells[4]

[1] University of York tommy.yuan@york.ac.uk
[2] University of York suresh.manandhar@york.ac.uk
[3] University of York tim.kelly@york.ac.uk
[4] Edinburgh Napier University s.wells@napier.ac.uk

Abstract. Safety cases play a significant role in the development of safety-critical systems. The key components in a safety case are safety arguments, that are designated to demonstrate that the system is acceptably safe. Inappropriate reasoning with safety arguments could undermine a system's safety claims which in turn contribute to safety-related failures of the system. Currently, safety argument reviews are conducted manually, require expensive expertise and are often labour intensive. It would therefore be desirable if software can be employed to help with the detection of flaws in the arguments. A prerequisite for this approach is the need for a formal representation of safety arguments. This paper proposes a predicate logic based representation of safety arguments and a method to detect argument fallacies. It is anticipated that the work contributes to the field of the safety case development as well as to the area of computational fallacies.

1 Introduction

As technology advances, microprocessors and the software that runs on them have found their way into the heart of products that many of us routinely use as part of our daily lives. The presence of microprocessor-based electronic control units in devices that people lives depend upon, such as the braking systems of cars and radiation therapy machines in hospitals, justifies the importance of safety as a foremost requirement in the engineering of these crucial systems. Safety-critical systems include any system where failure could result in loss of life, significant property damage, or damage to the environment. Safety-critical systems are deployed in a wide range of sectors and industries, such as high-speed rail in the transport sector and nuclear power plants in the energy sector.

These systems have high dependability requirements. That is, they are frequently subjected to industrial, national, and international regulations that require compliance to rules or procedures in their design, deployment, operation, and decommission process, the attainment of one or more minimum standards in areas such as security, reliability, availability, or safety. The construction of a safety case, or functionally equivalent documentation, is mandated in many standards used to guide the development of software for safety-critical systems, such as the UK Ministry of Defence standard DS 00-55 [1] and Part 3 of the International Electrotechnical Commission (IEC) standard 61508

[2]. A safety case is defined by Bishop and Bloomfield [3] as "A documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment". The approach is to support sophisticated engineering arguments, for example, by assuring a safety argument within a safety case. This approach aims to demonstrate clearly how the safety requirements are fulfilled by the presented evidence, and thus derive confidence in the system's dependability. A key strength of this approach is to make the set of arguments explicit and available for introspection. This in turn increases confidence that the form of argument and its conclusion are both sound. Arguments are by their nature subjective, and their robustness is not self-evident (e.g. confirmation bias [4]). To increase the soundness of the arguments, a review element is necessary for the assurance of safety cases. A review normally involves two parties: the proposing party, typically the system engineer, who asserts and defends the safety case, and the assessing party, e.g. an independent safety assessor, who represents the certification authority, and whose task is to scrutinise and attack the arguments to uncover any vulnerability. The objective of a review is for the two parties to form a mutual acceptance of their subjective positions [5]. A safety argument review model [6] and tool [7] have been developed to facilitate this process. Despite the usefulness of the review framework, the quality of review arguments is not guaranteed as this largely relies on the reviewers' strategic wisdom and expertise. A complementary approach, we argue, is to provide users with a software agent, which can assist the reviewers to detect argument flaws (e.g. conflict and circular arguments) on the fly so that the argument quality can be improved. This paper aims to investigate a suitable methods for the automatic detection of argument flaws leading to additional assurance regarding the dependability of the system. This is achieved by providing a formal underpinning for the Goal Structuring Notation [8, 9], a graphical language for safety arguments, that is widely adopted for describing safety critical systems during the safety assurance process.

The rest of the paper is organized as follows. Section 2 discusses the current graphical representation of safety arguments and the need of a formal representation at sentence level. Section 3 proposes a predicate logic-based ontology via domain analysis of an existing safety case. Section 4 discusses how safety argument fallacies can be detected via the ontology. Section 6 concludes the paper and gives pointers for our planned future work in this area.

2 Safety Argument Representation

Graphical notations are often deployed to represent arguments in a more structured and transparent manner as exemplified by, e.g. Buckingham Shum [10], Gordon and Walton [11], and Reed an Rowe [12]. In the safety-critical domain, there are two established, commonly used notations the Goal Structuring Notation (GSN) proposed by the University of York [8, 9] and the Claims Argument Evidence notation proposed by Adelard LLP [13]. The GSN has been adopted by an increasing number of companies in safety-critical industries and government agencies, such as the London Underground and the UK MoD, as a standard presentation scheme for arguments within safety cases [14].

For example, 75% of UK military aircraft have a safety case with safety arguments expressed in GSN [13]. This paper will use GSN to represent arguments graphically unless stated otherwise.

The GSN uses standardised symbols to represent an argument:

- individual constituent elements (claims, evidence, and context)
- relationships between elements (e.g. how claims are supported by evidence).

In GSN, claims in an argument are shown as Goals (rectangles). They are often broken down into sub-goals further down a hierarchy. Alternatively they may be supported by evidence, presented in the GSN as Solutions (circles), and for an argument to be robust, all sub-goals must eventually be supported by solutions at the bottom. Strategies adopted (especially when breaking down goals) are shown in parallelograms, and they are related to argument schemes [15]. Contexts in which goals are stated appear as bubbles resembling racetracks. If necessary, ovals are used in GSN to denote Assumptions and Justifications. They can be distinguished by an A or J at the lower right of the symbol. Two types of links are used to connect the constituent elements. A line with a solid arrowhead, representing a Supported by relation, declares an inferential or evidential relationship. Permitted connections are goal-to-goal, goal-to-strategy, goal-to-solution, strategy-to-goal. A line with a hollow arrowhead represents an In-context-of relation, that declares a contextual relationship. Permitted connections are goal-to-context, goal-to-assumption, goal-to-justification, strategy-to-context, strategy-to-assumption, and strategy-to-justification [8, 9].

An example use of the key components of the GSN is shown in Figure 1. The argument shows that in order to achieve the G1 both legs of evidence are collected and the arguing strategy is from diverse forms of evidence. For complex systems with many arguments, modular approaches [16] have been used to aid with argument abstraction and composition. There has been substantial experience of using graphical GSN-based arguments for a wide range of scales of safety argument (from single programmable devices to whole aircraft safety cases).

The graphical representation shown in Figure 1 is more structured and transparent compared to free text representation, and software can help with the conformance of GSN syntax to ensure a valid safety argument is a connected diagraph with each path ending in, at least, one item of evidence. However, the representation treats the content inside each GSN node as black-box and as result any argument flaws related to the content of the element cannot be detected by an automatic means. A formal representation for the contents inside the nodes is necessary so as to make them machine processible. There are at least three possible approaches to formal representation of a sentence that can be found from the literature, namely, propositional logic, predicate logic and description logic. A predicate-based approach is chosen in this paper due to its expressive power than a proposition-based approach. We will cater for the description-based approach in the future. A predicate-based representation of GSN elements will be discussed next.

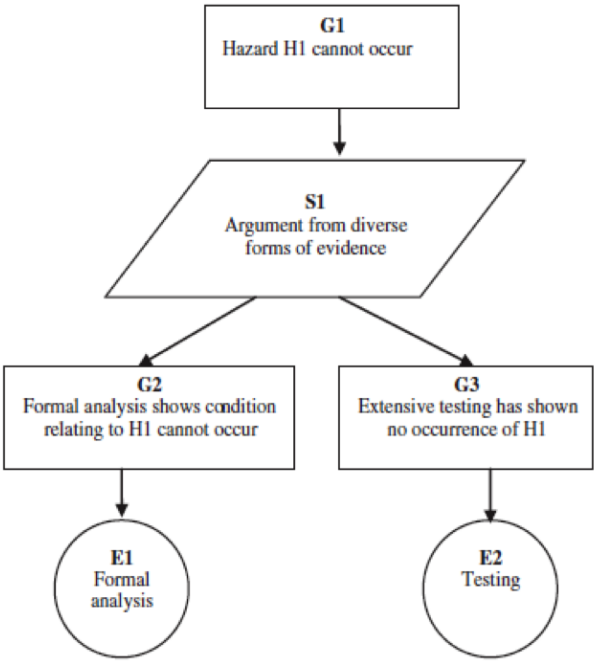

Fig. 1. An example showing the use of key components of the Goal Structuring Notation.

3 Safety Argument Ontology

For a predicate-based approach to formal representation of GSN elements, it is necessary to build an ontology that contains a set of constant symbols, function symbols and predicate symbols which form the vocabulary for the expressions of GSN nodes. To derive the ontology vocabulary, domain analysis of some existing safety cases has to be carried out. The preliminary study of the Europe Air Traffic Management (ATM) System Safety Case [17] was chosen to serve the domain analysis in order to form the first prototype of the ontology. The ATM study was conducted to evaluate the possibility of developing a whole airspace ATM system safety case for airspace belonging to EUROCONTROL member states.

Our domain analysis focuses on the GSN elements and their related documentation in the report. The report was read manually and the frequencies of relevant keywords were counted with the aid of the document search function. As a result of the analysis, the frequently used terms came into light and they are system, hazards, hardware, software, integrity level, probability and process. The frequently used relations or interactions among the terms are “meeting standards”, “lower than” and “greater than”. The actions normally placed on the objects are reviewed, analyzed, eliminated and mitigated. The subjective view is often safe. There are also patterns discovered when using these terms, for example, when hazards are used, the actions such as *eliminated* and *mitigated* will

follow; when process is used, actions such as *implemented* and *meet certain standard* will be used; when an item of evidence is used, terms like *review* and *analysis* will be used. The analysis helps to build up the initial set of safety-argument vocabulary [18], for example:

Constant Symbols :
System(Name), Standard(Name), Authority(Name), Hardware(Name), Software(Name),
Process(Name), Condition(Content), Adv(Content), Hazard(Name).
Function Symbols :
probabilityOf(Object), integrityOf(Object), hardwareOf(Object, namedAs), hazardOf(Object, namedAs), softwareOf(Object, namedAs), processOf(Object, namedAs), analysisOf(Software), reviewOf(Software)
Predicate Symbols :
meetStandard(Object,Standard), greaterThan(Object1, Object2), lowerThan(Object1,Object2), hasBeenImplemeted(Object, adv), isSafe(Object, Condition), hasBeenReviewed(Object, adv.), isAlike(Object1, Object2).

The atomic level is the constant symbols. Constant symbols denote the entities or objects in the domain. Function symbols denote functions from tuples of objects to objects. In system safety argument, it is essential to represent the objects inside the sentence, such as "hardware of system x". In order to achieve this, function symbols are used to solve this problem, hardwareOf(system(x)) will present "hardware of system x". Function symbols will serve to identify the properties of certain objects. Based on the nature of the properties, they can be further divided into two categories: countable and uncountable. For the countable properties, each property needs to be distinguished and it is necessary to do so, and this can be formulated as: symbols (object, name). For example, hazardOf(system(x), h1) can be read as "hardware of system x named as h1". So the "name" parameter inside the functions symbol will help to distinguish the countable properties. On the other hand uncountable properties are not necessarily distinguished, for example, integrityOf(system(x)). Comparing to predicate symbols, function symbols donate objects instead of a complete sentence. We use content to distinguish function symbols since a function symbol is normally an argument inside a predicate symbol.

Predicate symbols are used to represent a sentence. A sentence normally provides an action on the object or a description of an object. Based on the nature of the sentence, the predicate symbols can be divided into two categories. The actions can be formulated as symbol (object, adv.), such as eliminated (hazardOf(system(x)), completely), implemented(processOf(system(x)), safely). There are other actions such as meeting certain standard and comparison objects (e.g. lower than, higher than). These predicates can be formed as symbols (object 1, object 2), such as meetingStandard(processOf(system(x), standard(y)) can be translated as "the process of system x meet the standard y", and lowerThan(probabilityOf(hadzardOf(x), probabilityOf(standardOf(x0)) means that the probability of hazard x happens is lower than the probability required by standard x. It is also essential to present the adjective words that describe the object, such as safe. However the adjective words to describe the object can only be valid in some conditions.

In order to include the conditions in the predicate, the adjective predicate symbols can be formulated as: symbol (object, condition). For example: isSafe(system(x), Condition(hazards are avoided)) means that system x is safe when hazards are avoided.

4 Automatic Detection of Safety Argument Fallacies

A fallacy is defined by Damer [19] as a mistake in an argument that violates one or more of the five criteria of a good argument: i) A well-formed structure, ii) Premises that are relevant to the truth of the conclusion, iii) Premises that are acceptable to a reasonable person, iv) Premises that together constitute sufficient grounds for the truth of the conclusion, v) Premises that provide an effective rebuttal to all anticipated criticisms of the argument. In safety arguments, fallacies exist in different forms. Greenwell *et al.* [20] studied a number of safety cases, such as EUR Reduced Vertical Separation Mininums (RVSM) and EUR Whole Airspace Preliminary and derived a number of fallacies in safety cases organized into three categories namely, relevance, acceptability and sufficiency fallacies. This paper examines a subset of these argument fallacies and how they can be detected in an automatic means via the predicate-based representation as outlined in section 3 above. The fallacies e.g. appeal to improper authority, fallacious use of language, faulty analogy, circular argument, fallacious composition and confusion of necessary & sufficient condition, are discussed in turn below and each followed by means to detect them.

4.1 Appeal to improper authority

The fallacy of appeal to improper authority is a member of relevance fallacy family where arguments that violate the relevance criterion of a good argument. They employ irrelevant premises or make appeals to irrelevant factors to draw a conclusion. The fallacy of appeal to improper authority attempts to support a claim by appealing to the judgment of an authority which is actually not an authority in the field and likely to be biased [19]. The authorities cited in safety arguments could be individuals, committees, standard documents, “best practices”, and system pedigree [20]. The fallacies occur mostly in the form of transferring one authority’s competence into another field in which its competence is not valid. For example, an entertainer or athlete is appealed as an authority on marriage and family.

To automatically detect this fallacy, constant symbols such as standard() and Authority() should be used. A database can be built so that each authority can be checked against their field of expertise. For example, meetStandard(processOf(system(x)), standard(y)) => safe(system(x)). The standard(y) will be checked against the database to verify whether it is the correct one being applied.

4.2 Fallacious use of language

The fallacy of the use of language occurs when an argument violates the acceptability criterion for a good argument. There are five types of unacceptable premises: i) A claim

that contradicts credible evidence, a well-established claim, or a legitimated authority, ii) A claim that is inconsistent with ones own experience or observations, iii) A questionable claim that is not adequately defended in the context of the argument or not capable of being adequately defended by evidence in some other accessible source, iv) A claim that is self-contradictory or linguistically confusing, v) A claim that is based on another unstated but highly questionable assumption [19]. Fallacious use of language typically happens due to a lack of clarity in the meaning of a key word or phrase used in the premise. An ambiguous word, phrase, or sentence is the one that has more than one meaning. The inferential relationship between claims in argument should clearly define the exact meaning being used. A typical example is to describe the desirable system properties by using expressions such as safety, reliability and dependability interchangeably.

To detect this fallacy automatically, it is essential to know common misleading words, or phases. Since a sentence is represented by predicate symbols which are pre-defined, the fallacious use of language can be reduced significantly. For example: isSafe(), isReliable(), isDependent() will have different meaning in safety argument, and they are treated differently in the logical representation to avoid any confusion with their meanings.

4.3 Faulty Analogy

Faulty analogy is a type of acceptability fallacy. It assumes that because two things are alike in one or more respect, they are necessarily alike in some other important respect. This fallacy fails to distinguish the insignificance of their similarities and the significance of their dissimilarities [19]. In safety argument cases it could be that, using the argument for the development of the previous system to support current system without stating the differences between these two systems. There is a typical example, the Ariane 5 accident in 1996 could be the result from a faulty analogy within the rockets safety cases [21].

To enable a machine to detect such a fallacy, the predicate symbol isAlike() and isMinor() can be used. For example, isAlike(system(x), system(y)) $\wedge$ isSafe(system(y)) => isSafe(system(x)) can be read as because system x and system y are alike, system y is safe, so system x is safe. When this situation happens, it is easy to identity the missing elements such as the justification on the difference. The expression "isAlike(system(x), system(y)) $\wedge$ isSafe(system(y)) $\wedge$ isMinor(differenceOf(system(x), system(y)) =>isSafe(system(x))" seems to be more convincing than the one without such a justification.

4.4 Circular Argument

Circular argument is a type of acceptability fallacy that involves either explicitly or implicitly asserting in the premise of the argument is asserted in the conclusion of that argument [19]. Instead of providing supporting evidence, it simply brings up the conclusion as its evidence. In the standard form, it looks like: since A (premise), therefore,

A (conclusion). And it also can be implicitly assuming the conclusion is true. For example, when people argue God exists because he does not want to go to the hell. By arguing that, he already assumes that God exists, which is the conclusion he want to draw. Sometimes it is hard to detect this kind of fallacy, since different words or different forms may be used in the premises or conclusion. The complexity of an argument also causes the difficulty to detect the kind of fallacy because the conclusion may be drawn far away after the premises.

To automatically detect such a fallacy in safety arguments, it is necessary to find a similar argument that is in the circle, for example: isSafe(system(x)) => meetStandard(processOf(System(x)), Standard(y)), meetStandard(processOf(System(x)), Standard(y)) => isSafe(System(x)). Based on the predicate symbols used in the argument, it is feasible to trace the argument and identify similar arguments used as supporting arguments.

4.5 Fallacy of Composition

Fallacy of composition is a type of acceptability fallacy that assumes that every part is true therefore the whole is true, without taking the relations between the parts into account [19]. Typical example could be a football team with excellent players may not be a good team, because when gathering excellent players together, their skills may be compromised to team work. In safety cases, the example could be argument for the whole system is safe by supporting sub-system A, B, C are safe, which ignores the interactions between these sub-systems [20].

The fallacy can be detected with aid of the predicate-based language. With such a representation, The pattern of argument from system decomposition [15], e.g. isSafe(hardwareOf(System(x), h1)) $\wedge$ isSafe(hardwareOf(system(x), h2)) => isSafe(system(x)) can be identified. Upon the identification of the argument pattern, the predicate with regard to the component interaction can be searched for. As an example, the argument of isSafe(hardwareOf(system(x), h1)) $\wedge$ isSafe(hardwareOf(system(x), h2)) $\wedge$ isNone(interactionOf(hardwareOf(system(x))) => isSafe(system(x)) is a proper argument while missing the isNone will result in fallacy of composition.

4.6 Confusion of Necessary & Sufficient Conditions

This fallacy occurs when there is insufficient evidence is provided, e.g. no or little evidence, biased or week evidence, or omitting crucial types of evidence. When arguing for or against a position, relevant and acceptable reasons should be provided, together with justification of the sufficiency in number or weight to the acceptance of the conclusion.

Necessary condition of an event is a condition that must be present in order for an event to happen. Sufficient condition will trigger the occurrence of an event. In safety cases, a typical example argument is “hazards have been mitigated” with evidence showing the case that hazards have been mitigated. However, in order to mitigate hazards, the sufficient condition is to identify all the hazards. To better support the argument, the

identification process should be included [15]. In the argument structure, it is often valid to say isSafe(System(x)) => isSafe(System(x), condition1), while it is invalid to say isSafe(system(x), condition1) => isSafe(system(x)) due to insufficient conditions for the whole system to be safe. The fallacy of distinction without a difference is opposite to faulty analogy and fallacy of division is the opposite of fallacy of composition. The means to detect these fallacies are similar. There are also other safety argument fallacies that are categorized in [20] but not examined in this paper, e.g. red herring, drawing the wrong conclusion, using the wrong reasons, false dichotomy, pseudo-precision, hasty inductive generalization, arguing from ignorance, omission of key evidence, ignoring the counter-evidence and gambler's fallacy. We are going to cater for these in the future in terms of the appropriate representations in order to detect them by an automatic means

5 Implementation & Evaluation

In order to facilitate the evaluation of the fallacy detection approach and algorithms, the safety argument review tool, namely DiaSAR as reported in [22] has been extended by incorporating fallacy detection functionalities. DiaSAR operates a safety argument review model (SARM) which is a dialogue based model. The review process under the SARM encompasses three distinct phases: initiation, review, and revision. It starts with an initiation of a proposal of safety arguments followed by reviews conducted by independent reviewers. The proposer then responds and revises the initial proposal in light of the criticisms made by the reviewers. The revised version of the safety arguments will be further reviewed until reviewers reject or accept the arguments or the proposer withdraws. The number of iterations can be defined by mutual agreement of all participants if they wish to set such a limitation. However, the ultimate aim of the iterations is to reach a position where the safety arguments are mutually accepted by both proposer and reviewers.

DiaSAR has a graphical user-interface that supports multiple user access and a back-end database storing the user profiles and the review sessions. There are three types of users of the system: system administrator, argument proposer, and reviewer. The system administrator manages all the users of the system and the review sessions. The argument proposers propose and subsequently defend their arguments. The reviewers criticise the arguments made by the proposer. An example system interface for the proposer can be seen in Figure 2. A proposer can create a new review session which specifies a session name, a proposer, and a reviewer of the session. The interface displays the current status of the session, e.g. the current player, his/her role, and the current step of the review. A proposer can use provided tools (i.e. claim, strategy, evidence, context, assumption, and links) to construct safety arguments following GSN syntax as described in Section 2. A session can be saved and loaded for further editing. Once it has been done, the turn can be passed to the reviewer and a notification email will be sent to the reviewer. The session will then transit from the proposing state to the reviewing state.

A reviewer can log onto the system and load the sessions under review. By right clicking on any elements of the safety argument, a submenu will be available with items for

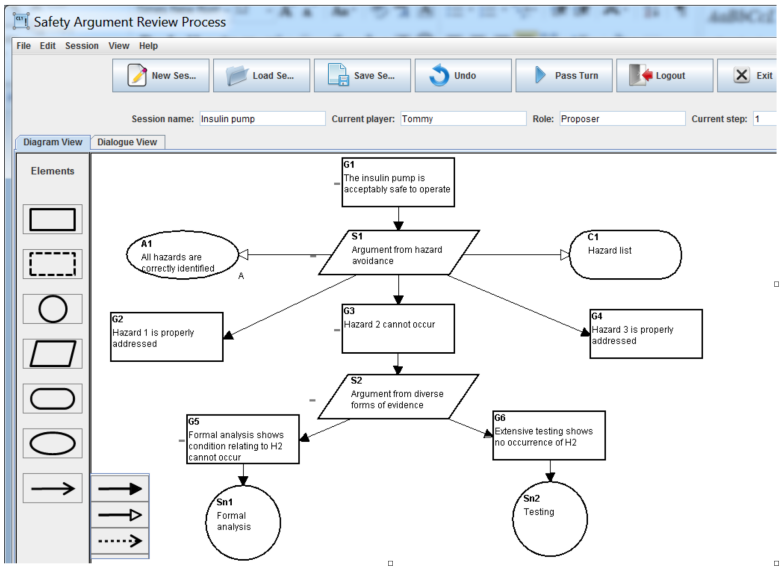

Fig. 2. An example user interface for the proposer.

the reviewer to accept, challenge, and question an argument element. A reviewer can also propose a counter-argument. Graphical notations have been developed for the set of the review tools alongside GSNs. Some of them can be seen in Figure 3. For example, a rectangle with dashed borderline represents a counter-argument and a dashed line with an open arrow represents an 'attacked by' relation (e.g. the argument 'Extensive testing shows no occurrence of H2' is attacked by argument 'Accident database shows the occurrence of H2').A question mark represents a question (e.g. Is the analyst experienced?) and an exclamation mark represents a challenge (Why is it the case that hazard 3 is properly addressed?) made by the reviewer. Accepted elements are coloured green and withdrawn elements red. A short vertical line with open arrow at both ends is used to mark the situation where a resolution demand is made to a conflict among two or more elements.

When the review is completed, the reviewer can pass the turn back to the proposer. The session will transit from the review state to the revision state and a notification email will be sent to the proposer. Upon receiving the notification, the proposer can respond to the criticisms and revise the arguments following SARM rules, e.g. to respond to a challenge with a withdrawal, claim, evidence, or strategy. These rules are implemented as submenus attached to a review element where the proposer can select a suitable response. The responses, e.g. newly made claims, strategies, evidence, context, or assumptions, are automatically displayed as part of the safety argument, and the symbol of the review being responded to is removed from the user-interface. Once all the reviews have been responded to and other necessary revisions are done, it can be

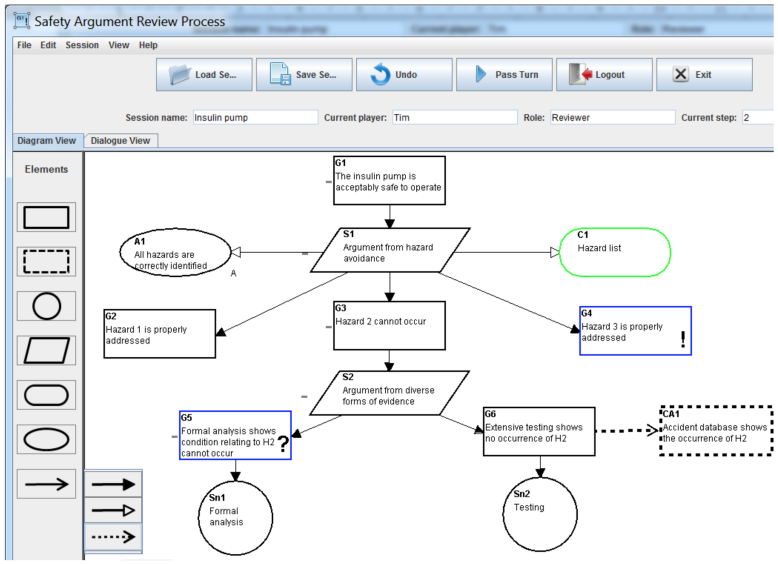

Fig. 3. An example user interface for the reviewer.

passed back to the reviewer for a second review. The process goes on until the safety argument has been fully accepted.

6 Conclusions & Future Work

The work presented in this paper is but one stage within a wider program of research on analysis of existing safety cases in order to enrich the ontology vocabulary and automated analysis of safety arguments. Additionally, there are opportunities within other domains, such as security policy documents which utilise similar constrained and formalised domain specific languages to represent clauses within the policies. We also plan to incorporate the safety argument ontology into current safety argument software tools such as that from [7]. Additionally, a dialectical model of interaction between the engineer and a safety-system agent could support the elicitation of higher quality safety cases in the first instance. For example, such a dialectical approach could build existing educational dialogue approaches [23], existing knowledge elicitation interactional approaches [24] or else involve the development and description of a wholly new protocol using appropriate technologies [25]. A similar dialectical model, but using a mixed initiative approach could support the assessor in thorougly exploring potential fallacies, and other deficiencies, within a case. However such approaches are reserved, at present, for future work. An important aspectthat has not been adequately addressed in this paper is evaluation. In one sense, identifying fallacies that would otherwise have been missed, would be weakly indicative of a useful system. For example, by applying the formal approach from this paper to the cases investigated by Greenwell *et al.* [20] and automatically identifying the fallacies previously discovered.

This paper reports our work in adopting a predicate-based approach to represent safety arguments in order to facilitate the automatic detection of fallacies. Particularly, we have conducted a domain analysis of an existing safety case and the analysis helps the generation of a set of ontology vocabularies. Via the domain ontology, methods for machine detection of some outstanding argument fallacies have been proposed. The fallacies cannot be detected without such a formal representation. In terms of the contribution of the paper, the proposed safety argument representation goes further than the current graphical representation, e.g. GSN, and the new representation makes the GSN nodes from black-boxes to white-boxes and thus being machine processible. The work will contribute to both the field of the safety case development and the area of computational fallacies.

References

1. UK Ministry of Defence: Defence standard 00-55 the procurement of safety critical software in defence equipment. `http://www.dstan.mod.uk/` (1997) Accessed: 20th May 2011.
2. International Electrotechnical Commission: Functional safety of electrical/electronic/programmable electronic safety-related systems (IEC 61508 ed2.0). `http://www.iec.ch/` (2010) Accessed: 20th May 2011.
3. Bishop, P.G., Bloomfield, R.E.: A methodology for safety case development. In: Safety-Critical Systems Symposium (SSS 98). (1998)
4. Leveson, N.: The use of safety cases in certification and regulation. Journal of System Safety **47**(6) (2011) 1–5
5. Kelly, T.P.: Reviewing assurance arguments a step-by-step approach. In: Proceedings of workshop on assurance cases for security The metrics challenge, dependable systems and networks (DSN), Edinburgh. (2007)
6. Yuan, T., Kelly, T.: Argument-based approach to computer system safety engineering. International Journal of Critical Computer-based Systems **3**(3) (2012) 151–167
7. Yuan, T., Kelly, T., Xu, T.: Computer-assisted safety argument review a dialectics approach. Argument & Computation (2014)
8. Kelly, T.P.: Arguing safety A systematic approach to safety case management. PhD thesis, Department of Computer Science, University of York, York (1999)
9. Kelly, T., Weaver, R.: The goal structuring notation a safety argument notation. In: Proceedings of the dependable systems and networks 2004 workshop on assurance cases, Florence. (2004)
10. Buckinghham Shum, S.: Proceedings of the 2nd international conference on computational models of argument (comma08), toulouse. In: Cohere: Towards web 2.0 argumentation. (2008)
11. Gordon, T., Walton, D.: The carneades argumentation framework: Using presumptions and exceptions to model critical questions. In: Proceedings of computational models of argument (COMMA 2006), IOS Press (2006) 195–207
12. Reed, C.A., Rowe, G.W.A.: Araucaria: Software for argument analysis, diagramming and representation. International Journal of AI Tools **13**(4) (2004) 961980
13. Emmet, L., Cleland, G.: Graphical notations, narratives and persuasion: A pliant systems approach to hypertext tool design. In: Proceedings of the thirteenth ACM conference on hypertext and hypermedia, conference on hypertext and hypermedia. (2002)
14. Group, O.M.: Argument metamodel. `http://www.omg.org/spec/ARM` (2010)
15. Yuan, T., Kelly, T.: Argument schemes in computer system safety engineering. Informal Logic **31**(2) (2011) 89–109

16. Kelly, T.P.: Using software architecture techniques to support the modular certification of safety critical systems. In: Proceedings of eleventh Australian workshop on safety-related programmable systems, Melbourne. (2005)
17. Kinnersly, S.: Whole airspace atm system safety case preliminary study. a report produced for eurocontrol by aea technology, aeat ld76008/2 issue 1. (2001)
18. Wan, F.: Auto-detecting fallacies in system safety arguments. Master's thesis, University of York, York (2013)
19. Damer, T.E.: Attacking Faulty Reasoning: A Practical Guide to Fallacy-Free Arguments. Sixth edn. Wadsworth Cengage Learning (2009)
20. Greenwell, W.S., Holloway, C.M., Knight, J.C.: A taxonomy of fallacies in system safety arguments. In: Proceedings of the international conference on dependable systems and networks, Yokohama, Japan. (2005)
21. Lions, J.L.: Ariane 501 failure: Report by the inquiry board. European Space Agency (July 1996)
22. Yuan, T., Kelly, T., Xu, T., Wang, H., Zhao, L.: A dialogue based safety argument review tool. In: Proceedings of the 1st International Workshop on Argument for Agreement and Assurance (AAA-2013), Kanagawa, Japan. (2013)
23. Yuan, T., Moore, D., Grierson, A.: A human-computer dialogue system for educational debate, a computational dialectics approach. International Journal of Artificial Intelligence in Education **18**(1) (2008) 3–26
24. Reed, C., Wells, S.: Dialogical argument as an interface to complex debates. IEEE Intelligent Systems Journal: Special Issue on Argumentation Technology **22**(6) (2007) 60–65
25. Wells, S., Reed, C.: A domain specific language for describing diverse systems of dialogue. Journal of Applied Logic **10**(4) (2012) 309–329