Supporting the resolution of inconsistencies in specifications based on mathematical argumentation theory

Keishi Okamoto and Kazuma Kokuta
National Institute of Technology, Sendai College

Abstract

In this paper, we propose a method to support for resolving "inconsistencies" in a requirement specification document which is written in a natural language. We also develop a tool based on the method.

We use mathematical argumentation theory and natural language processing to realize the method. Based on mathematical argumentation theory, we can formulate various " inconsistencies " including logical contradiction as an attack relation R in an argumentation framework (A, R). Then an extension S in (A, R) represents a set of acceptable descriptions of a requirement specification document. Moreover, an extension S suggests an engineer the set of descriptions which should be corrected to resolve "inconsistencies" by referring R.

Our method consists of the following methods. First, we adopt the method in [1], which is based on natural language processing, to generate an argumentation framework (A, R) from a requirement specification document. Second, we use the method in [2] to define an extension S of (A, R) in an extension of first-order logic, and then we use the method in [3] to enumerate extensions from (A, R) by solving a Partial Maximal Satisfiable Subsets Enumeration problem that is an extension of a Maximal Satisfiable Subsets Enumeration problem. Finally we visualize the (A, R) and S's to support for resolving "inconsistencies" in a requirement specification document.

1 Introduction

In IEEE 830-1998[4], quality attributes of requirement specification document consists of correctness, unambiguity, completeness, consistency, ranked for importance and/or stability, verifiability, modifiability, traceability. It is important to describe the requirements specification document so that it has these attributes. Many research has been conducted to validate and ensure these attributes of requirements specification document. However, in reality, these attributes are often ensured by manual review, and further research is needed. In this paper, we focus on "consistency" of requirement specification documents.

Again, in IEEE830, “consistency” is defined as the individual requirements do not “contradict” each other. In [5], the following are examples of “inconsistencies”:

- Inconsistent Software Operations: The behavior and output of the software for the same input are described in multiple places, and they are different.
- Inconsistent Definitions: The definition for the same word are described in multiple places, and they are different.
- Inconsistent Constraints: There are constraints, but there is no solution that satisfies the constraints.

Detecting “inconsistencies” in a requirement specification documents is time-consuming task. In most cases, we detect "inconsistencies" in requirement specification documents by reviewing. But this manual detection method is error-prone and time consuming. On the other hand, we can detect "inconsistencies" with formal methods when specifications are written in a formal language. In this case, we can detect that the set of sentences A_1 and A_2 is "inconsistent" by proving that $A_1 \wedge A_2$ yields logical contradiction. For instance, let A_1 be a sentence representing that "the initial value of the variable x must be set to 0". Then A_1 can be described as the formula of a temporal logic as follows:

$$A_1 : temp \geq 90 \wedge mode = heating \rightarrow \Box(mode = retention).$$

And, let A_2 be a sentence representing that "the initial value of the variable x must not be set to 0". Then A_2 can be described as the formula of a temporal logic as follows:

$$A_2 : temp \geq 90 \wedge mode = heating \rightarrow \Box(mode = heating).$$

Thus, we can prove that $A_1 \wedge A_2 \vdash \bot$, namely the set $\{A_1, A_2\}$ is "inconsistent" when $retention \neq heating$ and $temp \geq 90 \wedge mode = heating$ holds. However, describing an entire specification in a formal language is a very time-consuming task.

There are various kinds of "inconsistencies" in a requirement specification documents. In a requirement specification document, there are sets those are "inconsistent" but is not logically contradict. For instance, let A_3 be a sentence representing that "if the value of the water temperature sensor $\geq$ 90 degrees, transit from the heating mode to the heat retention mode". Then A_3 can be described as the following formula:

$$A_3 : temp \geq 90 \wedge mode = heating \rightarrow \Box(mode = retention).$$

And, let A_4 be a sentence representing that "if the value of the water temperature sensor is $\geq$ 95 degrees, transit from the heating mode to the heat retention mode". Then A_4 can be described as the following formula:

$$A_4 : temp \geq 95 \wedge mode = heating \rightarrow \Box(mode = retention).$$

A_3 logically implies A_4 while A_4 does not logically imply A_3. Thus, the set of A_3 and A_4 is not logically contradict. On the other hand, the sentence A_3 is more safer than A_4. Then we want to accept the safe requirement A_3 and reject the unsafe requirement A_4, namely the set of A_3 and A_4 is "inconsistent".

We can detect that the set of descriptions is logically contradict when we naively formalize a requirement specification document. Moreover, we can detect a wider range of "inconsistencies", if we add a wide range of inconsistencies as formal axioms to the formal specifications. However, adding these axioms is a very time-consuming task, as they involve many physical properties, tacit knowledge, etc.

Even if we can detect that sentence A and sentence B are logically contradictory, it is not possible to determine which sentence to accept and which to reject only from the information of A and B. In general, if we detect that the set of A and B is "inconsistent" then we want to decide which sentence is acceptable and the other should be rejected. However, we often need to consider other descriptions of the document to decide it. Moreover, it is difficult to find descriptions showing us which sentence is right.

This paper is structured as follows. We briefly introduce our previous work to define and enumerate extensions in Section 2. We show a method to construct an argumentation framework from a requirement specification document with natural language processing in Section 3. Then we conduct experiments to verify the validity and the scalability of our tool in Section 4. Finally, we conclude the paper in Section 5.

2 Defining and Enumerating Extensions

In this section we introduce our previous works [2, 3]. In [2], we define some extensions as FO-definable subset and other extensions as a maximal subset of FO-definable subset. Then, in [3], we enumerate extensions with an extension of an SMT solver Z3[6].

We show overall picture of support for resolving "inconsistencies" in Figure 1. Our method and tool is based on mathematical argumentation theory. First, we generate an argumentation framework (A, R) of mathematical argumentation theory from a requirement specification document with natural language processing. Second, we define and enumerate extensions S's from the argumentation framework. Extensions represent "consistent" sets of descriptions of the document, and some kinds of extensions are maximal satisfiable subsets (MSS's), satisfying a FO-formula, of (A, R). Finally, we support resolving "inconsistencies" of the document by showing a graph. The graph shows an acceptable set of descriptions that is a MSS S. The graph also shows a description of the document should be corrected referring to S and R.

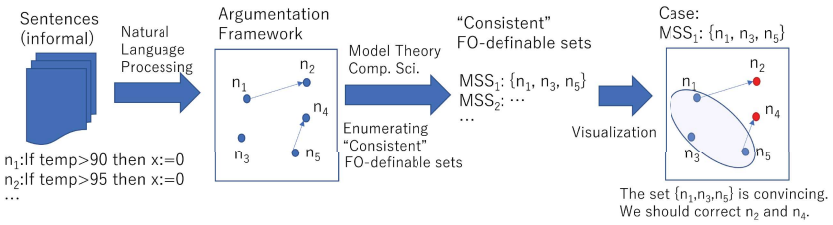

Figure 1: Overall picture of support for resolving "inconsistencies"

2.1 Mathematical Argumentation Theory

Our previous works[2, 3] is based on mathematical argumentation theory. Argumentation theory is an interdisciplinary field that has been studied interdisciplinarily in many fields, such as sociology, linguistics, psychology, logic, dialectics, etc., on the issue of "how to justify a claim". Moreover, mathematical argumentation theory is argumentation theory based on mathematical methods[7]. Mathematical argumentation theory is broader than mathematical logic because it also deals with a wider range of non-mathematical subjects. Therefore, by using the *argumentation framework* of mathematical argumentation theory, we can treat various "inconsistencies" other than logical contradictions in a broader sense. Moreover, by using the *extensions* of mathematical argumentation theory, we can mathematically define what is "consistent" under a broad sense, and thus provide evidence for deciding which sentences are trustworthy when the set of the sentences is "inconsistent". Moreover, once we have brought it into the framework of mathematical argumentation theory, we can use the methods of mathematical logic[8] and computer science[9] to automatically support resolving "inconsistencies".

Definition 1 *An argumentation framework (A, R) is a directed graph such that A is a set of vertex, called abstract arguments, and R is a relation, called an attack relation, on A.*

For instance, "a attacks b $(a \to b)$" for $a, b \in A$ means that "if a is true then b must be false". For "c: It will be sunny tomorrow" and "d: It will rain tomorrow", we have $c \leftrightarrow d$. And, for "e: Liquid temperature is always below 100 ℃ (possibly 95 ℃)" and "f: Liquid temperature is always below 90 ℃", we have $e \to f$ but $f \not\to e$.

2.2 Describing Extensions as Formulas

In this subsection, we introduce the previous work in [2] to describe extensions as FO-definable subset or maximal satisfiable subset satisfying a FO-formula.

Based on mathematical argumentation theory [10], we have formalized "inconsistency" as an attack relation of an argumentation framework[2]. In [2], we represent a document, which is a set of descriptions of the document, as a

directed graph (A, R) where a set A is a set of descriptions of the document and an attack relation $R \subseteq A \times A$ represents "inconsistencies" in the document. Then, a sentence, which is a kind of descriptions, is a node $a \in A$. And "inconsistent" pair of descriptions a_1 and a_2 represents an edge from a_1 to a_2 ($\in R$), which is denoted by $a_1 \to a_2$. For instance, $a_1 \to a_2$ represents that a_1 is safer than a_2.

We formally describe a stable extension and a preferred extension. In [11] authors describe some kinds of extensions, including stable extensions, as FO-formulas. We use the same description of [11] for these kinds of extensions. Moreover, we described a preferred extension as a maximal satisfiable subset satisfying a FO-formula. These descriptions allows us to enumerate extensions with an extension of the SMT solver Z3.

We give a definition of a stable extension.

Definition 2 *1. A subset S of A is conflict-free (CF) if $\neg\exists a, b \in S\,((a, b) \in R)$ holds.*

2. A subset S of A is a stable extension if it is CF and $\forall a \in A(a \notin S \to \exists b \in S(b, a) \in R)$ holds.

It important that these definition is FO-definable. Then a stable extension can be extract from (A, R) by Z3.

Next, we give a definition of a preferred extension.

Definition 3 *1. An argument a is defended by a set $S \subseteq A$ (or S defends a) if $\forall y(R(y, a) \to \exists z(S(z) \wedge R(z, y)))$ $(D(S, a))$ We say that "a is acceptable with respect to S" if " a is defended by S " .*

2. A subset S of A is admissible if it is CF and $\forall x(S(x) \to D(S, x))$

Note that these definition is FO-definable.

Definition 4 *A subset S of A is a preferred extension if it is a maximal admissible subset of (A, R).*

Since maximality of a FO-definable subset is not FO-definable, we need to extend the naive Z3 to extract a preferred extension.

We show an example of a stable extension and a preferred extension.

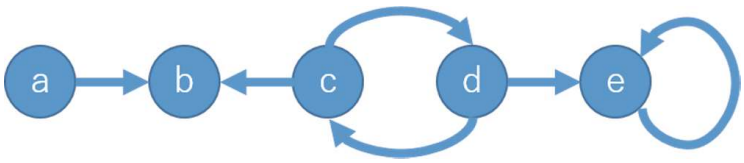

Figure 2: An example of an argumentation framework

In Figure 2, $\{a, d\}$ is CF and $\{e\}$ is not CF. Moreover, $\{a, d\}$ is a stable extension because $a \to b, d \to c, d \to e$. On the other hand, d is defend by

$\{a, d\}$, b is not defend by $\{a, d\}$. $\{a, c\}, \{a, d\}$ are admissible. But $\{b, d\}$ is CF but not admissible because $\{b, d\}$ does not defend b ($a \rightarrow b$ but $b \not\rightarrow a$ and $d \not\rightarrow a$). Thus, $\{a, c\}, \{a, d\}$ are preferred extensions.

2.3 Enumerating Extensions with an SMT Solver

We have proposed a method of resolving "inconsistency" based on model Theory and computer science [3]. In [3], we enumerate preferred extensions, which are maximal satisfiable subset satisfying a FO-formula, with an extension of the SMT solver Z3. In this subsection, we briefly introduce the method.

Since Z3 can extract a FO-definable subset of (A, R) and a stable extension is FO-definable, we can extract a stable extension from (A, R) with Z3. But a preferred extension is a maximal satisfiable subset satisfying a FO-formula, namely it is not FO-definable. We need to extend Z3 to extract a preferred extension. On the other hand, a naive enumeration method based on extraction method is time consuming. Thus we have proposed a enumeration method based on the Liffton's way of "Enumerating Maximal Satisfiable Subsets" (EMSS) [12]. We implement the method as an EMSS solver based on Z3.

We show an example of extracting an extension in Figure 2. Let S be a preferred extension, i.e. a maximal subset of A satisfying FO-definable properties of conflict-free $CF(S)$ and admissible $AS(S)$. In this case, we put hard constraints as $CF(S) \wedge AS(S)$ and soft constraints as $\{S(a), S(b), \ldots, S(e)\}$ where $A = \{a, b, c, d, e\}$. Then, EMSS solver extracts a $\{S(a), S(d)\}$ (and $\{S(a), S(c)\}$). This result shows that $S = \{a, d\}$ is a maximal subset of A satisfying $CF(S) \wedge AS(S)$. Thus, $\{a, d\}$ is a maximal admissible subset of A, namely $\{a, d\}$ is a preferred extension.

3 Generating an Argumentation Framework with Natural Language Processing

Once we have a method to enumerate extensions from an argumentation framework (A, R), the remaining issue is how we generate (A, R) from a requirement specification document which is described in a natural language, in particular, how detect "inconsistencies" in the documents to define an attack relation R representing "inconsistencies".

We adopt a method[1] which is based on natural language processing since it is almost impossible to formally define various "inconsistencies" in requirement specification documents. If we translate sentences to FO-formulas, and define axioms representing "inconsistencies" then we can detect "inconsistencies" with formal methods. But, it is a time consuming task and requires skill to translate sentences to FO-formulas. Moreover, the word "inconsistency" is ambiguous. Thus, it is almost impossible to define "inconsistency" in a formal way. On the other hand, the computer can learn "inconsistency" inductively with natural language processing.

In [1] we uses pre-trained language model Japanese BERT[13] and conduct fine-tuning to learn "inconsistencies" with Japanese SNLI dataset[14]. BERT is a language model, known for its high accuracy in a variety of tasks[15]. BERT learns linguistic features through pre-training. For example, BERT can understand two synonyms have the same meaning. And Japanese BERT is a language model trained on Japanese sentences. On the other hand, the SNLI dataset is a collection of triples (Class, Sentence1, Sentence2), where Class can be Entailment, Neutral or Contradiction and Sentence1 and Sentence2 are English sentences. Japanese SNLI dataset is a translation of SNLI dataset to Japanese.

4 Experiments

In this section, we conduct experiments to validate the usefulness and verify scalability of the methods and tools in Section 3. In Subsection 4.1, we apply our tool to a small example to show its usefulness. In Subsection 4.2, we apply our tool to randomly generated argumentation frameworks to show its scalability.

4.1 Experiments to validate usefulness

In this paper, we choose a requirement specification document of "話題沸騰ポット (electric kettle)"[16]. Natural language processing is highly depends on the target natural language, we adopt the document in Japanese. This document is a requirement specification document of a fictitious electric kettle for product development exercises. The document consists of 18 pages, about 4000 characters (in Japanese) The size of the resulting (A, R) is that $|A| = 150$ and $|R| = 6$.

We show a part of contents of the document.

- 本ポットでは水温を制御するため、以下の仕様に従ってヒータを制御します。

1 蓋が閉じられた場合

  - 温度制御可能な水位ならば沸騰状態に移行し、ポット内の水を加熱します。水温が 100 ℃に達した後も 3 分間加熱を続け、その後保温状態に移行します。
  - この場合、温度制御の操作量算出には目標温度 ON/OFF 方式（ヒステリシスなし）を適用します。

2 沸騰状態が終了した場合（高温モードが設定されている場合）

  - この場合、水温を 98 ℃に保つようにヒータを制御します。温度制御の操作量算出には PID 制御方式を適用します。

3 沸騰状態が終了した場合（節約モードが設定されている場合）

  - この場合、水温を 90 ℃に保つようにヒータを制御します。温度制御の操作量算出には PID 制御方式を適用します。

4 沸騰状態が終了した場合（ミルクモードが設定されている場合）

- この場合、水温を 60 ℃に保つようにヒータを制御します。温度制御の操作量算出には PID 制御方式を適用します。

5 沸騰ボタン押下により、強制沸騰する場合

- （1）の制御仕様と同様です。

6 保温設定ボタンにより、保温モードを変更した場合

- 移行した保温モードの制御仕様（(2)、(3)、(4)）に従い、目標温度に水温を保つようにヒータを制御します。
- ※ 仕様毎の温度制御の操作量算出方法は、機種によって変わる場合があります。

There are no “inconsistencies” in the document. Thus, we insert “inconsistent” pairs of sentences into the document to apply our method to the resulting (A, R). Now, among the “inconsistencies”, we focus on misrepresentations. In order to get closer to the natural misrepresentations contained in requirement specification documents, we deleted some parts of the sentences, instead of that we modify the correct representations to misrepresentations. As a result, misrepresentations of the numerical values are inserted.

In this case, we delete（高温モードが設定されている場合）from the sentence 2, （節約モードが設定されている場合） from the sentence 3 and （ミルクモードが設定されている場合） from the sentence 4, respectively, in anticipation of misrepresentations due to forgotten descriptions. By deleting these descriptions, the sentences 2, 3 and 4 give different instructions under the same conditions. Therefore, the sentences 2, 3 and 4 are pairwise “inconsistent”.

With our tool, we enumerate the following elements from the document and then the set of these elements is the universe A of an argumentation framework (A, R). The format of the element is (id, a headline or a sentence in the document) where id is used for visualization purpose.

- [0, ’温度制御仕様’]
- [1, ’本ポットでは水温を制御するため、以下の仕様に従ってヒータを制御します’]
- [2, ’蓋が閉じられた場合’]
- [3, ’温度制御可能な水位ならば沸騰状態に移行し、ポット内の水を加熱します’]
- [4, ’水温が 100 ℃に達した後も 3 分間加熱を続け、その後保温状態に移行します’]
- [5, ’この場合、温度制御の操作量算出には目標温度 ON/OFF 方式を適用します’]

- [6, ’ 沸騰状態が終了した場合’]
- [7, ’ この場合、水温を 98 ℃に保つようにヒータを制御します’]
- [8, ’ 温度制御の操作量算出には PID 制御方式を適用します’]
- [9, ’ 沸騰状態が終了した場合’]
- [10, ’ この場合、水温を 90 ℃に保つようにヒータを制御します’]
- [11, ’ 温度制御の操作量算出には PID 制御方式を適用します’]
- [12, ’ 沸騰状態が終了した場合’]
- [13, ’ この場合、水温を 60 ℃に保つようにヒータを制御します’]
- [14, ’ 温度制御の操作量算出には PID 制御方式を適用します’]
- [15, ’ 沸騰ボタン押下により、強制沸騰する場合’]
- [16, ’ の制御仕様と同様です’]
- [17, ’ 保温設定ボタンにより、保温モードを変更した場合’]
- [18, ’ 移行した保温モードの制御仕様、、）に従い、目標温度に水温を保つようにヒータを制御します’]
- [19, ’ ※ 仕様毎の温度制御の操作量算出方法は、機種によって変わる場合があります’]

Next, we detect “inconsistent” pairs of elements of A to define an attack relation R of (A, R) with our tool. Our tool detect the following “inconsistent” pairs of elements of A.

- ([7, ’ この場合、水温を 98 ℃に保つようにヒータを制御します’], [10, ’ この場合、水温を 90 ℃に保つようにヒータを制御します’]),
- ([7, ’ この場合、水温を 98 ℃に保つようにヒータを制御します’], [13, ’ この場合、水温を 60 ℃に保つようにヒータを制御します’]),
- ([10, ’ この場合、水温を 90 ℃に保つようにヒータを制御します’], [7, ’ この場合、水温を 98 ℃に保つようにヒータを制御します’]),
- ([10, ’ この場合、水温を 90 ℃に保つようにヒータを制御します’], [13, ’ この場合、水温を 60 ℃に保つようにヒータを制御します’]),
- ([13, ’ この場合、水温を 60 ℃に保つようにヒータを制御します’], [7, ’ この場合、水温を 98 ℃に保つようにヒータを制御します’]),
- ([13, ’ この場合、水温を 60 ℃に保つようにヒータを制御します’], [10, ’ この場合、水温を 90 ℃に保つようにヒータを制御します’])

Our tool detects "inconsistent" pairs which we intentionally injected and does not detect "inconsistent" pairs that are not inherently "inconsistent". Thus, in this case, our fine-tuned BERT language model can detect "inconsistent" pairs of elements as expected. Moreover, in this case, the relation representing "inconsistent" is symmetric, namely if the pair of sentences n_1 and n_2 is "inconsistent" then the pair of sentences n_2 and n_1 is also "inconsistent".

Now, we have an argumentation framework (A, R) from the document. Our EMSS solver enumerates the following Maximal Satisfiable Subsets (MSS), that are preferred extensions in this case, of (A, R) in 0.278[sec]. Note that the documents consists of descriptions with id 1, 2, ..., 149 and $A = \{n1, n2, \ldots, n149\}$.

1. MSS [..., S(n6), S(n8), S(n9), S(n10), S(n11), S(n12), S(n14),...]
2. MSS [..., S(n6), S(n7), S(n8), S(n9), S(n11), S(n12), S(n14),...]
3. MSS [..., S(n6), S(n8), S(n9), S(n11), S(n12), S(n13), S(n14),...]

For instance, the result shows that the first MSS shows that the subset

$$\{n1, \ldots, n6, n8, n9, n11, n12, n13, n14, \ldots, n149\}$$

of A is a preferred extension which does not include $n7$ and $n10$.

Finally, for every above preferred extension, we construct a graph of the elements of A coloured light blue for the elements included in the above preferred extension and orange for the elements not included in the preferred extension. Note that the arrows between the elements represent the attack relation R.

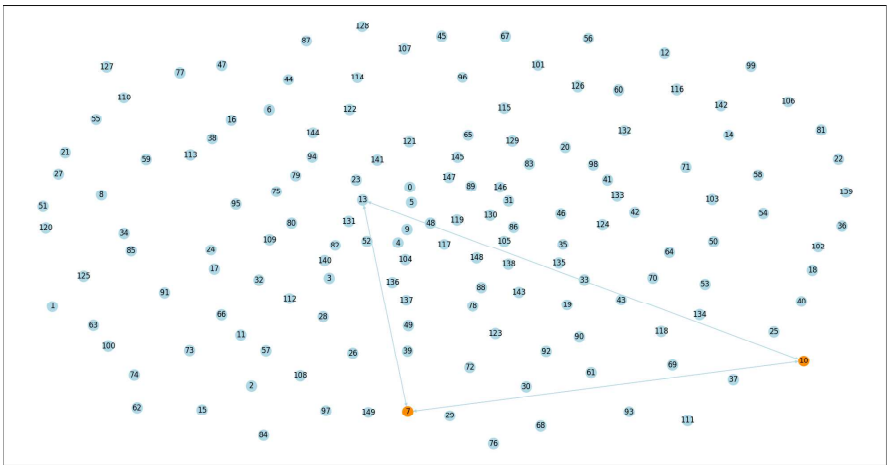

Figure 3: The graph of an argumentation framework

With the above graph of (A, R) and a MSS, we can support an engineer to resolve "inconsistency". For instance, the above graph shows of the first MSS shows that $n7$ (orange) and $n10$ (orange) are "inconsistent" with $n13$ (blue),

and $n13$ is in the MSS. Then, we accept $n13$ since it is in the MSS. And then, we find that $n7$ and $n10$ should be corrected referring to $n13 \to n7$ and $n13 \to n10$.

4.2 Experiments to verify scalability

In this subsection, we conduct an experiment to verify Scalability of our tool. We randomly generated argumentation framework (A, R). And then we enumerate preferred extensions as maximal satisfiable subsets of (A, R) with our tool.

We prepare some notations. $p(R)$ denotes the probability of the attack relation R. That is, for any two elements $a, b \in A$, we define whether $R(a, b)$ holds with probability $p(R)$. "Time" represents the time (in seconds) taken to enumerate the MSS's for a given (A, R). #(MSS) denotes the number of enumerated MSS's for a given (A, R). AS(MSS) is the average of the elements of the set $\{|S| \mid S \subseteq A, S \text{ is a MSS of } (A, R)\}$.

We assume some assumptions on the attack relation R of (A, R). We may assume that $p(R)(\approx |R|/|A|^2)$ is small enough. Because an engineer has a skill to write documents correctly. But they sometimes make a mistake. On the other hand, among the "inconsistencies", we now focus on misrepresentation. A sentence containing misrepresentation is not "inconsistent" with all other sentences. A single misrepresentation may only be "inconsistent" with the sentence containing the misrepresentation and its associated sentences. The sentence containing the misrepresentation is irrelevant to most other sentences. Thus, single misrepresentation yields a small number of "inconsistent" pairs of sentences. Moreover, misrepresentations-based attack relations are often symmetric,

We assume some assumptions on the size of (A, R). We assume that $|A| = 150$ since the size of the target document in our first experiment is 149. We also assume that one sentence containing misrepresentation causes symmetric attack relation with the other three relevant sentences. Moreover, we assume that one misrepresentation does not yields an attack relation with another misrepresentation. Then, one misrepresentation yields six attack relations. Thus, $|R|/6$ is the approximate number of misrepresentations. In the case of (A, R) such that $|A| = 150, p(R) = 0.01$, (A, R) contains approximately four misrepresentations since $|R| \approx |A|^2 \times p(R) \fallingdotseq 23$. Similarly, when $p(R) = 0.02$, it will contain approximately eight misrepresentations. Thus, we assume that $p(R) = 0.01$ or $p(R) = 0.02$.

We conduct two experiments for the case $p(R) = 0.01$ and $p(R) = 0.02$.

Table 1: $|A| = 150$ and $p(R) = 0.01$

$p(R)$	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Time [s]	1666	72	104	109	100	97	105	143	217	107
#(MSS)	1	1	1	1	2	1	1	2	2	1
AS(MSS)	37.0	72.0	76.0	70.0	74.0	74.0	74.0	69.5	72.0	76.0

Excluding outliers (the first case in Table 1 and the sixth case in Table 2), the average processing times are 117[s] ($p(R) = 0.01$) and 455[s] ($p(R) = 0.02$),

Table 2: $|A| = 150$ and $p(R) = 0.02$

$p(R)$	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
Time [s]	356	370	406	245	691	1246	581	391	631	420
#(MSS)	2	2	1	1	2	2	2	2	2	2
AS(MSS)	52.0	41.5	51.0	57.0	52.5	42.5	48.5	50.0	49.0	46.5

respectively. The first case in Table 1 takes longer to enumerate extensions than the others. At the same time, the case has an extremely small AS(MSS) compared to the others. Since a small $p(R)$ represents a small number of "inconsistent" pairs, in general, the size of the MSS tends to be larger. The case is therefore expected to be the case where the attack relations R are emergently related to each other. The sixth case in Table 2 also takes 1246[s], but the number and average size of MSSs are similar to the other cases. Thus, a detailed analysis of the structure of (A, R) is needed to clarify why only the sixth case takes particularly long to enumerate.

Table 3: $|A| = 150$ and R is defined randomly.

$p(R)$	0.03	0.04	0.05	0.10	0.15	0.20	0.25	0.30
Time [s]	514	1071	3048	3503	1520	2328	8849	4837
#(MSS)	1	4	2	4	0	2	6	4
AS(MSS)	42.0	33.5	33.0	17.75	0	11.5	9.57	8.0

Table 3 shows that as $p(R)$ increases, the Time is taken for enumerating extensions also tends to increase rapidly. However, as it can be assumed that the number of "inconsistencies" in a requirement specification document is small, the $|R|(\approx p(R))$ can be assumed to be small. In that case, the time required for the enumeration is feasible.

The results in Tables 1 and 2 show that when $|R|$ is sufficiently small, our tool can enumerate the extensions in practical time. On the other hand, Table 3 shows that when $|R|$ is not small, our tool cannot enumerate the extensions in practical time. However, it can be assumed that $|R|$ is small, and therefore our tool can enumerate extensions for a general requirements specification document in a realistic time.

5 Conclusion and Future Works

We propose a method and develop a tool to support for resolving "inconsistencies " in a requirement specification document which is written in a natural language. Experiments show that our tool is useful for supporting "inconsistency" resolution. On the other hand, our tool is not scalable with respect to the number of "inconsistencies" in a document. However, in general, it can be assumed that the number of "inconsistencies" is small, then our tool can support for resolving "inconsistencies " in realistic time.

One of our future works is to detect a various “inconsistencies”. In this paper, we trained our language model by SNLI dataset. But the dataset only contains contradictions as “inconsistencies”. Training a language model that can detect strong and weak safety sentences as “inconsistent” pairs requires a large number of such pairs with annotations.

Another future works is to improve the accuracy of “inconsistency” determination by NLP. Our language models are trained on a huge amount of generic documents. However, requirements specification documents often contain words that are not included in general documents, and these words often have important meanings. For example, product names are often combinations of nouns, numbers, and alphabets, and engineers often get product names wrong due to typos. And if the product name is wrong, the meaning of a specification is completely different. To solve this problems, it is necessary to use Japanese language processing-specific techniques to recognize nouns + numerals and alphabets as a single word, or to perform pre-training and fine-tuning of language models using a large number of requirement specification documents.

This work is a joint work with Hiroyuki Kido (Cardiff University) and Toshinori Takai (Nara Institute of Science and Technology). This work was supported by JSPS KAKENHI Grant Number JP19K11914. This work was supported by the Research Institute for Mathematical Sciences, an International Joint Usage/Research Center located in Kyoto University.

References

[1] 自然言語処理の応用による要求仕様書中の矛盾検出手法の提案, 穀田一真, 岡本圭史, 情報処理学会第 84 回全国大会 (2022)

[2] Expressing Dung’s Extensions as FO-Formulas to Enumerate Them with an SMT Solver, Keishi Okamoto, Hiroyuki Kido, Toshinori Takai, RIMS Kokyuroku, 2170 Model theoretic aspects of the notion of independence and dimension, pp.64-72 (2020)

[3] Enumeration of Dung’s Extensions with an SMT Solver, Keishi Okamoto, Hiroyuki Kido and Toshinori Takai, RIMS Model Theory Workshop 2020 (2020)

[4] The Institute of Electrical and Electronics Engineers (IEEE), “830-1998 – IEEE recommended practice for software requirements specifications” (1998)

[5] 要求仕様の品質特性, 大西淳, 佐伯元司, 情報処理 Vol.49 No.4 Apr. (2008)

[6] Z3 Prover, https://github.com/Z3Prover/z3/wiki

[7] 数理議論学，若木利子，新田克己，東京電機大学出版 (2017)

[8] Model Theory (Encyclopedia of Mathematics and its Applications), Wilfrid Hodges, Cambridge University Press (1993)

[9] Handbook of Satisfiability, Armin Biere, Marijn Heule, Hans Van Maaren, Toby Walsh (Ed.), IOS Press (2009)

[10] On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, Phan Minh Dung, Artificial Intelligence 77, pp.321-357 (1995)

[11] Checking the acceptability of a set of arguments, Philippe Besnard and Sylvie Doutre, NMR 2004: Whistler, Canada (2004)

[12] Enumerating Infeasibility: Finding Multiple MUSes Quickly, Mark H. Liffiton and Ammar Malik, In: Gomes C., Sellmann M. (eds) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. CPAIOR 2013. Lecture Notes in Computer Science, vol 7874. Springer, Berlin, Heidelberg. (2013)

[13] Pretrained Japanese BERT models, `https://github.com/cl-tohoku/bert-japanese`

[14] 日本語 SNLI(JSNLI) データセット, https://nlp.ist.i.kyoto-u.ac.jp/index.php?日本語 SNLI(JSNLI) データセット

[15] Jacob Devlin, et. al., BERT: Pre-training of deep bidirectional transformers for language understanding. arXivpreprintarXiv:1810.04805, 2018

[16] 話題沸騰ポット要求仕様書 (GOMA-1015 型) 第 6 版, SESSAME, `https://www.sessame.jp/workinggroup/WorkingGroup2/POT_Specification_v6.PDF`