Enumeration of Dung's Extensions with an SMT Solver

Keishi Okamoto
National Institute of Technology, Sendai College

Abstract

In this paper, we propose an approach to enumerate Dung's extensions by solving a Partial Maximal Satisfiable Subsets Enumeration problem which is an extension of a Maximal Satisfiable Subsets Enumeration problem[1]. More precisely, based on results in [2], we describe hard and soft constraints, required to enumerate extensions, in the input language of an SMT solver Z3[3]. Then we solve the problem with a solver by combining a MSSEn solver of [1] and Z3. Finally, we conduct experiments to enumerate extensions.

1 Introduction

Software-intensive product defects are often caused by inadequate software specifications. Therefore, it is important to improve the quality of software specifications. In IEEE830-1998[4], 8 quality attributes of requirement specification documents are defined as follows: correctness, unambiguity, completeness, consistency, ranked for importance and/or stability, verifiability, modifiability and traceability. There are research to improve these quality attributes. In this paper, we focus on consistency.

Consistency is a quality attribute that the individual requirements do not contradict each other. In [5] the authors showed following 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, and [Inconsistent Constraints] there are constraints, but there is no solution that satisfies the constraints.

A traditional method to detect inconsistencies in specification documents is reviewing. A reviewer manually detect inconsistencies in specification documents. On the other hands, there are automatic detection methods, for instance formal methods. As formal methods are based on mathematical logic, we can detect logical contradiction $A \wedge \neg A$ for propositions "A: the initial value of the variable x must be set to 0" and "$\neg A$: the initial value of the variable x must not be set to 0".

There are also inconsistencies that are not logical contradictions. For instance, consider the following two sentences A_1 and A_2 where "A_1: if the value of the water temperature sensor ≥ 90°C, transit from the heating mode to the heat retention mode", "A_2: if the value of the water temperature sensor is ≥ 95°C, transit from the heating mode to the heat retention mode". A_1 and A_2 are not logically contradict. A_1 requires more safety than A_2 since A_1 implies A_2 but not vice versa. If we formalize the description language in detail, we can detect this inconsistency with formal methods. But such detailed formalization yields complicated and huge formal specification documents and then detecting inconsistencies in the documents will be infeasible.

Our future goal is to propose a method to support for resolving inconsistencies in specification documents based on mathematical argumentation theory, and then is to implement the porpoising method. In particular, we suggest acceptable consistent set of descriptions in a specification document as Dung's extensions [6] of mathematical argumentation theory. Once a developer has acceptable consistent sets of descriptions, they will be able to choose the most acceptable set and fix the other descriptions to resolve inconsistencies. Based on mathematical argumentation theory, proposing tool will have the following advantages. First, our tool will be able to help engineers to resolve various inconveniences, including logical contradiction and others as various inconsistencies can be formulated as a binary relation, called an attack relation, of a argumentation framework. Second, our tool will be able to help engineers to resolve inconsistencies in a large specification document since inconsistencies are abstracted as an attack relation.

In this paper, we propose a method to enumerate Dung's extensions [6] by solving a Partial Maximal Satisfiable Subsets Enumeration (PMSSEn) problem that is an extension of a Maximal Satisfiable Subsets Enumeration problem[1]. More precisely, we describe hard and soft constraints to enumerate extensions in the input language of Z3. Then we propose a PMSSEn(SMT) solver by combining the MSSEn solver of [1] and a SMT solver Z3. Then we conduct an experiment to enumerate extensions with the PMSSEn(SMT) solver.

The structure of this paper is as follows. In Section 2, we introduce definitions of Dung's extensions of argumentation theory. Moreover, we also show an idea to resolve inconsistencies in a specification documents by enumerating extensions. In Section 3, we describe Dung's extensions of an argumentation frameworks (A, R) in the input language of Z3. In Section 4, we mention a PMSSEn(SMT) solver and experimental results. Finally we make some concluding remarks and future works in Section 5.

2 Preliminaries

In this section, we introduce the mathematical foundations to enumerate extensions with an SMT solver. First, we introduce definitions of Dung's extensions of argumentation theory. Second, we review the constraints to enumerate extension in [2].

2.1 Dung's Extensions

In this subsection, we introduce definitions of Dung's extensions of argumentation theory. Moreover, we show an idea to resolve inconsistencies in a specification documents by enumerating extensions.

Argumentation theory is an interdisciplinary field that has been studied interdisciplinary in many fields, such as sociology, linguistics, psychology, logic, dialectics on the issue of "how to justify a claim". And, mathematical argumentation theory is argumentation theory based on mathematical methods In mathematical argumentation theory, an argumentation framework (A, R) is a directed graph such that A is a set of abstract arguments and R is an attack relation. For any $a, b \in A$, $R(a, b)$ means that "if a is true then b must be false". We say that "a attacks b" if $R(a, b)$ holds. A subset S of A is consistent if there are no $a, b \in S$ such that $R(a, b)$ holds. An extension is a subset of A that is consistent and has certain properties. Intuitively, an extension is a set of acceptable arguments. We only show the definition of stable extension. Please see [6] for the definitions of other extensions.

Definition 1 (Conflict-Free Subset, Stable Extension) *A subset S of A is conflict-free if there are no two arguments a and b such that $R(a, b)$. A subset S of A is a stable extension if it is conflict-free and, for each argument $a \in A$, if $a \notin S$ then there exists an argument $b \in S$ such that $R(b, a)$.*

A argumentation framework (A, R) can be considered as a specification document, where an argument is a description in a specification document (e.g., a sentence, a diagram, a logical formula, etc.) and an attack relation represents inconsistencies in the specification document (e.g., a sentence a logically contradicts to a sentence b). Then, an extension S of (A, R) can be considered as an acceptable set of descriptions. Thus, S is consistent and trustworthy.

We show an example of resolving inconsistencies in a specification document by enumerating extensions. Let (A, R) be an argumentation framework of a specification document where $A = \{a, b, c, d\}$. Moreover, assume that subsets $\{a, c\}$ and $\{a, d\}$ of A are extensions that are acceptable subsets of A. If you accept $\{a, c\}$ then the other arguments b and d must be corrected because b and d are inconsistent with an argument in $\{a, c\}$.

2.2 Constraints to Enumerate Extensions

In this subsection, we review the constraints to enumerate extension in [2]. We introduce three kinds of soft constraints, namely soft constraints to enumerate maximal satisfiable subsets, soft constraints to enumerate minimal satisfiable subset and soft constraints to enumerate satisfiable subset.

Let $G(A, R)$ be the set of FO-formulas representing an argument framework (A, R), namely an FO-formula representing an enumeration of A and the set $\{R(a, b) \mid (a, b) \in R\}$.

2.2.1 Preferred Extensions

We generally extract a maximal subset S of A satisfying a FO-formula $\Psi(S)$ by extracting a Maximally Satisfiable Subset (MSS) of a hard constraint $\Psi(S)$ and soft constraints $S(a_1), S(a_2), \ldots, S(a_n)$ where $A = \{a_1, a_2, \ldots, a_n\}$. Since a preferred extension is a maximal subset S of A satisfying a FO-formula $\mathbf{CF}(S) \wedge \mathbf{AS}(S, f_2)$, we enumerate preferred extensions by enumerating MSS's for

- Hard Constraints: $\mathbf{CF}(S) \wedge \mathbf{AS}(S, f_2)$, $G(A, R)$ and
- Soft Constraints: $S(a_1), S(a_2), \ldots, S(a_n)$ where $A = \{a_1, a_2, \ldots, a_n\}$

with a PMSSEn(SMT) solver.

We show an example of enumerating preferred extensions whose soft constraints $S(a), S(b), S(c), S(d), S(e)$. Assume that a PMSSEn(SMT) solver returns a result including a subset $\{S(a), S(d)\}$ of the set $\{S(a), S(b), S(c), S(d), S(e)\}$ of all the soft constraints. The resulting subset shows that the subset $S = \{a, d\}$ of A is a maximal subset satisfying the hard constraint $\mathbf{CF}(S) \wedge \mathbf{AS}(S, f_2)$. Thus the set $\{a, d\}$ is a maximal admissible subset, namely a preferred extension.

2.2.2 Grounded Extension

The grounded extension is a minimal subset of A satisfying a FO-formula $\mathbf{AS}(S, f_2) \wedge \mathbf{CE}(S, f_3)$, then it is the complement of a maximal subset $\bar{S}$ of A satisfying the negation of $\mathbf{AS}(\bar{S}, f_2) \wedge \mathbf{CE}(\bar{S}, f_3)$. Thus we extract the grounded extension from (A, R) by enumerating MSS's for the following constraints:

- Hard Constraints: $\mathbf{AS}(S, f_2) \wedge \mathbf{CE}(S, f_3)$, $G(A, R)$ and
- Soft Constraints: $\neg S(a_1), \neg S(a_2), \ldots, \neg S(a_n)$ where $A = \{a_1, a_2, \ldots, a_n\}$.

with a PMSSEn(SMT) solver. We note that the result of enumeration of the above MSS's is a single subset by the definition of the grounded extension.

We show an example of extracting the grounded extension whose soft constraints $\neg S(a), \neg S(b), \neg S(c), \neg S(d), \neg S(e)$. Assume that a PMSSEn(SMT) solver returns a result including a subset $\{\neg S(b), \neg S(c), \neg S(d), \neg S(e)\}$ of the set $\{\neg S(a), \neg S(b), \neg S(c), \neg S(d), \neg S(e)\}$ of all the soft constraints. The resulting subset shows that the subset $S = \{a\}$ of A is a minimal subset satisfying the hard constraint $\mathbf{AS}(S, f_2) \wedge \mathbf{CE}(S, f_3)$. Thus the set $\{a\}$ is a minimal complete extension, namely the grounded extension.

2.2.3 Stable Extensions and Complete Extensions

Since a stable extension is a subset S of A satisfying a FO-formula $\mathbf{CF}(S) \wedge \mathbf{SE}(S, f_1)$, we enumerate stable extensions by enumerating MSS's of

- Hard Constraints: $\mathbf{CF}(S) \wedge \mathbf{SE}(S, f_1)$, $G(A, R)$ and

- Soft Constraints: $S(a_1), S(a_2), \ldots, S(a_n), \neg S(a_1), \neg S(a_2), \ldots, \neg S(a_n)$ where $A = \{a_1, a_2, \ldots, a_n\}$

with a PMSSEn(SMT) solver. We enumerate complete extensions in a similar way.

We show an example of enumerating stable extensions whose soft constraints $S(a), S(b), S(c), S(d), S(e), \neg S(a), \neg S(b), \neg S(c), \neg S(d), \neg S(e)$. Assume that a PMSSEn(SMT) solver returns a result including a subset $S(a), \neg S(b), \neg S(c), S(d), \neg S(e)$ of the set of all the soft constraints. The resulting subset shows that the subset $S = \{a, d\}$ of A is a subset satisfying the hard constraint $\mathbf{CF}(S) \wedge \mathbf{SE}(S, f_1)$. Thus the set $\{a, d\}$ is a stable extension.

3 Modeling Extensions with an SMT Solver Z3

In this section, we describe hard and soft constraints of extensions in the input language of Z3 to enumerate extensions with Z3. First, we declare some symbols to describe an argumentation framework (A, R) and constraints. Second, we describe hard constraints of the attack relation R and extensions. Finally, we describe soft constraints of extensions.

3.1 Declaration of (A, R), Extensions and Skolem Functions

We declare a set A by enumerating elements of A and an attack relation R as a function. We also declare an extension S that is a subset of A as a membership function. We declare the function S as an uninterpreted function so that an SMT solver can find an realization of S [7].

```
# Node = A in <A,R>
Node, (a, b, c, d, e) = EnumSort('Node',('a','b','c','d','e'))
# Atk(a,b)==True <=> a attacks b
Atk = Function('Atk', Node, Node, BoolSort())

# To extract an extension S,
# we define a membership function S:Node->Bool.
S = Function('S', Node, BoolSort()) # S(a)==True <=> a is in S
```

Formulas of the form $\forall x \exists y \varphi(x, y)$ is required to describe admissible extensions, complete extensions and stable extensions. As a formula of the form is hard to solve with an SMT solver, we translate it to a logically equivalent formula $\forall x \varphi(x, f(x))$ with a Skolem function f. We declare Skolem functions as uninterpreted functions.

```
# We use the following constants and functions
# to describe formulas with quantifiers.
x, y  = Consts('x y', Node)
```

```
# Skolem function for "Admissible Extension"
Skolem1 = Function('Skolem1', Node, Node)
# Skolem function for "Complete Extension"
Skolem2 = Function('Skolem2', Node, Node, Node)
# Skolen function for "Stable Extension"
Skolem3 = Function('Skolem3', Node, Node)
```

3.2 Hard Constraints of R and Extensions

We describe an attack relation R of (A, R) as hard constraints. More precisely, we define whether an element a attacks an element b for any $a, b \in A$.

```
Atk(a, a)==False,
Atk(a, b)==True,  # means that "a" attacks "b".
Atk(a, c)==False, # means that "a" does not attacks "c".
Atk(a, d)==False,
Atk(a, e)==False,
...
```

As we have describe FO-formulas of extensions [2], we translate some of the FO-formulas to hard constraints in Z3.

- **CF**(S): ForAll([x, y], Implies(And(S(x), S(y)), (Not(R(x,y)))))
- **SE**(S): ForAll([x], Or(S(x), And(S(Skolem3(x)),R(Skolem3(x),x))))
- **AS**(S): ForAll([x, y], Implies(And(S(x), R(y,x)),
  And(S(Skolem1(y)),R(Skolem1(y),y)))),
- **CE**(S): ForAll([x],
  Implies(Or(Not(R(Skolem3(a),a)),And(S(c),R(c,Skolem3(a)))),S(a)),

3.3 Soft Constraints of Extensions

A preferred extension is a MSS of the hard constraint $\mathbf{CF}(S) \wedge \mathbf{AS}(S, f_2)$ and soft constraints $S(a)$ for any element a of A. The grounded extension is the MSS of the hard constraint $\mathbf{AS}(S, f_2) \wedge \mathbf{CE}(S, f_3)$ and soft constraints $\neg S(a)$ for any element a of A. A stable extension can be a MSS of the hard constraint $\mathbf{CF}(S) \wedge \mathbf{SE}(S, f_1)$ and soft constraints $S(a), \neg S(a)$ for any element a of A. We only describe the soft constraints of a stable extension where $A = \{a, b, c, d, e\}$.

```
S(a), # S(a)==True <=> a is in S
Not(S(a)),
S(b),
Not(S(b)),
S(c),
Not(S(c)),
S(d),
```

```
Not(S(d)),
S(e),
Not(S(e)),
```

4 Experiments: Enumerating Extensions

In this section, we first review a MSSEn Solver in [1] and extend it to a partial MSSEn solver for constraints in FOL to enumerate Dung's extensions. Second, we show the correctness of our modeling in Section 3 and the proposing solver by showing that our solver generate the same result in Example1 of [8]. Finally, we show some experimental results to enumerate extensions.

4.1 PMSSEn(SMT) Solver

In this subsection, we mention a Partially Maximally Satisfiable Subsets Enumeration (PMSSEn) solver for constraints in FOL. A MSSEn solver is proposed in [1]. With an SMT solver, the MSSEn solver can solve a MSSEn problem for constraints in FOL. On the other hand, the MSSEn solver can be extended to a Partial MSSEn solver by dividing constraints into hard constraints and soft constraints. Then, we have a Partial MSSEn solver for constraints in FOL by combining the MSSEn solver and the SMT solver Z3[3]. We call it a PMSSEn(SMT) solver. Thus, we can enumerate extensions by solving a Partial MSSEn problem with the PMSSEn(SMT) solver since we have described hard constraints and soft constraints as FO-formulas to enumerate extensions.

4.2 Correctness of our Modeling and Solver

In this subsection, we show the correctness of our modeling in Section 3 and the PMSSEn(SMT) solver by showing that our solver generate the same result in Example1 of [8].

Example 2 (Example 1 [8]) *Let* (A, R) *be the argument framework where* $A = \{a, b, c, d, e\}$ *and* $R = \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, e)\}$. *Then we have the following extensions:*

- *Stable extension(s):* $\{a, d\}$,
- *Preferred extensions:* $\{a, c\}, \{a, d\}$,
- *Complete extensions:* $\{a, c\}, \{a, d\}, \{a\}$ *and*
- *Grounded extension:* $\{a\}$.

We describe the above argumentation framework (A, R) and then enumerate extensions of (A, R) with the PMSSEn(SMT) solver. Thus, the solver returns the same results as in [8]. We only show the result of enumeration of complete extensions.

```
 # means that {a,d} is a complete extension.
MSS [S(a), Not(S(b)), Not(S(c)), S(d), Not(S(e))]
 # means that {a} is a complete extension.
MSS [S(a), Not(S(b)), Not(S(c)), Not(S(d)), Not(S(e))]
 # means that {a,c} is a complete extension.
MSS [S(a), Not(S(b)), S(c), Not(S(d)), Not(S(e))]
```

4.3 Experimental Results

In this subsection, we show some experimental results to enumerate extensions. Preferred extensions can be enumerated as a maximal subsets satisfying certain property in FOL. The grounded extension can be extracted as the unique minimal subset satisfying certain property in FOL. Stable extensions and complete extensions can be enumerate as subsets satisfying certain property in FOL. Indeed, as stable extensions and complete extensions can be enumerated by solving satisfiable subset enumeration problems, enumerating preferred extensions is the most difficult problem to solve [9]. Thus, we show the result of an experiment to enumerate preferred extensions in a argumentation framework (A, R).

We define the attack relation R using random numbers. More precisely, we assume that $R(a, b)$ holds for the two elements a and b of A with a probability of 50%. Moreover, we also assume that there are no element a of A such that $R(a, a)$ holds. Because an element a represents a self-contradictory description in a specification document when $R(a, a)$ holds, while real specification documents may have self-contradictory descriptions.

We assume that $|A| = 100$ in the first experiment. The PMSSEn(SMT) solver returns seven preferred extensions in 1387.30099988 seconds.

```
# Experimental Result
# MSS [S(n37), S(n21), S(n25), S(n30), S(n74), S(n97)]
# MSS [S(n20), S(n49), S(n87), S(n94), S(n1)]
# MSS [S(n3), S(n35), S(n53), S(n58)]
# MSS [S(n53), S(n80), S(n20), S(n33), S(n95)]
# MSS [S(n71), S(n91), S(n38), S(n78)]
# MSS [S(n63), S(n87), S(n94), S(n1), S(n49)]
# MSS [S(n32), S(n63), S(n82), S(n99), S(n45)]
# elapsed_time:1387.30099988[sec]
```

We assume that $|A| = 150$ in the second experiment. The PMSSEn(SMT) solver returns five preferred extensions in 6589.398 seconds.

```
# Experimental Result
# MSS [S(n0), S(n56), S(n93), S(n12), S(n114)]
# MSS [S(n129), S(n42), S(n53), S(n79), S(n106), S(n148)]
# MSS [S(n6), S(n48), S(n116), S(n139), S(n1), S(n72)]
# MSS [S(n87), S(n124), S(n136), S(n73)]
# MSS [S(n30), S(n96), S(n136), S(n8), S(n89), S(n119)]
# elapsed_time:6589.398[sec]
```

5 Concluding Remarks and Future Works

We proposed a way to model Dung's extensions in the input language of an SMT solver Z3. We also proposed a PMSSEn(SMT) solver by combining the MSSEn solver in [1] and Z3. Then we show the correctness of our modeling and the PMSSEn(SMT) solver. This way of modeling and the PMSSEn(SMT) solver can support developers to resolve inconsistencies in specification documents. A challenge is to show the validity of our modeling and the PMSSEn(SMT) solver. Thus, our future work is to enumerate extensions from the argumentation framework that corresponds to a real specification document.

An issue is to enumerate extensions of an argumentation framework (A, R) such that $|A|$ is large. With a randomly generated argumentation framework (A, R) where $|A| = 100$, the PMSSEn(SMT) solver completed the enumeration in a realistic amount of time. However, in a randomly generated argumentation framework whose (A, R) where $|A| = 150$, the time required for enumeration increased sharply. As the number of constraints required to enumerate extensions is proportional to $|A|^2$, it is presumed that both the search cost of one maximal subset (the size of a maximal subset) and the cost of enumerating the maximum subsets (the number of maximal subsets) increased in proportion to $|A|^2$.

Another issue is to construct an argumentation framework (A, R) of a real specification document and conduct an experiment to enumerate extensions of (A, R). We plan to use a method such as natural language processing to estimate R from a real specification document. In a real specification document, the developer describe specifications while paying attention to consistency, then it is natural to assume that the generated argumentation framework has very few elements to be attacked. On the other hand, the smaller the number of attacked elements of A is, it will be more easy to find an extension (a maximal consistent subset).

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.

References

[1] 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)

[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) http://hdl.handle.net/2433/261556

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

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

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

[8] Checking the acceptability of a set of arguments, Philippe Besnard and Sylvie Doutre,

[9] Computing Preferred Extensions in Abstract Argumentation: A SAT-Based Approach, Federico Cerutti, Paul Dunne, Massimiliano Giacomin and Mauro Vallati, In: Black E., Modgil S., Oren N. (eds) Theory and Applications of Formal Argumentation. TAFA 2013. Lecture Notes in Computer Science, vol 8306. Springer, Berlin,Heidelberg (2014)