Dialogue Protocols for Argumentation Frameworks with Sets of Attacking Arguments

Wei Chen

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Master of Science in Artificial Intelligence
of the
University of Aberdeen.

1 4 9 5

Department of Computing Science

August 2021

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed: Wei Chen

Date: August 2021

Abstract

Motivation and Aim: SETAFs is a newly developed framework that extends the Abstract Argument Framework (AAF) study in Artificial Intelligence. Although the research on the proof dialogue of AAF is rich, however, for now, few studies focus on the application of proof dialogue in SETAF. Therefore, this project aims to study how existing proof dialogue can be extended in the context of SETAFs.

Method: For the theoretical methodology, this paper extended the theoretical framework from AAFs to SETAFs and proved the reliability of the dialogue protocols for the semantics in SETAF. Then, this paper further developed a web application to run the proof dialogue games for an intuitive understanding of argumentation based on the algorithm of SETAF. Finally, this thesis conducted self-evaluation and ran a survey to collect feedback from public users.

Result: This study successfully explains the SETAF concept and operates the dialogue games according to public users' feedback. However, potential threats might deteriorate the accuracy or the robustness of the game result when conducting self-evaluation.

Conclusion: This thesis is the first attempt to fill the gap in previous research that lacks proof dialogue based on SETAFs, which could develop the application in the argumentation studies.

Acknowledgements

I would like to thank the support from my supervisor Dr. Bruno Yun for inspiring me with his ideas about the the project. I could not forget my family and university friends, who made the last year an inspiring adventure, filled with shared efforts and laugh. At last, I would like to thank the department of Computer Science, University of Aberdeen for cultivating me over the past year.

Contents

Chapter 1

Introduction

1.1 Introduction

Argumentation is the cornerstone of the development of modern civilised society. Chris Reed argued in his blog that the process of argumentation underpins government, science and religion [1]. As technology reshapes our lives, humans are waking up to its importance. In the era of news and information explosion, it is difficult to verify the authenticity and accuracy of information because of the fake news that flood the Internet world. Faced with such challenges, critical thinking has become a must. Using argumentation techniques, computers are now able to argue critically, i.e. they can tell right from wrong quickly.

In the field of Artificial Intelligence(AI), this technique is an argumentation framework that is a method to handle controversial information and draw the conclusion from it applying formalised arguments. Previously, researchers employed the Abstract Argumentation Framework (AAF) initially introduced by Dung as the principle of dealing with contentious information. This framework focus on a binary relation on the set of arguments which represents conflicts between arguments.

However, although Dung's AAF shows success in evaluating the acceptability of arguments, it lacks flexibility in complex interaction in controversial information. Therefore, recently, a generalisation of Dung's AAF that allows joint attacks has gained a lot of popularity (Yun et al., 2020a,b; Flouris and Bikakis, 2019; Dvorák et al., 2018). This framework, called SETAF, was introduced by Nielsen and Parsons (2006) as they argued that it allows for more complex interactions. As a result, many argumentation semantics and techniques were extended to the context of SETAFs.

Nevertheless, there has been no work on extending the existing proof dialogue to this new framework. Thus, this thesis provides the first trial to fill the gap in the current study that lacks proof dialogue to this new framework, which could contribute to developing the study of SETAFs in the study of argumentation. Besides, this research attempts to create a dialogue game, which could be an example of an application for argumentation in controversial information and thus could save time to argue critically.

[1]https://theconversation.com/how-ai-can-make-us-better-at-arguing-85938

1.2 Research Goals

The main research goal of this project is to study how existing proof dialogue can be extended in the context of SETAFs. Thus, this paper splits this into the following sub-goals:

G1 is to extend the proof dialogue protocols for several semantic for SETAFs and proofs of correctness;

G2 is to implement an online application that can take as input a SETAF graph in the ASPARTIX format and interact with the user using a dialogue;

G3 is to evaluate the proposed application.

1.3 Thesis Structure

The remainder of this thesis is structured as follows. Chapter 2 introduces the theoretical background of AAFs, including the definitions of Dung's AAFs, argumentation semantics, labelling method and dialogue protocols for grounded, preferred and stable semantics in AAFs. Chapter 3 recalls the definition of SETAFs, and creates the labelling method with SETAFs and dialogue protocols for grounded, preferred and stable semantics in SETAFs. Chapter 4 displays the web tool's project setting and the essential ideas to implement three kinds of dialogue protocols for SETAFs using Typescript. Chapter 5 analyses and summarises self-evaluation and public evaluation, respectively. Chapter 6 concludes with a summary, discusses issues in the implementation and suggests future work following this thesis.

Chapter 2

Background and Literature Review

Since SETAF is based on the framework of AAFs, this chapter will provide the background and literature review related to AAF. Furthermore, to better understand the argumentation framework, this chapter will also introduce the theoretical background of Dung's AAFs, including the definitions of Dung's AAFs, argumentation semantics, labelling method, and dialogue protocols in AAFs.

The AI community's interest in argumentation started in 1995 with Dung's seminal paper (Dung, 1995). He introduced the Abstract Argumentation Framework (AAF) as a clear and intuitive formalism to represent arguments and the relationships between arguments or attacks using direct graphs.

Due to the clarity and generality of this framework, it has been widely used in real-world systems, in multiple domains ranging from medical diagnosis to model human decision making, to solve tasks such as non-monotonic reasoning and decision making (McCarthy, 1980; Sapino et al., 2020; Strasser and Antonelli, 2019; Siegel, 1954).

From an AAF, one can use Dung's original argumentation semantics (complete, stable, preferred, and the grounded semantics) to compute sets of justified arguments (Dung, 1995). Please note that other argumentation semantics such as stage semantics (Caminada, 2011), and semi-stable semantics (Caminada, 2006) will be introduced later.

All of those aforementioned argumentation semantics are based on the notion of acceptability, which means the sets of justified arguments returned can defend themselves against other arguments; those justified sets of arguments represent *consistent viewpoints* on the argumentation framework. However, it can be argued that simply providing those sets of justified arguments is not intuitive to the end-user. Therefore, *proof dialogues* are proposed to explain if an argument belongs (or not) to one or all of the justified sets of arguments according to some semantics.

Proof dialogue is usually a sequence of utterances/moves between two agents: the *proponent* and the *opponent*. The proponent wants to show that an argument is justified, while the opponent wants to show that it is not. A set of multiple constraints, called the *protocol* , is put in place to govern the agents' behaviour, specify who begins the dialogue and what are the conditions for the dialogue to end, and ensure the correctness of the dialogue (Modgil and Caminada, 2009).

Multiple proof dialogues have been defined in the literature for Dung's semantics, such as preferred dialogue, stable dialogue, etc (Devred and Doutre, 2007; Cayrol et al., 2001; Vreeswijk and Prakken, 2000; Caminada and Wu, 2009). Other works have tried to propose a general setting for dialogue protocols by identifying modular parameters (Amgoud et al., 2006). The summary of proof dialogue theories for AAF could be referred to Caminada's work (Modgil and Caminada, 2009).

However, to the best of the author's knowledge, few studies have currently set proof dialogue using SETAF. Thus, this is the first paper to generate proof dialogue employing SETAF and will contribute to filling the gap of application related to multi-agent argumentation.

The following sections 2.1-2.4 in this chapter will introduce the detailed and formal definition of the terms mentioned above following Dung's study.

2.1 Dung's Abstract Argumentation Framework

Dung introduces a framework to represent arguments and their interactions as a directed graph (Dung, 1995). Definition 1 will recall his Abstract Argumentation Framework in a formal way.

Definition 1 (Abstract Argumentation Framework) *An Abstract Argumentation Framework (AAF) is a pair $\mathcal{F} = (\mathcal{A}, \mathcal{C})$, where $\mathcal{A}$ is a set of arguments and $\mathcal{C} \subseteq \mathcal{A} \times \mathcal{A}$ is a set of attacks.*

Given $A, B \in \mathcal{A}$, we say that A attacks B (or B is attacked by A) if $(A, B) \in \mathcal{C}$.

Example 1 *We can represent this situation with the AAF $\mathcal{F} = (\mathcal{A}, \mathcal{C})$, where $\mathcal{A} = \{A, B\}$ and a single attack $\mathcal{C} = \{(A, B)\}$. This AAF is represented in Figure 2.1.*

A: "Will is a dog and he flies."

B: "Dogs don't fly."

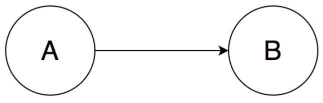


Figure 2.1: Representation of the AAF $\mathcal{F}$

An AAF is a directed graph whose nodes correspond to arguments and whose arcs correspond to attacks. The attacking relation allows for arguments to invalidate other arguments. According to Dung, the way humans argue follows a simple principle which is illustrated by the old saying *The one who has the last word laughs best*. For example, in Example 1, the winning argument is A as it invalidates B. Hence, the winning arguments are called *justified arguments*. Dung's original argumentation semantics was previously applied to decide whether an argument is acceptable or not (see in section 2.2). However, a more expressive method, namely, *Labelling*, to decide the acceptance was developed. The Labelling method will be introduced in section 2.3.

2.2 Argumentation Semantics

To compute sets of justified arguments, one can use Dung's original argumentation semantics (complete, stable, preferred, and the grounded semantics) from an AAF. All of those argumentation semantics is based on the notion of acceptability, i.e. the sets of justified arguments returned can defend themselves against other arguments. As a result, those justified sets of arguments represent *consistent viewpoints* on the argumentation framework.

Followed by Dung's study, this paper will explain the following semantics by referring to an arbitrary but fixed argumentation framework $\mathcal{F} = (\mathcal{A}, \mathcal{C})$.

Definition 2 (Conflict-free Extension) *A set of arguments $S \subseteq \mathcal{A}$ is a conflict-free extension iff there is no argument $A, B \in S$ such that A attacks B.*

Example 2 *Given in Figure 2.2, $\mathcal{F} = (\mathcal{A}, \mathcal{C})$ where $\mathcal{A} = \{A, B, C\}$, $\mathcal{C} = \{(A, B), (B, C)\}$. The set of all conflict-free extensions is $\{\emptyset, \{A\}\{B\}\{C\}\{A, C\}\}$.*

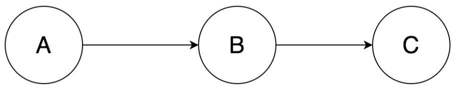


Figure 2.2: AAF with 3 Arguments

Definition 3 (Acceptability) *We say that $A \in \mathcal{A}$ is acceptable w.r.t. $S \subseteq \mathcal{A}$, iff for every $B \in \mathcal{A}$ that attacks A, there exists $C \in S$ such that $(C, B) \in \mathcal{C}$.*

Definition 4 (Admissible Extension) *A set of arguments $S \subseteq \mathcal{A}$ is an admissible extension iff S is conflict-free and each argument in S is acceptable with respect to S.*

Example 3 *Given in Figure 2.3, $\mathcal{F} = (\mathcal{A}, \mathcal{C})$ where $\mathcal{A} = \{A, B, C, D, E\}$, $\mathcal{C} = \{(A, C), (B, D), (C, E), (D, E)\}$. It is easy to see that E is acceptable w.r.t {A,B} because E is defended by A and B at the same time. Then $\{A, B, E\}$ is one set of all admissible extension. Please note that $\{A, E\}$ and $\{B, E\}$ also are admissible extension.*

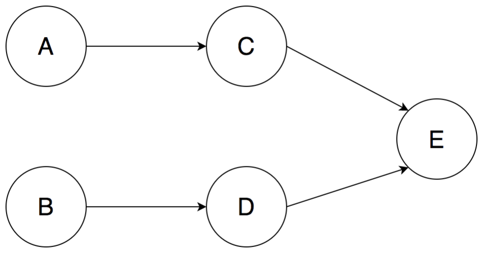


Figure 2.3: AAF with 5 Arguments

Definition 5 (Complete Extension) *An admissible set S is called a complete extension, if all arguments that are acceptable with respect to S are in S.*

Example 4 *Given in Figure 2.4,* $\mathcal{F} = (\mathcal{A}, \mathcal{C})$ *where* $\mathcal{A} = \{A, B, C, D, E\}$, $\mathcal{C} = \{(A,B), (B,A), (B,C), (C,D), (D,E), (E,C)\}$. *It is easy to find that* $\{A\}, \{B,D\}, \{\}$ *are the three complete extensions.*

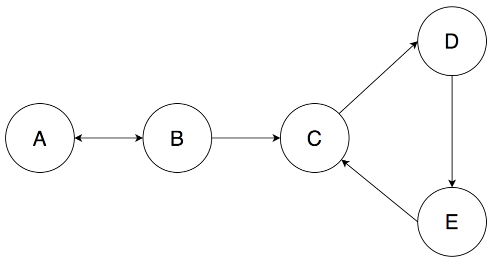


Figure 2.4: AAF with 5 Arguments

A skeptical reasoner only accepts what is undeniably true. This is reflected by the notion of grounded extension (see Definition 6).

Definition 6 (Grounded Extension) *A set of arguments* $S \subseteq \mathcal{A}$ *is called a grounded extension iff it is the minimal complete extension (with respect to set inclusion), i.e. there is no admissible set* $S' \subseteq A$, *such that* $S \subset S'$. *The grounded extension is unique and is included in any other complete extension.*

Example 5 (Cont'd Example 4) *Among the 3 complete extensions, there is the only one grounded extension which is* $\emptyset$.

A credulous reasoner will accept things that are possibly true. This is reflected by the notion of preferred extension (see Definition 7).

Definition 7 (Preferred Extension) *A set of arguments* $S \subseteq \mathcal{A}$ *is called a preferred extension iff it is the maximal complete extension (with respect to set inclusion), i.e. there is no admissible set* $S' \subseteq A$, *such that* $S \subset S'$.

Example 6 *Given the AAF* $\mathcal{F} = (\mathcal{A}, \mathcal{C})$ *such that* $\mathcal{A} = \{B, F\}$ *and* $\mathcal{C} = \{(B,F), (F,B)\}$, *represented in Figure 2.5, there are two preferred extensions:* $\{\{B\}, \{F\}\}$. *Please note that multiple preferred extensions can exist.*

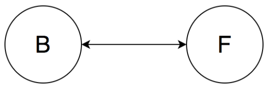


Figure 2.5: AAF with 2 Arguments

Definition 8 (Stable Extension) *A conflict-free set S is a stable extension if S attacks all arguments in* $\mathcal{A} \setminus S$.

Example 7 *As Figure 2.6 shown, $\mathcal{F} = (\mathcal{A},\mathcal{C})$ where $\mathcal{A} = \{A,B,C,D\}$, $\mathcal{C} = \{(A,B),(B,A),(B,C),(A,C),(C,D)\}$. $\{A,D\}$ and $\{B,D\}$ are the stable extensions, and the preferred extensions. Every stable extension is a preferred extension, but not vice versa.*

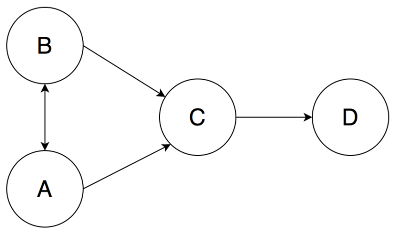


Figure 2.6: AAF with 4 Arguments

Definition 9 (Sceptically/Credulously Accepted) *Given an AAF $\mathcal{F} = (\mathcal{A},\mathcal{C})$, we say that an argument $A \in \mathcal{A}$ is sceptically (resp. credulously) accepted for a semantics iff A belongs to all (resp. at least one) extension for that semantics.*

This report will only focus on the original semantics defined by Dung. Although many more fine-grained semantics have emerged, such as semi-stable and stage semantics, this report will not discuss them but the reader is invited to read more about them (Caminada, 2006, 2011; Baroni et al., 2011).

2.3 Labelling with AAFs

The labelling-based approach was proposed by Verheij (Verheij, 1999). Unlike Dung's original semantics, where extensions are computed for different semantics, functions that affect labels to arguments are computed in this approach. The basic idea is that the status assignment to arguments defined by semantics can be directly defined by assigning labels to the arguments in the framework's corresponding argument graph. Later, in Caminada's approach, the set of labels has been defined as $\{IN,OUT,UNDEC\}$, where IN indicates that the argument is not defeated and is justified. OUT indicates that the argument is defeated and overruled, and $UNDEC$ indicated that the status of the argument is undecided (Modgil and Caminada, 2009). The third label $UNDEC$ is typically used in AAF with self-attacking, and can solve the problem when an idea cannot be adequately proved or adequately rejected. The notion of labelling and legal labelling will be recalled in following definitions.

Definition 10 (Labelling in AAFs) *Given an AAF $\mathcal{F} = (\mathcal{A},\mathcal{C})$, a labelling for $\mathcal{F}$ is a function $\mathcal{L} : \mathcal{A} \rightarrow SYMB$, where SYMB is a set of symbols.*

A legal labelling is a particular labelling that affects the labels $IN,OUT,UNDEC$ to arguments such that some constraints are satisfied.

Definition 11 (Legal labelling) *We say that a labelling $\mathcal{L} : \mathcal{A} \longrightarrow \{IN,OUT,UNDEC\}$ for a given AAF $\mathcal{F} = (\mathcal{A},\mathcal{C})$ is legal if the following constraints are satisfied:*

- *$\forall a \in \mathcal{A}, \mathcal{L}(a) = IN$ iff all of a's defeaters are labelled OUT. We denote the set of all arguments labelled IN as $IN(\mathcal{L}) = \{a \in \mathcal{A}$ s.t. $\mathcal{L}(a) = IN\}$.*

- *$\forall a \in \mathcal{A}, \mathcal{L}(a) = OUT$ iff there is at least one of its defeaters that is labelled IN. We denote the set of all arguments labelled OUT as $OUT(\mathcal{L}) = \{a \in \mathcal{A}$ s.t. $\mathcal{L}(a) = OUT\}$.*
- *$\forall a \in \mathcal{A}, \mathcal{L}(a) = UNDEC$ iff not all of its defeaters are labelled OUT and there is none of its defeaters that is IN. We denote the set of all arguments labelled $UNDEC$ as $UNDEC(\mathcal{L}) = \{a \in \mathcal{A}$ s.t. $\mathcal{L}(a) = UNDEC\}$.*

Given the *IN* labelled argument (Complete Labelling), we could generate grounded and preferred labelling by minimising or maximising the number of arguments labelled *IN*. In terms of stable labelling, it can similarly be defined by requiring no arguments to be labelled *UNDEC*. Benefiting from these features, labelling approach is often used in some dialogue protocols.

According to the Labelling method in AAF, one can then apply proof dialogues to provide an intuitive result to end-users to identify conflicts between arguments and thus draw conclusions in contentious context. The detailed information will be provided in section 2.4.

2.4 Dialogue Protocols for AAFs

This section will introduce multiple proof dialogues including Grounded Discussion Game, Skeptical Preferred Discussion Game, and Credulous Stable Discussion Game.

2.4.1 Grounded Discussion Game

In the study of Caminada, the Grounded Discussion Game (GDG) was introduced with two players (PRO and OPP), four different moves, each of which has an argument as a parameter and some specified rules (Caminada, 2015).

Definition 12 (Players in GDG) *A grounded discussion game is usually assumed to take place between two players.*

- *The proponent (PRO), who wishes to demonstrate that the argument is justified.*
- *The opponent (OPP), who wishes to demonstrate that argument is not justified.*

Definition 13 (Moves in GDG) *Given argument A and B, the moves for a grounded discussion game are as follow:*

- *HTB(A): Argument A has to be the case - A is in the grounded labelling (moved by PRO);*
- *CB(B): Argument B is not out in the grounded labelling (moved by OPP);*
- *CONCEDE(A): Signals that A is in (moved by OPP);*
- *RETRACT(B): Signals that B is out (moved by OPP);*

where HTB, CB, CONCEDE and RETRACT denote to be the movements of the game; A and B represent the parameters of the argument. The rules of the movements are defined as follow in Definition 14.

Definition 14 (Rules in GDG) *Let $\mathcal{F} = (\mathcal{A}, \mathcal{C})$ be an argumentation framework. A grounded discussion game is a sequence of discussion moves constructed by applying the following principles.*

- *HTB(A) is either the first move, or the previous move was CB(B) in which case A must defeat B, and PRO can't CONCEDE or RETRACT.*
- *CB(B) is moved when B defeats the last HTB(A) statement where CONCEDE(A) has not yet been made; B has not been retracted; the last move was not a CB move, and CONCEDE and RETRACT cannot be played.*
- *CONCEDE(A) can be played when HTB(A) was moved earlier, and all defeaters of A have been retracted, and CONCEDE(A) has not been played.*
- *RETRACT(A) can be played when CB(A) was moved in the past, and an defeaters of A has been conceded, and RETRACT(A) has not been played.*

GDG starts with the *proponent* making an HTB statement. The *opponent* then can make one or more CB, CONCEDE and RETRACT statements. Every time when the proponent makes an HTB, the cycle will repeat. Please note that the opponent can make multiple moves for each move of the proponent.

Definition 15 (Termination of GDG) *A terminated grounded discussion with $HTB(A)$ for some $A \in \mathcal{A}$ is won by the proponent iff the discussion contains $CONCEDE(A)$, otherwise it is won by the opponent.*

Example 8 *Considering the argumentation framework shown in 2.7, $\mathcal{F} = (\mathcal{A}, \mathcal{C})$ where $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ and $\mathcal{C} = \{(A,B), (A,F), (B,C), (B,F), (B,E), (C,D), (D,E), (G,D), (G,H), (H,G)\}$. The GDG could go as following.*

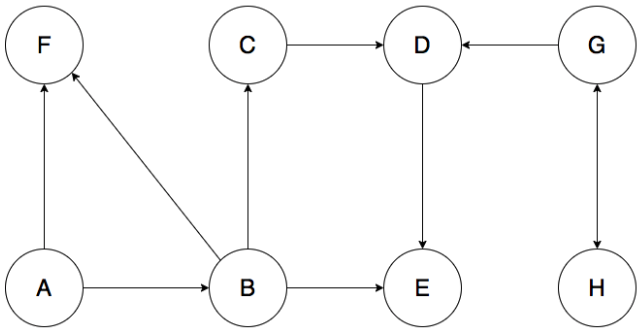


Figure 2.7: AAF from Caminada (2015)

1: PRO: HTB(C)	4: OPP: CONCEDE(A)
2: OPP: CB(B)	5: OPP: RETRACT(B)
3: PRO: HTB(A)	6: OPP: CONCEDE(C)

If the opponent concedes the original argument, the proponent wins this game. The game above which the proponent wins contains HTB and CONCEDE for the same argument C. Otherwise, the opponent wins the game finally in the following game.

1: PRO: HTB(B) 2: OPP: CB(A)

If a repetition of HTB, CB or HTB-CB occurs for the same argument, the opponent wins. This is because the proponent has burden of proof. It has to establish the acceptance of the main argument. The opponent merely has to cast sufficient doubts. Also, the proponent has to make sure that the discussion does not go around in circles.

1: PRO: HTB(F)	4: OPP: CONCEDE(A)
2: OPP: CB(B)	5: OPP: RETRACT(B)
3: PRO: HTB(A)	6: OPP: CB(A)

2.4.2 Skeptical Preferred Discussion Game

The study of Oren (2016) propose a dialogue game, namely, Preferred Discussion Game, for skeptical preferred acceptance, which differs from Vreeswijk's study (Vreeswijk and Prakken, 2000). In the former study, it is not restricted to cases when the preferred and stable semantics coincide (Shams and Oren, 2016). Besides, it is also distinct from the approaches of Modgil and Caminada (2009)'s, where they did not use a meta-dialogue based approach (Modgil and Caminada, 2009). Skeptical preferred semantics seem to capture human intuitions better than credulous preferred semantics, which has been modelled into dialogue games in the past.

This skeptical preferred dialogue game (SPDG) based on a labelling approach different from existing approaches was introduced with two players (the proponent *PRO* and the opponent *OPP*), two moves and two phases.

Definition 16 (Moves in SPDG) *The moves for a skeptical preferred dialogue game are as following.*

- *What is (WI) - requests a label to be assigned to an argument.*
- *Claim (CL) - assign a label to an argument.*

In the definition above, *What is (WI)* and *Claim (CL)* denote to the movement in SPDG, and the rules of movements are described in Definition 17.

Definition 17 (Phases in SPDG) *The phases for a skeptical preferred dialogue game are as following. Players take turns to make a single move, with PRO beginning both phases.*

- *Phase 1 (OPP advances an extension where the argument under discussion is OUT or UNDEC)*
  - *PRO plays WI moves.*
  - *OPP responds with a CL move assigning a legal label to the argument.*
  - *PRO's WI moves are for arguments which defeat a previous CL move.*
  - *Play continues until no moves are possible, an illegal CL is made, or the focal argument is claimed IN. Then Phase 2 begins, otherwise PRO wins.*
- *Phase 2 (PRO shows that this extension is not a preferred extension)*
  - *PRO begins by playing CL on a UNDEC labelled argument.*

- *OPP plays WI on a UNDEC defeater of the CL.*
- *This repeats until no more moves can be made. PRO wins the game if it has made at least one move during this phase, and the labelling is legal.*

Definition 18 (Termination of SPDG) *SPDG terminations will occur in different phases as long as they meet the following conditions.*

Phase 1 terminates when no more moves are possible. OPP will loses the game if:

(1) it utters $CL(IN(X))$, where X is the focal argument;

(2) it labels an argument IN, having previously labelled one of its defeater IN or $UNDEC$;

(3) it labels an argument $UNDEC$, having previously labelled one of its defeater IN;

(4) when no more moves are possible, there is an argument labelled $UNDEC$ for which no defeaters are labelled $UNDEC$;

(5) when no more moves are possible, there is an argument labelled OUT for which no defeaters are labelled IN.

If OPP does not lose the game when phase 1 terminates, phase 2 will begin, and then we have to wait until Phase 2 terminates to determine whether OPP or PRO win the game. When no further moves are possible, the dialogue will terminate. PRO wins the game iff it has made at least one move during Phase 2, and the Labelling at the end of Phase 2 is legal. Otherwise, OPP wins the game.

Example 9 *Considering the argumentation framework shown in Figure 2.8, $\mathcal{F} = (\mathcal{A},\mathcal{C})$ where $\mathcal{A} = \{A,B,E,F,G\}$, and $\mathcal{C} = \{(A,B),(B,G),(G,A),(E,G),(E,F),(F,E)\}$. The SPDG won by OPP could go as following.*

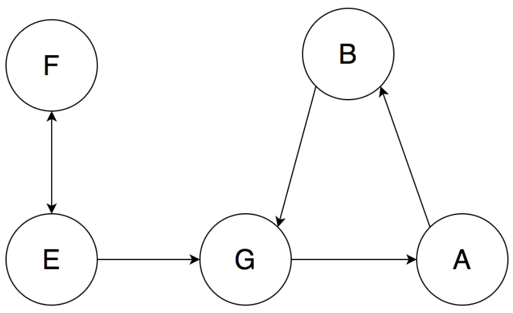


Figure 2.8: AAF for SPDG won by OPP

Phase 1:

PRO : WI(A)
OPP : CL(UNDEC(A))
PRO : WI(G)
OPP : CL(UNDEC(G))
PRO : WI(B)
OPP : CL(UNDEC(B))
PRO : WI(E)
OPP : CL(OUT(E))
PRO : WI(F)
OPP : CL(IN(F))

Phase 2:

PRO : CL(IN(G))
OPP : WI(B)
PRO : CL(OUT(B))
OPP : WI(A)
PRO : CL(IN(A))
OPP : WI(G)
PRO : CL(OUT(G))

PRO have contradicted itself in Phase 2, and OPP therefore wins this game. A is not skeptically preferred.

Example 10 *Considering the argumentation framework shown in Figure 2.9, $\mathcal{F} = (\mathcal{A},\mathcal{C})$ where $\mathcal{A} = \{A,B,C,D\}$, $\mathcal{C} = \{(A,B),(B,A),(B,C),(A,C),(C,D)\}$. The SPDG won by PRO could go as following.*

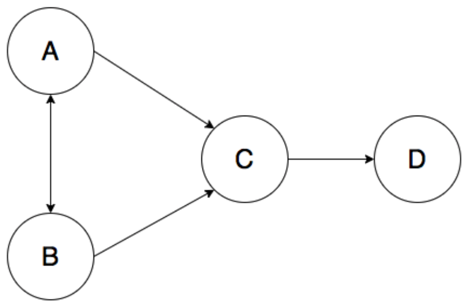


Figure 2.9: AAF for SPDG won by PRO

Phase 1:

PRO : WI(D)
OPP : CL(UNDEC(D))
PRO : WI(C)
OPP : CL(UNDEC(C))
PRO : WI(B)
OPP : CL(UNDEC(B))
PRO : WI(A)
OPP : CL(OUT(A))

Phase 2:

PRO : CL(IN(D))
OPP : WI(C)
PRO : CL(OUT(C))
OPP : WI(B)
PRO : CL(IN(B))
OPP : WI(A)
PRO : CL(OUT(A))

In phase 2, PRO successfully changes an UNDEC argument to IN, and therefore wins; D is skeptically preferred.

In phase 1, OPP identifies an admissible labelling where the focal argument is not IN. If it is a preferred extension, then OPP should win the game, otherwise, they have cheated. Phase 2 allows

PRO to prove that OPP has cheated in phase 1. As a result, there is a winning strategy for PRO or OPP depending on whether the argument is or isn't skeptically preferred.

2.4.3 Credulous Stable Discussion Game

In the work of Caminada, which is inspired by preferred discussion game from Vreeswijk's work, the approach of argument labelling has been adjusted for stable semantics (Vreeswijk and Prakken, 2000; Caminada and Wu, 2009). Note that one thing is unusual, in which stable semantics does not satisfy the property of relevance.

Similar to the preferred discussion game, this credulous stable discussion game (CSDG) was introduced with the same setting (players and moves), except for an additional QUESTION move. By questioning an argument, OPP asks PRO to give an explicit opinion on whether it should be labelled IN or OUT.

Definition 19 (Moves in CSDG) *The moves for a credulous stable discussion game are as following.*

- *IN - label an argument IN.*
- *OUT - label an argument OUT.*
- *QUESTION - involve those arguments sets never been uttered before.*

Definition 20 (Rules in CSDG) *Credulous stable discussion game is described as follows. Players take turns to make a single move, with PRO beginning.*

- *Each move of OPP is either of the form OUT(A), where A is a defeater of some move of PRO, or of the form QUESTION(A), where A is an argument that has never been uttered before. Note that OPP is the only one allowed to do a QUESTION move.*
- *The first move of PRO is of the form IN(A), where A is the main argument of the discussion. If the directly preceding move of OPP is oof the form OUT(B) then A is a defeater of B. If the directly preceding move of OPP is of the form QUESTION(B) then A is either equal to B or a defeater of B.*
- *OPP may not repeat any of its OUT moves.*
- *PRO could repeat its own moves, but could not do an IN(A) move if A has been labelled OUT by OPP before.*

Definition 21 (Termination in CSDG) *A credulous stable discussion game is terminated when it meet the condition below.*

- *OPP wins if it is able to do an OUT(A) move, where argument A has been labelled IN before.*
- *OPP wins if PRO have no possibility to move.*
- *Otherwise, PRO wins.*

Example 11 *Considering the argumentation framework shown in Figure 2.10, $\mathcal{F} = (\mathcal{A},\mathcal{C})$ where $\mathcal{A} = \{A,B,C,D\}$, $\mathcal{C} = \{(A,B),(B,A),(B,C),(C,D),(D,C),(D,D)\}$. The CSDG won by PRO could go as following.*

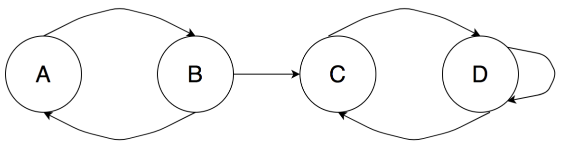


Figure 2.10: AAF for CSDG from Caminada and Wu (2009)

1: PRO: IN(A)	*5: PRO: IN(C)*
2: OPP: OUT(B)	*6: OPP: OUT(D)*
3: PRO: IN(A)	*7: PRO: IN(C)*
4. OPP: QUESTION(C)	

The PRO wins the discussion game since the OPP cannot move anymore. The above example shows that the outcome of one discussion may depend on PRO's response to a QUESTION move, i.e. if PRO replied with IN(D), then it would lose the discussion since OPP would then do OUT(D).

Chapter 3

Methodology

Based on the framework of AAF, an extended approach which is called Argumentation Framework with Set of Attacking Arguments(SETAF), will be employed as the primary method in this thesis. SETAF could allow joint attacking in controversial information, thus would be more flexible than AAF in complicated dialogue. This chapter will introduce the conceptual methodology of SETAF in sections 3.1-3.3, including the definition of SETAFs, Labelling with SETAFs and Dialogue Protocols for SETAFs.

3.1 Argumentation Frameworks with Sets of Attacking Arguments

Recently, a generalisation of Dung's AAF that allows joint attacks has gained a lot of popularity, as reflected by the recent work (Yun et al., 2020a,b; Flouris and Bikakis, 2019; Dvorák et al., 2018). This framework is called SETAF as they argued that it generalize the binary attacks in Dung's AAFs to collective attacks (Nielsen and Parsons, 2006). It also provide a much more elegant and flexible model expressing the complex attack relationships between arguments.

Definition 22 (Argumentation Framework with Sets of Attacking Arguments) *An argumentation framework with sets of attacking arguments (SETAF) is a pair $\mathcal{T} = (\mathcal{E}, \mathcal{R})$, where $\mathcal{E}$ is a set of arguments and $\mathcal{R} \subseteq \left(2^{\mathcal{E}} \backslash \emptyset\right) \times \mathcal{E}$ is a set of attacks.*

Example 12 *Assume a situation shown in Figure 3.1 explaining why Andy who is under 18 cannot buy alcohol if he is not accompanied by an adult. We can represent this situation with the SETAF $\mathcal{T} = (\mathcal{E}, \mathcal{R})$, where $\mathcal{E} = \{A, B_1, B_2\}$ and a attack set $\mathcal{R} = \{(\{B_1, B_2\}, A)\}$, where:*

A: "Andy can buy alcohol."

B1: "Andy is under 18."

B2: "Andy isn't accompanied by an adult."

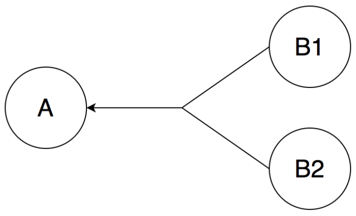


Figure 3.1: SETAF with 3 Arguments

Please note that if either $B1$ or $B2$ is proven to be false, then the argument A (Andy can buy alcohol) will be justified.

Many argumentation semantics have been transplanted and adjusted to be compatible with the framework of SETAFs, such as Dung's original acceptability semantics (conflict-free, admissible, complete, grounded, preferred and stable semantics) and other new semantics (naive, semi-stable, eager, ideal and stage semantics). However, as the new semantics are not what this paper focuses on, the discussion of the new acceptability semantics is excluded from this thesis.

Definition 23 (Argumentation Semantics for SETAFs) *Given a SETAF $\mathcal{T} = (\mathcal{E}, \mathcal{R})$, $A \in \mathcal{E}$, and $S \subseteq \mathcal{E}$. We say that:*

- *S is conflict-free extension iff there is no non-empty set of argument $P \subseteq S$ and $B \in S$ such that $(P,B) \in \mathcal{R}$.*
- *A is acceptable w.r.t. S iff for every set $X \subseteq \mathcal{E}$ that attacks A, there exists $Y \subseteq S$ and $B \in X$ and $(Y,B) \in \mathcal{R}$.*
- *S is admissible extension iff S is conflict-free and each argument in S is acceptable w.r.t S.*
- *S is complete extension if all arguments that are acceptable w.r.t S are in S.*
- *S is grounded extension iff it is the minimal complete extension.*
- *S is preferred extension iff it is the maximal complete extension.*
- *S is stable extension if a conflict-free extension S attacks all arguments in $\mathcal{E} \setminus S$.*

To illustrate the SETAF framework more intuitively, this paper will follow Flouris and Bikakis (2019)'s research to show how those semantics are applied in Example 13 (Flouris and Bikakis, 2019).

Example 13 *Given the SETAF $\mathcal{T} = (\mathcal{E}, \mathcal{R})$, such that $\mathcal{E} = \{A, B1, B2, C, D, E\}$, and $\mathcal{R} = \{(\{A\}, B1),\ (\{A\}, B2), (\{B1\}, A), (\{B2\}, A),\ (\{B1, B2\}, C), (\{C\}, D), (\{D\}, E), (\{E\}, C)\}$, represented in Figure 3.2.*

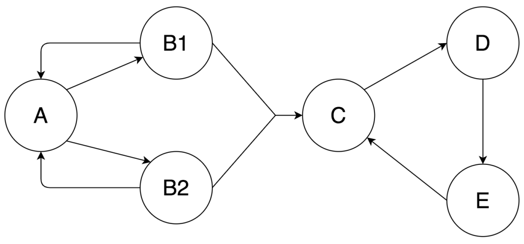


Figure 3.2: SETAF from Flouris and Bikakis (2019)

The extensions for the different argumentation semantics (defined from Definition 23) for the SETAF (represented in Figure 3.2) are show in Table 3.1.

Extension type	Extensions
Conflict-free	$\emptyset, \{A\}, \{B1\}, \{B2\}, \{C\}, \{D\}, \{E\}, \{A,C\}, \{A,D\}, \{A,E\}, \{B1,B2\}, \{B1,B2,D\}, \{B1,B2,E\}, \{B1,C\}, \{B1,D\}, \{B1,E\}, \{B2,C\}, \{B2,D\}, \{B2,E\}$
Admissible	$\emptyset, \{A\}, \{B1\}, \{B2\}, \{B1,B2\}, \{B1,B2,D\}$
Complete	$\emptyset, \{A\}, \{B1,B2,D\}$
Grounded	$\emptyset$
Preferred	$\{A\}, \{B1,B2,D\}$
Stable	$\{B1,B2,D\}$

Table 3.1: Extensions for the different semantics for SETAF represented in Figure 3.2

3.2 Labelling with SETAFs

This thesis will apply the approach of Labelling rather than Dung's original semantics to decide an argument's acceptability, given Labelling could be more expressive in the framework of SETAF. Based on current purpose of extending existing proof dialogue, this section will provide three labelling-based discussion games for SETAFs following what previous study did in Chapter 2. In order to apply the labelling notion defined for AAF to the SETAF context, as described in Definition 24, this study will introduce a new definition for labelling with SETAFs.

Definition 24 (Labelling with SETAFs) *Let $\mathcal{L}_s$ be a labelling for a given SETAF $\mathcal{T} = (\mathcal{E}, \mathcal{R})$ which can be described as a function $\mathcal{L}_s : \mathcal{E} \longrightarrow \{IN, OUT, UNDEC\}$, such that:*

- *$\forall e \in \mathcal{E}, \mathcal{L}_s(e) = IN$ iff all of e's defeaters are labelled OUT. We denote the set of all arguments labelled IN as $IN(\mathcal{L}_s) = \{e \in \mathcal{E}$ s.t. $\mathcal{L}_s(e) = IN\}$.*
- *$\forall e \in \mathcal{E}, \mathcal{L}_s(e) = OUT$ iff there is at least one of its defeaters that is labelled IN. We denote the set of all arguments labelled OUT as $OUT(\mathcal{L}_s) = \{e \in \mathcal{E}$ s.t. $\mathcal{L}_s(e) = OUT\}$.*
- *$\forall e \in \mathcal{E}, \mathcal{L}_s(e) = UNDEC$ iff not all of its defeaters are labelled OUT and there is none of its defeaters that is IN. We denote the set of all arguments labelled $UNDEC$ as $UNDEC(\mathcal{L}_s) = \{e \in \mathcal{E}$ s.t. $\mathcal{L}_s(e) = UNDEC\}$.*

Following the work of Caminada and Gabbay (2009), the approach in this study will have the same properties with AAF labelling (Caminada and Gabbay, 2009). Given complete labelling, we could generate grounded and preferred labelling by minimising or maximising the number of arguments set labelled *IN*. In terms of stable labelling, it can similarly be defined by requiring no arguments set to be labelled *UNDEC*.

3.3 Dialogue Protocols for SETAFs

Following the dialogue protocols for AAF discussed in section 2.4, this section will describe the approach of proof dialogue games under the scenario of SETAF in three different types of games: Grounded Discussion Game, Skeptical Preferred Discussion Game, and Stable Discussion Game.

3.3.1 Grounded Discussion Game

The Grounded Discussion Game with sets of attacking arguments (SETGDG) defined in current section is inspired by Caminada (Caminada, 2015). With the similar setting of GDG, including Players and Moves, this paper replaces the parameter with a set of arguments instead of a single argument.

Definition 25 (Moves in SETGDG) *The moves for one SETGDG are as following.*

- *HTB(A): A set of arguments A has to be the case - Set A is in the grounded labelling. Moved by PRO.*
- *CB(B): A set of arguments B is not out in the grounded labelling. Moved by OPP.*
- *CONCEDE(A): Signals that set A is in. Moved by OPP.*
- *RETRACT(B): Signals that set B is out. Moved by OPP.*

Definition 26 (Rules in SETGDG) *Let $\mathcal{T} = (\mathcal{E}, \mathcal{R})$ be a SETAF. A grounded discussion game is a sequence of discussion moves constructed by applying the following principles.*

- *HTB(A) is either the first move, or the previous move was CB(B) in which case A must defeat at least one arguments set from B, and PRO can't CONCEDE or RETRACT.*
- *CB(B) is moved when arguments set B defeats the last HTB(A) statement where CONCEDE(A) has not yet been made; B has not been retracted; the last move was not a CB move, and CONCEDE and RETRACT cannot be played.*
- *CONCEDE(A) can be played when HTB(A) was moved earlier, and one of defeaters set of A have been retracted, and CONCEDE(A) has not been played.*
- *RETRACT(A) can be played when CB(A) was moved in the past, and a defeater set of A has been conceded, and RETRACT(A) has not been played.*

$SETGDG$ starts with the proponent making a HTB statement. The opponent then can make one or more CB, $CONCEDE$ and $RETRACT$ statements. Every proponent makes a HTB and then the cycle repeats. Please note that the opponent can make multiple moves for every proponent move.

Definition 27 (Termination of SETGDG) *A terminated $SETGDG$ with $HTB(E)$ for some arguments set $E \in \mathcal{E}$ is won by the proponent iff the discussion contains $CONCEDE(E)$, otherwise it is won by the opponent.*

Example 14 *Considering the SETAF shown in Figure 3.3,SETAF $\mathcal{T} = (\mathcal{E}, \mathcal{R})$, such that $\mathcal{E} = \{A, B1, B2, C, D, E, F\}$, and $\mathcal{R} = \{(\{A\}, B1), (\{A\}, B2), (\{A\}, F), (\{B1, B2\}, F), (\{B1, B2\}, C), (\{C\}, D), (\{C\}, E), (\{E\}, D)\}$. SETGDG could go as following.*

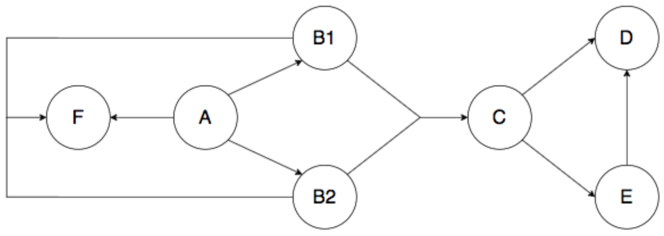


Figure 3.3: SETAF with 7 arguments

1: PRO: $HTB(\{C\})$	4: OPP: $CONCEDE(\{A\})$
2: OPP: $CB(\{B_1,B_2\})$	5: OPP: $RETRACT(\{B_1,B_2\})$
3: PRO: $HTB(\{A\})$	6: OPP: $CONCEDE(\{C\})$

If OPP concedes the original arguments set, PRO wins this game. The game above which PRO wins contains HTB and CONCEDE for the same arguments set $\{C\}$. Otherwise, OPP wins the game finally in the following game.

1: PRO: $HTB(\{B_1,B_2\})$ 2: OPP: $CB(\{A\})$

Considering the situation in Figure 3.3, argument A attacks single argument $B1$ and $B2$ both, then arguments set $\{A\}$ invalid $\{B_1,B_2\}$. Please note that if one arguments set need to invalid some other set, the only thing needed is to make this single focal argument attack at least one argument which belong to the target set.

If a HTB, CB or HTB-CB repeat occurs for the same arguments set, OPP wins. This is because PRO has burden of proof. It has to establish the acceptance of the main arguments set. OPP merely has to cast sufficient doubts. Also, PRO has to make sure that the discussion does not go around in circles.

1: PRO: $HTB(\{F\})$	4: OPP: $CONCEDE(\{A\})$
2: OPP: $CB(\{B_1,B_2\})$	5: OPP: $RETRACT(\{B_1,B_2\})$
3: PRO: $HTB(\{A\})$	6: OPP: $CB(\{A\})$

Regarding the property of the game, now that the working of SETGDG have been outlined, this paper will then prove its soundness and completeness formally with respect to ground semantics inspired by Caminada (2015a).

For soundness, if the focal argument set in one discussion won by PRO is in the grounded extension, the game is said to have soundness. In order to prove this, the first step is to introduce the notions of PRO's and OPP's labelling.

Definition 28 (Labelling of Players) *Let $[M_1 \ldots M_n]$ be a grounded discussion in SETAF $\mathcal{T} = (\mathcal{E},\mathcal{R})$ without any HTB-CB repeats.*
The proponent labelling $\mathcal{E}_P$ is defined as

$$IN(\mathcal{E}_P) = \{A \mid \exists i \in \{1\ldots n\} : M_i = HTB(A)\}$$
$$OUT(\mathcal{E}_P) = \{A \mid \exists i \in \{1\ldots n\} : M_i = CB(A)\}$$
$$UNDEC(\mathcal{E}_P) = \mathcal{E} \setminus (IN(\mathcal{E}_P) \cup OUT(\mathcal{E}_P))$$

The opponent labelling $\mathcal{E}_O$ is defined as

$$IN(\mathcal{E}_O) = \{A \mid \exists i \in \{1 \dots n\} : M_i = CONCEDE(A)\}$$
$$OUT(\mathcal{E}_O) = \{A \mid \exists i \in \{1 \dots n\} : M_i = RETRACT(A)\}$$
$$UNDEC(\mathcal{E}_O) = \mathcal{E} \setminus (IN(\mathcal{E}_O) \cup OUT(\mathcal{E}_O))$$

Notice that $\mathcal{E}_O$ in Definition 28 does not depend on the absence of HTB-CB repeats whereas the well-definiteness of $\mathcal{E}_P$ does. When applying $\mathcal{E}_O$, we will often do so without having ruled out any HTB-CB repeats.

Theorem 1 *Let $\mathcal{E}_O$ be the opponent's labelling related to discussion $[M_1 \dots M_n]$. It holds that $\mathcal{E}_O$ is strongly admissible.*

Theorem 1 shows that at any stage of the discussion, $\mathcal{E}_O$ is strongly admissible (proved by Caminada). Hence, when the game is terminated and won by PRO, we have a strongly admissible labelling where the focal arguments set is labelled IN. It follows that the focal arguments set is labelled IN by the grounded labelling and is therefore an element of the grounded extension, leading to the following theorem.

Theorem 2 *Let $[M_1 \dots M_n]$ be a terminated grounded discussion, won by PRO, with focal arguments set A, It holds that A is in the grounded extension.*

Although it is possible to infer that an arguments set is in the grounded extension when PRO wins a game, it is difficult to infer that an arguments set is not in the grounded extension when PRO loses a discussion. This is because loss of a game could be due to PRO following a flawed strategy. For instance, in the SETAF (Figure 3.4), such that $\mathcal{E} = \{A, B1, B2, C, D, E, F\}$, and $\mathcal{R} = \{(\{A\}, B1), (\{A\}, B2), (\{B1, B2\}, C), (\{C\}, D), (\{D\}, E), (\{D\}, F), (\{E\}, C), (\{F\}, D)\}$. we could have the following discussion:

1: PRO: $HTB(\{C\})$
2: OPP: $CB(\{E\})$
3: PRO: $HTB(\{D\})$
4: OPP: $CB(\{F\})$
5: PRO: $HTB(\{D\})$

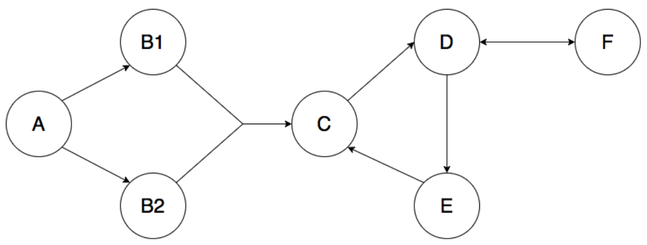


Figure 3.4: SETAF with 7 arguments

This discussion is terminated at step 5 because there exist a HTB-CB repeat for arguments set $\{D\}$. The focal arguments set is not conceded, so OPP win the discussion.

After showing the soundness of the game, the next procedure is to pay attention to completeness. Caminada mentioned that an obvious thing to prove completeness would be "if A is in the grounded extension, then there exists a discussion won by PRO with A as the main argument" (Caminada, 2015). The lowest number strategy had been presented as well to help PRO win the discussion. Please note that if PRO uses a lowest number strategy, then no HTB-CB repeats occur.

Theorem 3 *Let E be an arguments set in SETAF $\mathcal{T} = (\mathcal{E}, \mathcal{R})$. If PRO uses a lowest number strategy, he will win the discussion for focal arguments set E.*

As the presence of a winning strategy trivially implies the presence of at least one discussion that is won by PRO, it could be immediately obtained that there exists at least one terminated grounded discussion, won by PRO, for focal arguments set E.

3.3.2 Skeptical Preferred Discussion Game

This section will defined the Skeptical Preferred Discussion Game (SETSPDG) inspired by the work of Sham and Oren (2016), which consist of two players (PRO and OPP), two moves and two phases (Shams and Oren, 2016).

Definition 29 (Moves in SETSPDG) *The moves for a skeptical preferred dialogue game are as following.*

- *What is (WI) - requests a label to be assigned to an arguments set.*
- *Claim (CL) - assign a label to an arguments set.*

Definition 30 (Phases in SETSPDG) *The phases for a skeptical preferred dialogue game are as following. Players take turns to make a single move, with PRO beginning both phases.*

- *Phase 1 (OPP advances an extension where the arguments set under discussion is OUT or UNDEC)*
  - *PRO plays WI moves.*
  - *OPP responds with a CL move assigning a legal label to the arguments set.*
  - *PRO's WI moves are for arguments sets which defeat a previous CL move.*
  - *Play continues until no moves are possible. Then Phase 2 begins.*
- *Phase 2 (PRO shows that this extension is not a preferred extension)*
  - *PRO begins by playing CL on a UNDEC labelled arguments set.*
  - *OPP plays WI on a UNDEC defeater of the CL.*
  - *This repeats until no more moves can be made. PRO wins the game if it has made at least one move during this phase, and the labelling is legal.*

Definition 31 (Termination of SETSPDG) *SETSPDGs termination will occur in different phases as long as they meet the following conditions.*

Phase 1 terminates when no more moves are possible. OPP will loses the game if it

(1) it utters CL(IN(X)), where X is the focal arguments set.

(2) it labels an arguments set IN, having previously labelled one of its defeater IN or UNDEC.

(3) it labels an arguments set UNDEC, having previously labelled one of its defeater IN.

(4) when no more moves are possible, there is an arguments set labelled UNDEC for which no defeaters are labelled UNDEC.

(5) when no more moves are possible, there is an arguments set labelled OUT for which no defeaters are labelled IN.

If OPP does not lose the game when phase 1 terminates, phase 2 will begin. Then we have to wait until phase 2 terminates to determine whether OPP or PRO win the game. When no further moves are possible, the dialogue will terminate. PRO wins the game iff it has made at least one move during phase 2 and the labelling at the end of phase 2 is legal; otherwise, OPP wins the game.

Example 15 *Considering the SETAF shown in Figure 3.5, SETAF* $\mathcal{T} = (\mathcal{E},\mathcal{R})$*, such that* $\mathcal{E} = \{A, B1, B2, C, D, E\}$*, and* $\mathcal{R} = \{(\{A\}, B1), (\{A\}, B2), (\{B1, B2\}, C), (\{C\}, D), (\{D\}, E), (\{E\}, C)\}$*. SETSPDG won by OPP could go as following.*

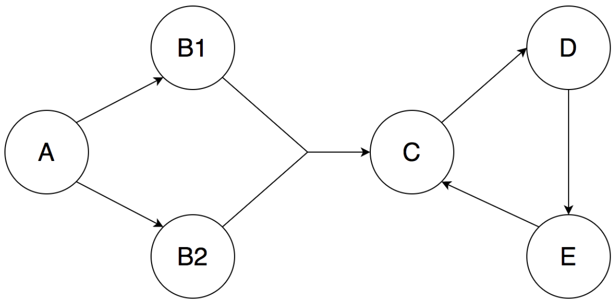


Figure 3.5: SETSPDG won by OPP

Phase 1:

$PRO : WI(\{D\})$
$OPP : CL(UNDEC(\{D\}))$
$PRO : WI(\{C\})$
$OPP : CL(UNDEC(\{C\}))$
$PRO : WI(\{E\})$
$OPP : CL(UNDEC(\{E\}))$
$PRO : WI(\{B1, B2\})$
$OPP : CL(OUT(\{B1, B2\}))$
$PRO : WI(\{A\})$
$OPP : CL(IN(\{A\}))$

Phase 2:

$PRO : CL(IN(\{C\}))$
$OPP : WI(\{E\})$
$PRO : CL(OUT(\{E\}))$
$OPP : WI(\{D\})$
$PRO : CL(IN(\{D\}))$
$OPP : WI(\{C\})$
$PRO : CL(OUT(\{C\}))$

PRO have contradicted itself in Phase 2, and OPP therefore wins this game. $\{D\}$ is not skeptically preferred.

Example 16 *Considering the SETAF shown in Figure 3.6, SETAF* $\mathcal{T} = (\mathcal{E},\mathcal{R})$*, such that* $\mathcal{E} = \{A, B1, B2, C, D\}$*, and* $\mathcal{R} = \{(\{A\}, B1), (\{A\}, D), (\{B1, B2\}, C), (\{D\}, A), (\{D\}, B2)$*. SETSPDG won by PRO could go as following.*

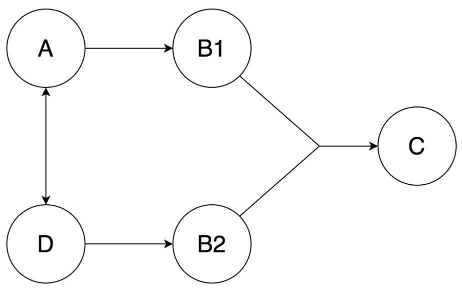


Figure 3.6: SETSPDG won by PRO

Phase 1:

PRO : $WI(\{C\})$
OPP : $CL(UNDEC(\{C\}))$
PRO : $WI(\{B1,B2\})$
OPP : $CL(UNDEC(\{B1,B2\}))$
PRO : $WI(\{D\})$
OPP : $CL(UNDEC(\{D\}))$
PRO : $WI(\{A\})$
OPP : $CL(OUT(\{A\}))$

Phase 2:

PRO : $CL(IN(\{C\}))$
OPP : $WI(\{B1,B2\})$
PRO : $CL(OUT(\{B1,B2\}))$
OPP : $WI(\{D\})$
PRO : $CL(IN(\{D\}))$
OPP : $WI(\{A\})$
PRO : $CL(OUT(\{A\}))$

In Phase 2, PRO successfully changes an UNDEC arguments set to IN, and therefore wins; {D} is skeptically preferred.

In phase 1, OPP identifies an admissible labelling where the focal arguments set is not in. If it is a preferred extension, then OPP should win the game, otherwise, they have cheated. Phase 2 allows PRO to prove that OPP has cheated in phase 1. As a result, there is a winning strategy for PRO or OPP depending on whether the arguments set is or isn't skeptically preferred.

3.3.3 Credulous Stable Discussion Games

In this section, a credulous stable discussion game with sets of attacking arguments (SETCSDG) will be introduced, whose key idea come from the work of Caminada and Wu (Caminada and Wu, 2009).

Definition 32 (Moves in SETCSDG) *The moves in SETCSDGs can be described as following.*

- *IN - label an arguments set IN.*
- *OUT - label an arguments set OUT.*
- *QUESTION - involve those arguments sets never been uttered before.*

Definition 33 (Rules in SETCSDG) *SETCSDGs have rules as following. Players take turns to make a single move, with PRO beginning.*

- *Each move of OPP is either of the form OUT(A), where A is a defeater of some move of PRO, or of the form QUESTION(A), where A is an arguments set that has never been uttered before. Note that OPP is the only one allowed to do a QUESTION move.*

- *The first move of PRO is of the form IN(A), where A is the focal arguments set of the discussion. If the directly preceding move of OPP is of the form OUT(B) then A is a defeater of B. If the directly preceding move of OPP is of the form QUESTION(B) then A is either equal to B or a defeater of B.*
- *OPP may not repeat any of its OUT moves.*
- *PRO could repeat its own moves, but could not do an IN(A) move if A has been labelled OUT by OPP before.*

Definition 34 (Termination in SETCSDG) *SETCSDGs terminate when it meet the condition below.*

- *OPP wins if it is able to do an OUT(A) move, where arguments set A has been labelled IN before.*
- *OPP wins if PRO have no possibility to move.*
- *Otherwise, PRO wins.*

Example 17 *Considering the SETAF shown in Figure 3.7, SETAF $\mathcal{T} = (\mathcal{E},\mathcal{R})$, such that $\mathcal{E} = \{A, B1, B2, C, D\}$, and $\mathcal{R} = \{(\{A\}, B2), (\{A\}, D), (\{B1, B2\}, C), (\{C\}, C), (\{C\}, B2), (\{D\}, A)$. SETCSDG won by PRO can go as following.*

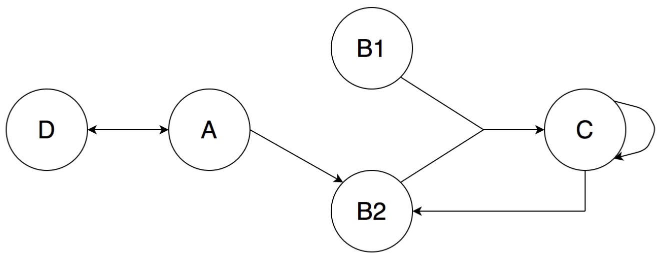


Figure 3.7: A SETAF for CSDG

1: PRO: $IN(\{D\})$	*7: PRO: $IN(\{B1\})$*
2: OPP: $OUT(\{A\})$	*8: OPP: $QUESTION(\{B1, B2\})$*
3: PRO: $IN(\{D\})$	*9: PRO: $IN(\{B1, B2\})$*
4. OPP: $QUESTION(\{B2\})$	*10: OPP: $OUT(\{C\})$*
5: PRO: $IN(\{B2\})$	*11: PRO: $IN(\{B2\})$*
6: OPP: $QUESTION(\{B1\})$	

In Example 17, the PRO finally wins the discussion game, since the OPP cannot move anymore. The above example shows that the outcome of one discussion may depend on PRO's response to a QUESTION move, i.e., if PRO replied with $IN(\{C\})$ in step 5, then it would lost the discussion, since OPP would then do $OUT(\{C\})$.

Theorem 4 *Let A be an arguments set in SETAF $\mathcal{T} = (\mathcal{E},\mathcal{R})$. There exists a stable labelling $\mathcal{L}$ with $\mathcal{L}(A) = IN$ iff there exists a stable admissible discussion for A that is won by PRO.*

In order to prove the correctness of theorem 4, it will be discussed from two angles of its sufficiency and necessity. The sufficiency for theorem 4 will be firstly proved. Suppose that there exists a stable labelling $\mathcal{L}$ with $\mathcal{L}(A) = IN$. It means that we have a game in which all in-labelled moves are also labelled IN in the stable labelling $\mathcal{L}$.

Let $[M_1 \ldots M_n]$ be an unfinished discussion where M_n is a proponent move and all proponent moves are labelled IN in $\mathcal{L}$. Based on the fact that the discussion is unfinished, it follows that OPP can do a move M_{n+1} which is either of the form $OUT(B)$, where B is a defeater of some earlier proponent move ($IN(A)$) or $QUESTION(B)$. After PRO play $IN(A)$ move, OPP could play $OUT(B)$ or $QUESTION(B)$ move. In the former case, it holds that B is labelled OUT in $\mathcal{L}$, since A is labelled IN in $\mathcal{L}$. It then follows that there exists an arguments set C which defeats B and is labelled IN in $\mathcal{L}$, which makes it possible for PRO to respond with $IN(C)$. In the latter case, PRO can either respond with $IN(B)$ if B is labelled IN in $\mathcal{L}$, or with $IN(C)$ if B is labelled OUT in $\mathcal{L}$, where C is a defeater of B. In any case, the discussion will terminate with a proponent move as the last move, and all proponent moves labelled IN in $\mathcal{L}$.

Then, for the proof of necessity, let us suppose that there exists a stable discussion game for arguments set A that is won by PRO. Let AR be the set of the in-labelled arguments set. AR is conflict-free, otherwise the opponent would have labelled an arguments set out that was labelled in by PRO earlier and would have won the discussion. Furthermore, AR defeats each argument that is not in AR. This implies that there has not been a proponent move $IN(B)$. When the discussion is finished, it follows that PRO has played each and every possible move. PRO did not play $IN(B)$ then implies that OPP play $OUT(B)$ or $QUESTION(B)$. In the former case (OPP played $OUT(B)$), it follows that PRO reacted with a move $IN(C)$ where C is a defeater of B. In the latter case (OPP played $QUESTION(B)$), it follows that PRO reacted with a move $IN(C)$ where C is a defeater of B. In either case, PRO reacted with $IN(C)$ where C is a defeater of B. So AR contains an argument C that defeats B.

Chapter 4

Implementation

This chapter will provide the implementation of the proof dialogue games, namely, the discussion games. Specifically, this paper will generate a web application in order to display the structure of SETAF and to play three semantic discussion games followed by Yun et al., (2020c)'s WBSETAF [1] project. However, unlike Yun et al., (2020c)'s project that is developed by Java language, this paper will use Typescript language to implement this application based on front-end library React.js [2] as Typescript has better compatibility than Java. Typescript could support Javascript syntax and provide some class-based object-oriented programming syntax similar to Java language.

4.1 Libraries Chosen

As the web application is a front-end project which has presentation and interaction functions, this paper mainly uses the library as follows:

(1) React.js is a JavaScript library for building user interfaces.

(2) Ant Design.js[3] is an enterprise-class UI design language and React UI library.

(3) Vis.js[4] is a visualization to display networks and networks consisting of nodes and edges.

4.2 Directory Structure

Figure 4.1 suggests the main structure of creating the web application following a logical way that could be accessible and easily located. Details and explanations could be found below.

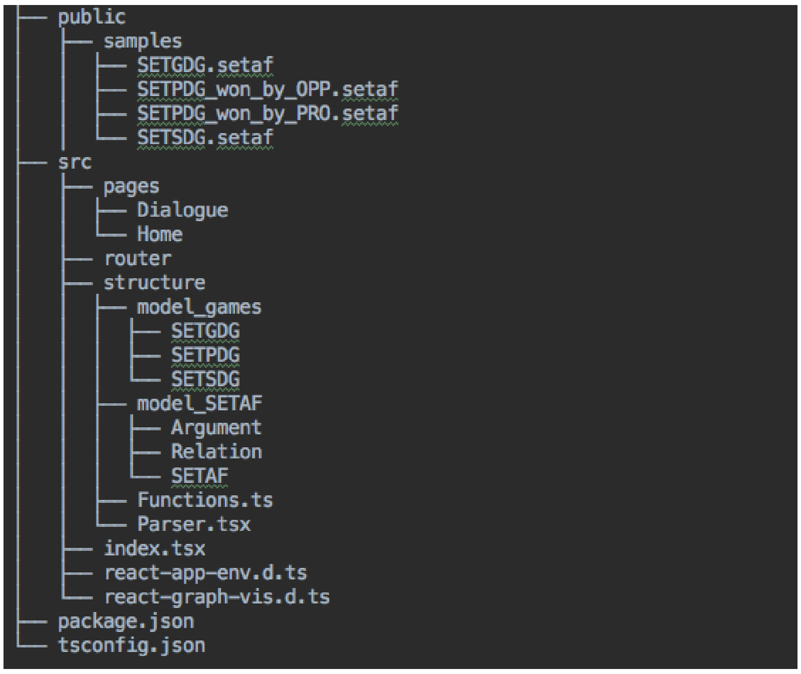
- **public**: is a public folder.
  - **samples**: is folder for saving sample files.
    - **SETGDG.setaf**: is a .setaf sample file for SETGDG.
    - **SETPDG_won_by_OPP.setaf**: is a .setaf sample file for SETPDG won by OPP.
    - **SETPDG_won_by_PRO.setaf**: is a .setaf sample file for SETPDG won by PRO.
    - **SETSDG.setaf**: is a .setaf sample file for SETSDG.
- **src**: is a main running folder.

[1]https://github.com/brunoyun/WBSETAFs
[2]https://github.com/facebook/react/
[3]https://github.com/ant-design/ant-design
[4]https://github.com/visjs/vis-network

- **pages**: is a page folder.
  - **Dialogue**: is the folder for saving components and interacting algorithms on the game-playing page.
  - **Home**: is the folder for saving components and interacting algorithms on the landing page.
- **router**: is the folder for saving the route component, route setting.
- **structure**: is the folder for saving game models and SETAF models.
  - **model_games**: is a folder saving game models.
    - **SETGDG**: is the SETGDG model file.
    - **SETPDG**: is the SETPDG model file.
    - **SETSDG**: is the SETSDG model file.
  - **model_SETAF**: is a folder saving SETAF models.
    - **Argument**: is the Argument model file.
    - **Relation**: is the Relation model file.
    - **SETAF**: is the SETAF model file.
- **index.tsx**: is the main program entry of this web application.
- **react-app-env.d.ts**: is a necessary declaration file for react lib, which is used to compile for Typescript language.
- **react-graph-vis.d.ts**: is a necessary declaration file for vis lib, which is used to compile for Typescript language.

• **package.json**: is a configuration file for libs in node module.

• **tsconfig.json**: is a configuration file for compiling .ts file.



```
├── public
│   ├── samples
│   │   ├── SETGDG.setaf
│   │   ├── SETPDG_won_by_OPP.setaf
│   │   ├── SETPDG_won_by_PRO.setaf
│   │   └── SETSDG.setaf
├── src
│   ├── pages
│   │   ├── Dialogue
│   │   └── Home
│   ├── router
│   ├── structure
│   │   ├── model_games
│   │   │   ├── SETGDG
│   │   │   ├── SETPDG
│   │   │   └── SETSDG
│   │   ├── model_SETAF
│   │   │   ├── Argument
│   │   │   ├── Relation
│   │   │   └── SETAF
│   │   ├── Functions.ts
│   │   └── Parser.tsx
│   ├── index.tsx
│   ├── react-app-env.d.ts
│   └── react-graph-vis.d.ts
├── package.json
└── tsconfig.json
```

Figure 4.1: Project structure

4.3 Web Application

The implementation of this web application is divided into three main stages: SETAF models setup, proof dialogue games setup, and user interface design.

4.3.1 SETAF Models Setup

The Proof Dialogue Games in the web application is based on SETAF models, which are set up using three elements: Argument, Attacking Relation and SETAF. The SETAF models are designed to create a complete SETAF network, including the attacking argument instances, the target argument instances, and the attacking relation instances. The detailed steps and codes for setting up SETAF models could be found in Appendix 1.

4.3.2 Dialogue Protocols Setup

After setting up the SETAF model, the Dialogue Protocols and thus the models for the Proof Dialogue Games could be constructed. Dialogue Protocols Setup includes two parts: dialogue game models setup and interacting algorithms in the front-end. This application includes three different types of SETAF-based dialogue games (i.e., SETGDG, SETPDG, SETSDG) whose related definition and theory have been introduced in section 3.3. This section mainly provides the codes of moves and rules in protocols in these three game models. Meanwhile, the state of players and termination rules have been claimed in the interacting algorithms.

In all three dialogue games, users and the system will play the PRO and OPP roles, respectively. PRO always plays first and presents a focal argument. After each move of PRO, the system will try to find an argument following the protocols of discussion games. Then, according to OPP's move, the system will try to find all the defenders as choices for PRO. Users (PRO) can pick one of these options as their defender and continue playing the game. Games will keep this loop until some moves meet the termination rules or there are no available moves for one of the players.

SETGDG Game

With the method of labelling in SETAF, all arguments need to be allocated to the set of IN or OUT hold by the SETGDG model in an ideal situation (namely, the focal argument is justified). Meanwhile, the model maintains other sets to track the progress of the game. SETGDG model also defines the moves' function and provide some available functions for the interacting algorithms (details and codes could be found in Appendix 2)

SETPDG Game

In SETPDG, there are four collections maintained by model instance: IN, OUT, UNDEC and TODO. The IN set is responsible for receiving arguments after every CL moves where PRO or OPP gives arguments as label IN. The OUT set stores arguments after each CL move, where players set arguments as label OUT. The UNDEC set is an arguments container saving the undecided arguments after the CL moves. Finally, the set of TODO tracks the arguments to be selected.

Based on the collections in the SETPDG model, there are two phases in this game. In the first phase, PRO makes a focal argument with a WI move. Then OPP may try to give this argument one label. If this focal argument has no attacker, it will be labelled as IN. Otherwise, it will be labelled as UNDEC. All the arguments can be given one label IN or UNDEC in this phase, and then the second phase will begin. In the second phase, PRO will come up with one argument from the UNDEC set and label it as IN. Then OPP may try to find one attacker of this argument and give it an OUT label. If at this stage, PRO's actions are all legal, namely, the labels PRO assigned being agree with the already labelled argument, the game is won by PRO; otherwise, OPP wins. (details and codes could be found in Appendix 2).

SETSDG Game

In SETSDG, there are three collections maintained by model instance: IN, OUT, and QUESTION. The IN set is responsible for receiving arguments after PRO performs IN moves. The OUT set stores arguments after OPP performs the OUT moves. The QUESTION set saves the arguments that are neither in IN nor OUT set and inits with all origin arguments in attacking relations. SETSDG starts with PRO proposing the Focal argument. PRO played by users can choose one option from the given arguments set and label it IN. Then OPP acted by the system will query all attacking relations in SETAF to determine whether the focal argument attacks any arguments. If there are any arguments, namely target arguments, attacked by focal argument, OPP will choose one of them and make an OUT move. Then PRO will find one of the target arguments attacked by the argument chosen by OPP and act an IN move. However, if no arguments are attacked by focal argument, OPP will choose one argument from the unlabelled arguments and make a Question move. Then PRO will do the same thing to label the argument attacked by OPP's option as an IN argument.

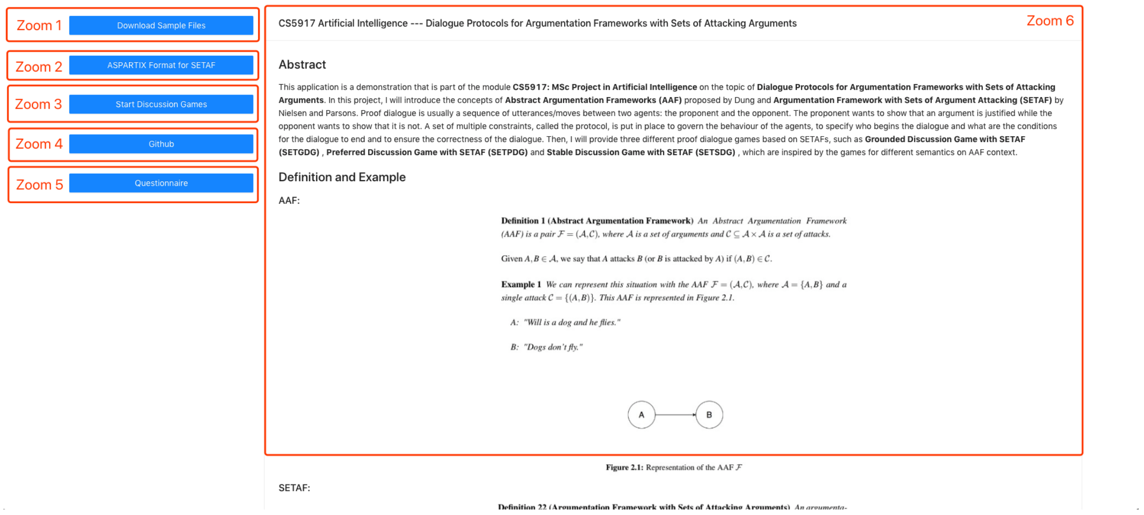
With this loop between PRO and OPP, when OPP gives an OUT move for the argument labelled IN, or PRO has no possibility to move further, the discussion game terminates, and OPP wins the game (details and codes could be found in Appendix 2).

4.3.3 User Interface Design

The user interface in this web application contains two parts: Home Page and Dialogue Game Page.

Home Page

The home page is the landing page for this application. Users could find the explanation and examples of necessary terminology and guide to use this web application. The first column of Figure 4.2 denotes the function bar divided into five zooms 1-5 ranging from sample files to questionnaires. The largest area on the right-hand side represents the window where users could read abstract, theoretical background, application guides, and so force.



CS5917 Artificial Intelligence --- Dialogue Protocols for Argumentation Frameworks with Sets of Attacking Arguments

Abstract

This application is a demonstration that is part of the module **CS5917: MSc Project in Artificial Intelligence** on the topic of **Dialogue Protocols for Argumentation Frameworks with Sets of Attacking Arguments**. In this project, I will introduce the concepts of **Abstract Argumentation Frameworks (AAF)** proposed by Dung and **Argumentation Framework with Sets of Argument Attacking (SETAF)** by Nielsen and Parsons. Proof dialogue is usually a sequence of utterances/moves between two agents: the proponent and the opponent. The proponent wants to show that an argument is justified while the opponent wants to show that it is not. A set of multiple constraints, called the protocol, is put in place to govern the behaviour of the agents, to specify who begins the dialogue and what are the conditions for the dialogue to end and to ensure the correctness of the dialogue. Then, I will provide three different proof dialogue games based on SETAFs, such as **Grounded Discussion Game with SETAF (SETGDG)** , **Preferred Discussion Game with SETAF (SETPDG)** and **Stable Discussion Game with SETAF (SETSDG)** , which are inspired by the games for different semantics on AAF context.

Definition and Example

AAF:

Definition 1 (Abstract Argumentation Framework) *An Abstract Argumentation Framework (AAF) is a pair $\mathcal{F} = (\mathcal{A}, \mathcal{C})$, where $\mathcal{A}$ is a set of arguments and $\mathcal{C} \subseteq \mathcal{A} \times \mathcal{A}$ is a set of attacks.*

Given $A, B \in \mathcal{A}$, we say that A attacks B (or B is attacked by A) if $(A, B) \in \mathcal{C}$.

Example 1 *We can represent this situation with the AAF $\mathcal{F} = (\mathcal{A}, \mathcal{C})$, where $\mathcal{A} = \{A, B\}$ and a single attack $\mathcal{C} = \{(A, B)\}$. This AAF is represented in Figure 2.1.*

A: "Will is a dog and he flies."

B: "Dogs don't fly."

$$\begin{tikzcd} A \arrow[r] & B \end{tikzcd}$$

Figure 2.1: Representation of the AAF $\mathcal{F}$

SETAF:

Definition 22 (Argumentation Framework with Sets of Attacking Arguments) *An argumenta-*

Figure 4.2: Application interface in Home Page

Below is the detailed explanation of zoom 1 to zoom 6 for the user interface on the home page.

- **Zoom 1**: is a link to download the sample zip file, which includes four .setaf files. Users can choose any one of them for any discussion game.
- **Zoom 2**: is the link to the website of ASPARTIX format for SETAF, where there are more details of SETAF format by clicking the link.
- **Zoom 3**: is the link to the dialogue game page.
- **Zoom 4**: is the link to the Github repository, in which users can find the project source code.
- **Zoom 5**: is the link to the feedback form (made by Google Form),
- **Zoom 6**: shows some information about this web application, including an introduction, explanation of terms and guides that show how to use this application.

Firstly, users are suggested to download the sample files by clicking Zoom 1 after reading the information in Zoom 6. Afterwards, they could either obtain more information on SETAF format by clicking Zoom 2 if they would like to or directly go to the dialogue game page by clicking the Zoom 3 button. After playing the game, users are encouraged to provide feedback by linking to the Zoom 5 button. Finally, if they are interested in this project's source code, they could refer to the information provided by the link of Zoom 4.

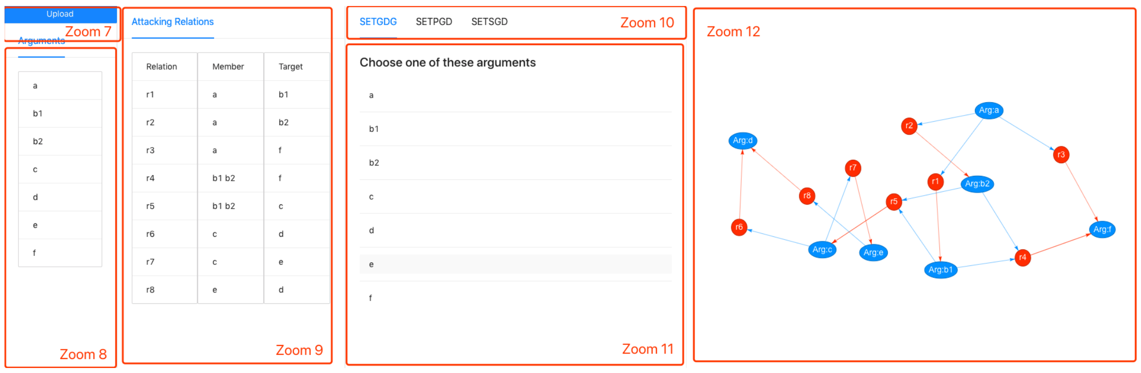
Dialogue Game Page

The procedure to start the game includes three main steps mentioned above: downloading sample files, reading the information in Zoom 6, and accessing the dialogue game hitting Zoom 3. Following these three steps, users could access the Dialogue Game Page, which plays a central role in interaction. If users land on the dialogue games page for the first time, they will be asked to upload one of the sample files downloaded before (shown in Figure 4.4).

The dialogue game page is an integration of most of the functions of this web application. It is designed to display argument sets and attacking relation sets and to play dialogue games. Considering the readability of the arguments set and attacking relation set in SETAF, the whole network of SETAF is visualized and designed in a diagram.

After uploading the files, they can find the content of the files in the SETAF reader shown in Figure 4.5. If users click the *Done* button, this program will automatically parse the *.setaf* file into a SETAF model (introduced in the following section). The general idea is that after the model of SETAF being initialized, the parsed text should be classified into argument set and attacking relation set according to regular expressions.

Intuitively, this page can be divided into six areas named Zoom 7 to Zoom 12, as shown in Figure 4.3.



Relation	Member	Target
r1	a	b1
r2	a	b2
r3	a	f
r4	b1 b2	f
r5	b1 b2	c
r6	c	d
r7	c	e
r8	e	d

Choose one of these arguments

a

b1

b2

c

d

e

f

Figure 4.3: Application interface in Dialogue Game Page

- **Zoom 7** denotes a link that allows users to upload the .setaf files for different discussion games.

- **Zoom 8** shows the arguments after file parsing.

- **Zoom 9** displays the attacking relations, which consist of the set of attacking arguments and the arguments attacked.

- **Zoom 10** is a selector to switch types of games.

- **Zoom 11** shows the progress of a specified dialogue game. Users can pick one of the given arguments to interact with the computer player.

- **Zoom 12** presents the network of the whole SETAF, where attacking relation could be represented by using nodes and edges (arrows) on the canvas. There are two types of nodes in the canvas: the red circle nodes denoting the attacking relations and the blue oval nodes representing the arguments. Edges (arrows) indicates the direction of the attacking relations that could have binary directions. For example, in Zoom 12 in Figure 4.3, arguments b1 and b2 are attacking argument c through attacking relation r5.

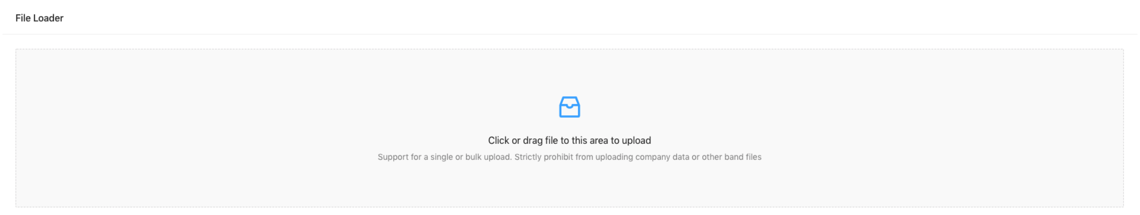
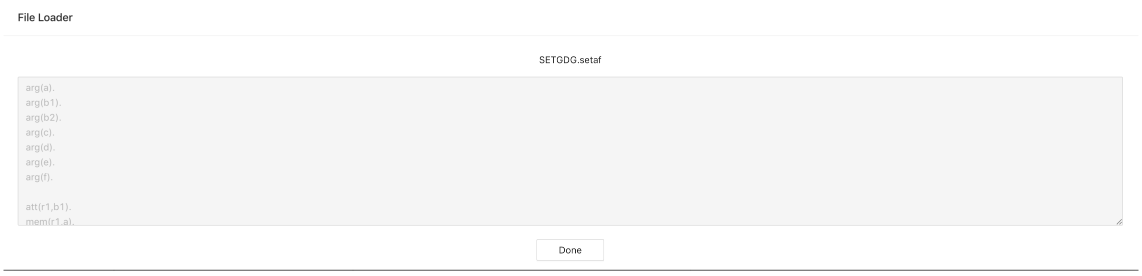

File Loader
Click or drag file to this area to upload
Support for a single or bulk upload. Strictly prohibit from uploading company data or other band files

Figure 4.4: Application interface of file loader



```
arg(a).
arg(b1).
arg(b2).
arg(c).
arg(d).
arg(e).
arg(f).

att(r1,b1).
mem(r1,a).
```

Figure 4.5: Application interface of file reader

Chapter 5

Evaluation

This chapter resolves the question related to the final research goal, which is to evaluate the proposed application. The evaluation will be divided into self-evaluating and users' feedback to maximize the accuracy of the testing. The purpose of the self-evaluation is to check whether any potential threats affect the program's stability during algorithm development, such as extreme SETAF settings and special attacking relations. On the other hand, users' feedback is designed to check the experience of non-professional users when using the web application.

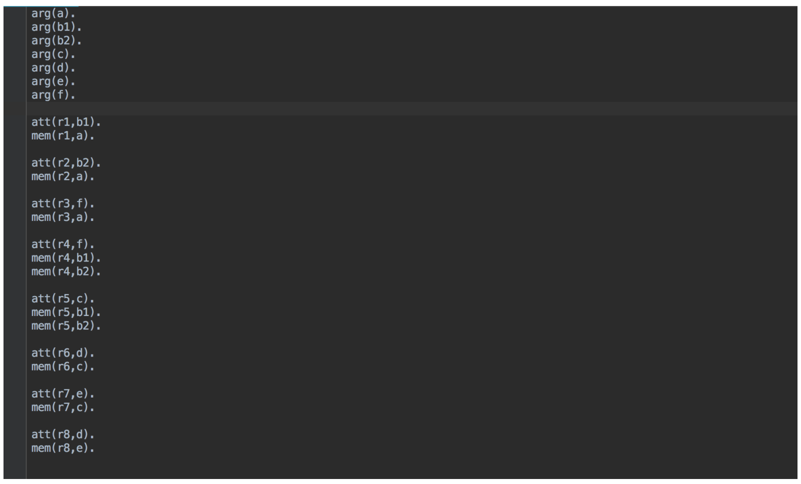
5.1 Self-evaluation

In this section, some typical exceptional cases will be employed to test the accuracy and completeness of SETAF displaying. The evaluation mainly includes two stages. In the first stage (stated in section 5.1.1), I will change repeatable argument or attacking relations in SETAF setting to check whether the behaviours of this application are robust. Then, in the second stage, the author will control attacking relations, including no attacker arguments, self-attacking arguments, and two-way attacking relations, to examine the application's robustness (described in sections 5.1.2-5.1.4).

5.1.1 Input with Repeatable Arguments or Attacking Relations

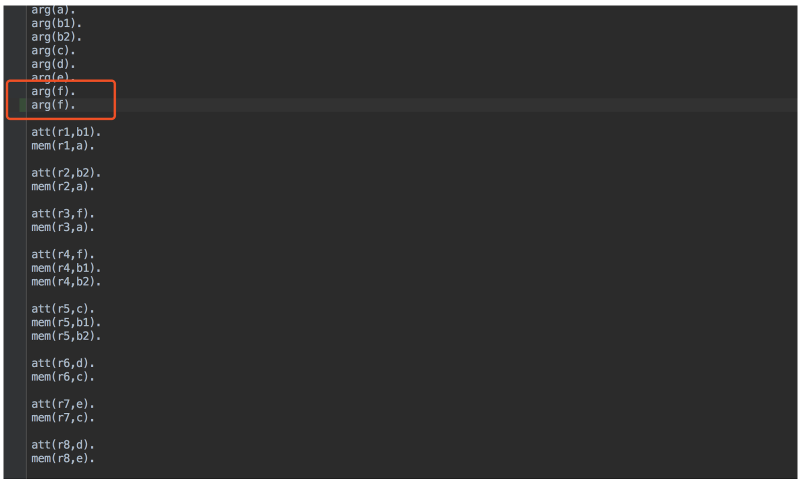
When users start a dialogue game with their own SETAF, there might be a probability that the input files do not fit the SETAF format. The most common reasons that files do not fit the SETAF format are repeated arguments and attacking relations.

To test the accuracy and completeness of SETAF displaying and game experience, the author modified the SETGDG.setaf file in the application (see Figure 5.1) by adding an existing argument (see Figure 5.2) and an existing attacking relation (see Figure 5.3), respectively.



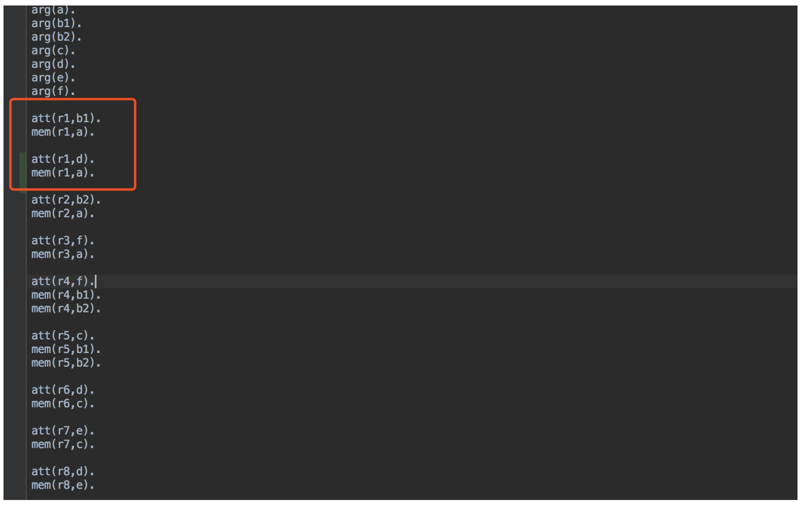
```
arg(a).
arg(b1).
arg(b2).
arg(c).
arg(d).
arg(e).
arg(f).

att(r1,b1).
mem(r1,a).

att(r2,b2).
mem(r2,a).

att(r3,f).
mem(r3,a).

att(r4,f).
mem(r4,b1).
mem(r4,b2).

att(r5,c).
mem(r5,b1).
mem(r5,b2).

att(r6,d).
mem(r6,c).

att(r7,e).
mem(r7,c).

att(r8,d).
mem(r8,e).
```

Figure 5.1: Sample file of SETGDG



```
arg(a).
arg(b1).
arg(b2).
arg(c).
arg(d).
arg(e).
arg(f).
arg(f).

att(r1,b1).
mem(r1,a).

att(r2,b2).
mem(r2,a).

att(r3,f).
mem(r3,a).

att(r4,f).
mem(r4,b1).
mem(r4,b2).

att(r5,c).
mem(r5,b1).
mem(r5,b2).

att(r6,d).
mem(r6,c).

att(r7,e).
mem(r7,c).

att(r8,d).
mem(r8,e).
```

Figure 5.2: Sample file of SETGDG added a repeat argument

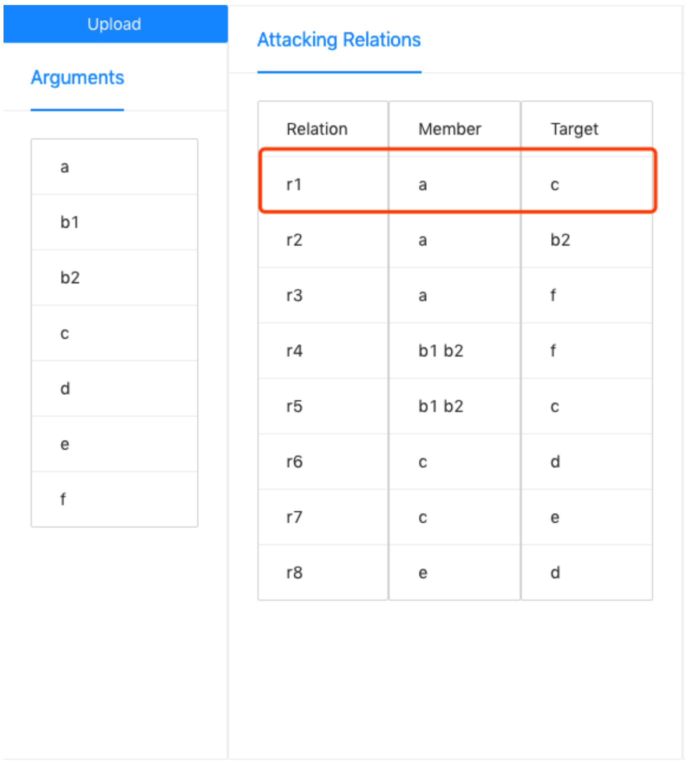


```
arg(a).
arg(b1).
arg(b2).
arg(c).
arg(d).
arg(e).
arg(f).

att(r1,b1).
mem(r1,a).

att(r1,d).
mem(r1,a).

att(r2,b2).
mem(r2,a).

att(r3,f).
mem(r3,a).

att(r4,f).
mem(r4,b1).
mem(r4,b2).

att(r5,c).
mem(r5,b1).
mem(r5,b2).

att(r6,d).
mem(r6,c).

att(r7,e).
mem(r7,c).

att(r8,d).
mem(r8,e).
```

Figure 5.3: Sample file of SETGDG added a repeat attacking relation

In the case shown in Figure 5.2, a new argument named f has been added to the origin file. However, the web tool ends up crashing because arguments have to be unique and do not repeat.

In Figure 5.3, a new attacking relation named $r1$, which means a attacked c, has been added to the origin file. As a result, although the web tool could normally run, as shown in Figure 5.4, an unexpected new problem appears, where the new $r1$ has cover the old one, which means a attacked $b1$. This problem would suggest solving further in the future.



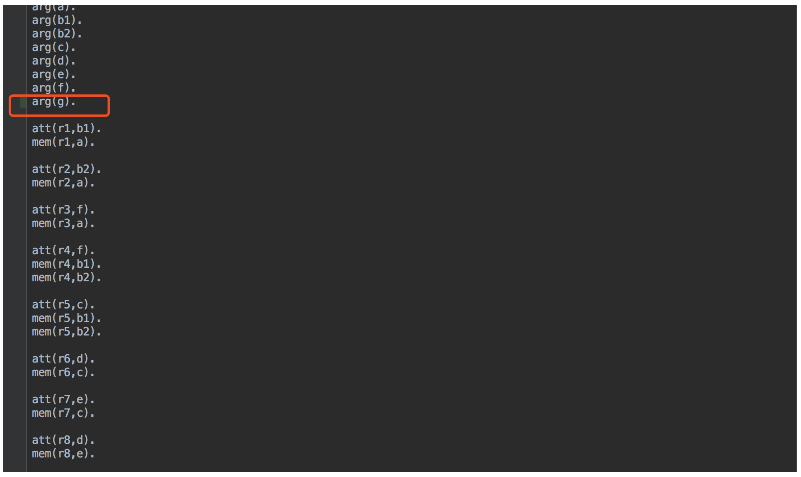
Arguments
a
b1
b2
c
d
e
f

Relation	Member	Target
r1	a	c
r2	a	b2
r3	a	f
r4	b1 b2	f
r5	b1 b2	c
r6	c	d
r7	c	e
r8	e	d

Figure 5.4: Parsing result of SETAF with a repeat attacking relation

5.1.2 Input with No Attacking Relation Arguments

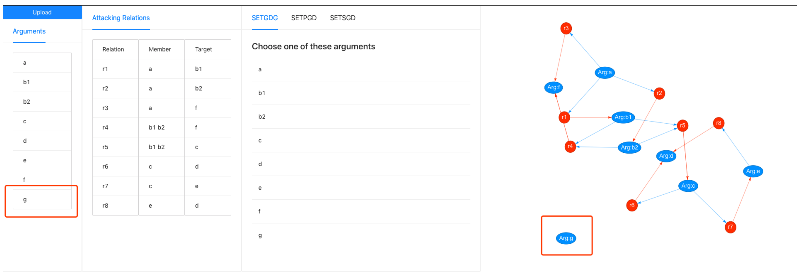
A typical case of special arguments or relations is when a user defines an independent argument without attacking relations in SETAF. Based on the sample SETAF shown in Figure 5.1, a new argument g with no attacking relations has been added as a test case.



```
arg(a).
arg(b1).
arg(b2).
arg(c).
arg(d).
arg(e).
arg(f).
arg(g).

att(r1,b1).
mem(r1,a).

att(r2,b2).
mem(r2,a).

att(r3,f).
mem(r3,a).

att(r4,f).
mem(r4,b1).
mem(r4,b2).

att(r5,c).
mem(r5,b1).
mem(r5,b2).

att(r6,d).
mem(r6,c).

att(r7,e).
mem(r7,c).

att(r8,d).
mem(r8,e).
```

Figure 5.5: Sample file of SETGDG added a new argument with no attacking relation

As the parsing result shows (see Figure 5.6), the independent argument with no attacking relations can be parsed as usual, and the network could be constructed well.



a
b1
b2
c
d
e
f
g

Relation	Member	Target
r1	a	b1
r2	a	b2
r3	a	f
r4	b1 b2	f
r5	b1 b2	c
r6	c	d
r7	c	e
r8	e	d

Choose one of these arguments

a

b1

b2

c

d

e

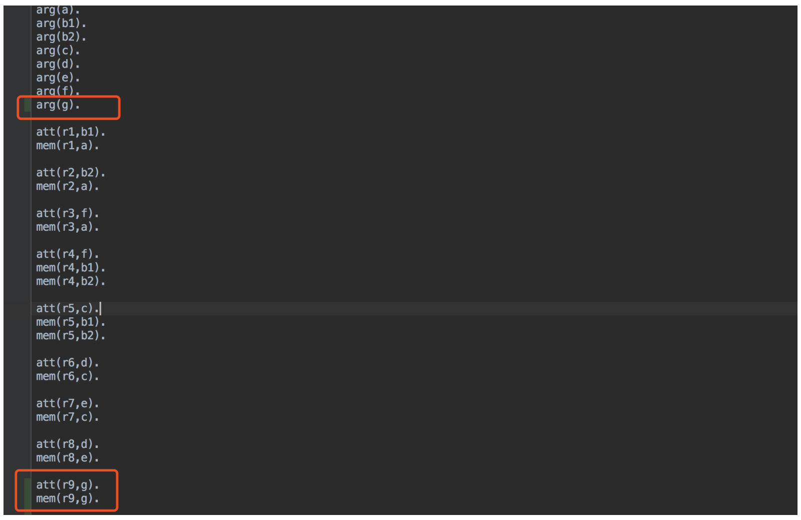
f

g

Figure 5.6: Parsing result of SETAF with a new argument having no attacking relation

5.1.3 Input with Self-attacking Relations

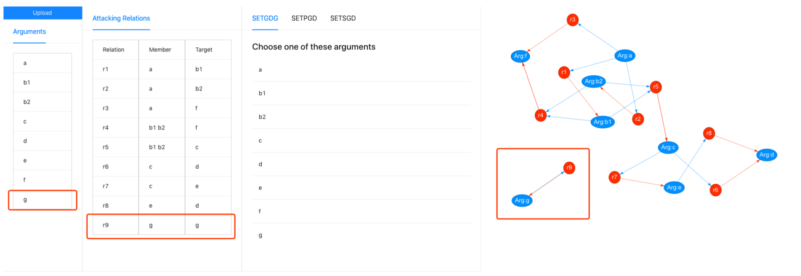
Another particular type of attacking relation is self-attack. Theoretically, this relation is unrealistic, but users' input errors could generate it. Therefore, based on the SETAF shown in Figure 5.7, a new argument *g* and a new attacking relation denote *g* attacking itself have been added.



```
arg(a).
arg(b1).
arg(b2).
arg(c).
arg(d).
arg(e).
arg(f).
arg(g).

att(r1,b1).
mem(r1,a).

att(r2,b2).
mem(r2,a).

att(r3,f).
mem(r3,a).

att(r4,f).
mem(r4,b1).
mem(r4,b2).

att(r5,c).
mem(r5,b1).
mem(r5,b2).

att(r6,d).
mem(r6,c).

att(r7,e).
mem(r7,c).

att(r8,d).
mem(r8,e).

att(r9,g).
mem(r9,g).
```

Figure 5.7: Sample file of SETGDG added a new argument with self-attacking

Figure 5.8 suggests the result of adding a self-attack. The parsing of SETAF goes on well, and the graph of the network of self-attacking relations could be loaded well.



a
b1
b2
c
d
e
f
g

Relation	Member	Target
r1	a	b1
r2	a	b2
r3	a	f
r4	b1 b2	f
r5	b1 b2	c
r6	c	d
r7	c	e
r8	e	d
r9	g	g

Choose one of these arguments

a

b1

b2

c

d

e

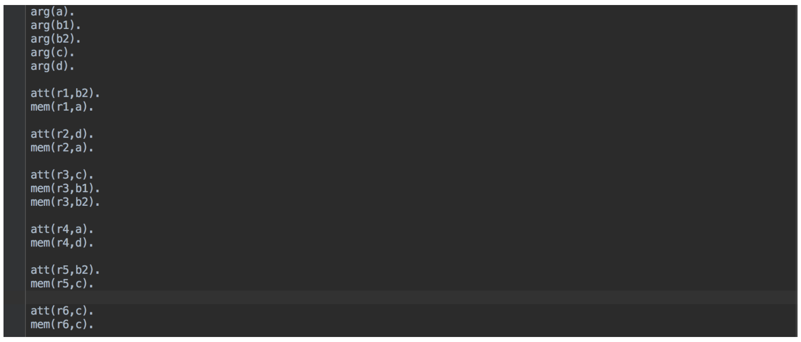
f

g

Figure 5.8: Parsing result of SETAF with a new argument having self-attacking

5.1.4 Input with Two-way Attacking Relations

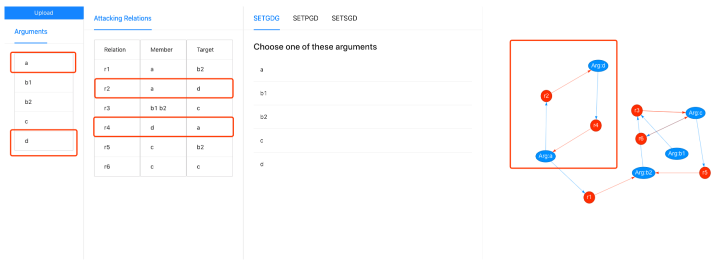
The last special attacking case in this section will discuss the SETAF with two-way attacking relations. The sample file of SETSDG is a good case for testing since SETSDG includes two-way attacking relations. Thus, no modification is needed for the evaluation in this part (Figure 5.9).



```
arg(a).
arg(b1).
arg(b2).
arg(c).
arg(d).

att(r1,b2).
mem(r1,a).

att(r2,d).
mem(r2,a).

att(r3,c).
mem(r3,b1).
mem(r3,b2).

att(r4,a).
mem(r4,d).

att(r5,b2).
mem(r5,c).

att(r6,c).
mem(r6,c).
```

Figure 5.9: Sample file of SETSDG with two-way attacking relations

As shown in Figure 5.10, the parsing of SETAF runs well, and the graph network can be loaded well with two-way attacking relations.



Arguments
a
b1
b2
c
d

Relation	Member	Target
r1	a	b2
r2	a	d
r3	b1 b2	c
r4	d	a
r5	c	b2
r6	c	c

Choose one of these arguments

a

b1

b2

c

d

Figure 5.10: Parsing result of SETAF with two-way attacking relations

5.2 Feedback of Users

This research has invited 30 participants to take the survey using a questionnaire made by Google Form and provide feedback after experiencing the web application. After excluding the responses that showed misunderstanding the core concept and thus failing to give feedback, 21 out of 30 responses are regarded as effective responses and will be included in this analysis.

5.2.1 Design of Questionnaire

The questionnaire was designed in three aspects. The first aspect is to check whether participants understand the core concept of SETAF in order to give accurate feedback. If the respondents correctly answered the question on the core concept, then the next aspect is considered to be tested effectively. The second aspect aims to determine the readability and experience of users when applying the web tool. Finally, the third aspect aims to determine the accuracy of the result of the three games, namely, SETGDG, SETPDG and SETSDG. The details of the questionnaire could be found in Appendix 3.

5.2.2 Analysis of Feedback

As suggested in Figure 5.12-5.13, more than 70% of users reported the user interface to be intuitive (denoted by value 4) or very intuitive (denoted by value 5). In addition, more than 80% of users felt the web application explained SETAF well, and more than 90% of users got a better understanding by visualizing the network of SETAF.

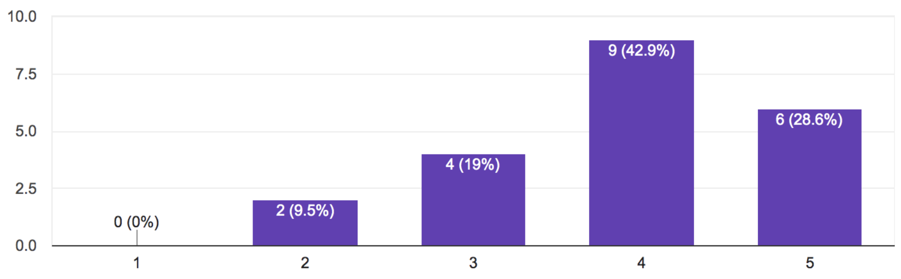


Figure 5.11: How intuitive is the User Interface?

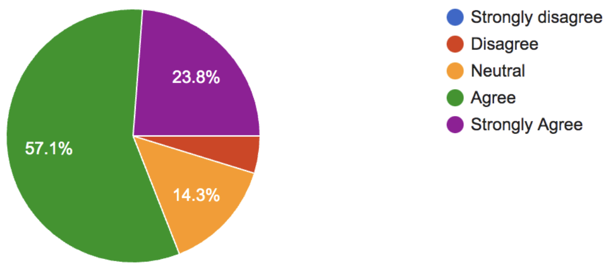


Figure 5.12: Is the SETAF explained well?

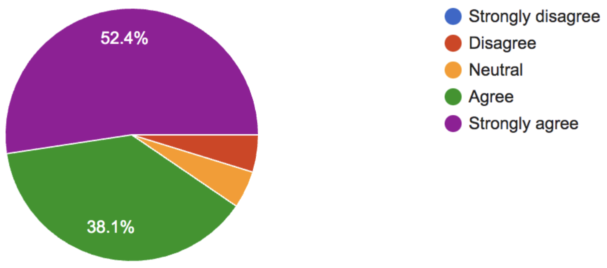


Figure 5.13: Is visualization graph of SETAF meaningful or helpful for understanding?

For the accuracy of the three games, both SETGDG and SETPDG are reported to have similar accuracy since the total fraction of respondents who strongly agree or agree the game being accurate is 85.7% (Figure 5.14-5.16). However, fewer respondents regard SETSDG as accurate as of that of the other two games, since 4.7% of users do not think the result of SETSDG is accurate.

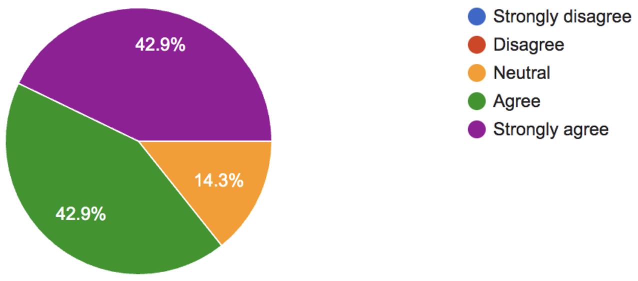


Figure 5.14: Is the SETGDG accurate?

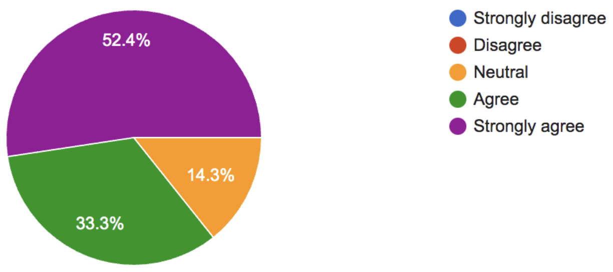


Figure 5.15: Is the SETPDG accurate?

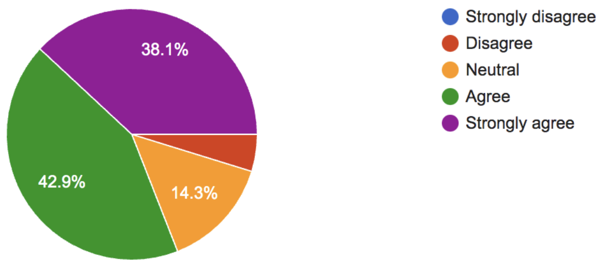


Figure 5.16: Is the SETSDG accurate?

Chapter 6

Conclusions

To summarize, this research aims to generate a proof dialogue application using the new framework SETAF, which is the first attempt to study argumentation. For a quick recapitulation, this thesis has proved the correctness of extending the proof dialogue protocols for the semantics in SETAFs. After then, the author has developed a web application for users to experience more complicated argumentation through dialogue games based on theories of SETAF. Finally, to examine the accuracy of the output of the dialogue game in this web application, the author controlled different variables to check the robustness of the game and created a survey for the public to get users' feedback.

In general, this study has successfully developed the protocol dialogue game using SETAF. The conclusion of the users' feedback turns out to be satisfying, as most users reported being well-experienced in interaction in the web application and accurate results in the games. However, when conducting self-evaluation, potential threats might deteriorate the accuracy or the robustness of the games' result. For example, repeated arguments or attacking relations might end up collapsing the game or distorting the results. These problems need to be further checked in future work.

However, there are still limitations in this study that needs further examination. First, the algorithm of the dialogue games needs to be optimized. For example, in this protocol dialogue, the target arguments in the algorithm are in the form of a set. Yet this form might sophisticate the progress since this algorithm will iterate all the arguments in the set and thus impair the program's efficiency. If the set contains many arguments, the program could run much slower due to the inefficiency of the iterated algorithm. Therefore, the optimization of the algorithm is necessary to generalize the application of the dialogue protocols.

Besides, this paper only focuses on proof dialogue games based on three semantics according to SETAF. Therefore, for future work, proof dialogue games based on other extending theories of AAF are encouraged to explore. For example, it would be interesting to develop proof dialogue games based on additional semantics such as stage, ideal, semi$-$stable, and force.

In addition, the application of Nash Equilibrium in the argumentation system is also worthy of further studying. Bruno's study constructed a theoretical framework based on the Extended Argumentation Framework (EAF) (Yun et al., 2020c). It is also an exciting topic if the Nash Equilibrium could also be employed in SETAF in the future.

Bibliography

Amgoud, L., Belabbes, S., and Prade, H. (2006). A formal general setting for dialogue protocols. In Euzenat, J. and Domingue, J., editors, *Artificial Intelligence: Methodology, Systems, and Applications, 12th International Conference, AIMSA 2006, Varna, Bulgaria, September 12-15, 2006, Proceedings*, volume 4183 of *Lecture Notes in Computer Science*, pages 13–23. Springer.

Baroni, P., Caminada, M., and Giacomin, M. (2011). An introduction to argumentation semantics. *Knowl. Eng. Rev.*, 26(4):365–410.

Caminada, M. (2006). Semi-stable semantics. In Dunne, P. E. and Bench-Capon, T. J. M., editors, *Computational Models of Argument: Proceedings of COMMA 2006, September 11-12, 2006, Liverpool, UK*, volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 121–130. IOS Press.

Caminada, M. (2011). A labelling approach for ideal and stage semantics. *Argument Comput.*, 2(1):1–21.

Caminada, M. (2015). A discussion game for grounded semantics. In Black, E., Modgil, S., and Oren, N., editors, *Theory and Applications of Formal Argumentation - Third International Workshop, TAFA 2015, Buenos Aires, Argentina, July 25-26, 2015, Revised Selected Papers*, volume 9524 of *Lecture Notes in Computer Science*, pages 59–73. Springer.

Caminada, M. and Wu, Y. (2009). An argument game for stable semantics. *Log. J. IGPL*, 17(1):77–90.

Caminada, M. W. A. and Gabbay, D. M. (2009). A logical account of formal argumentation. *Stud Logica*, 93(2-3):109–145.

Cayrol, C., Doutre, S., and Mengin, J. (2001). Dialectical proof theories for the credulous preferred semantics of argumentation frameworks. In Benferhat, S. and Besnard, P., editors, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 6th European Conference, ECSQARU 2001, Toulouse, France, September 19-21, 2001, Proceedings*, volume 2143 of *Lecture Notes in Computer Science*, pages 668–679. Springer.

Devred, C. and Doutre, S. (2007). Dialectical proof theories for the credulous prudent preferred semantics of argumentation. In Mellouli, K., editor, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 9th European Conference, ECSQARU 2007, Hammamet, Tunisia, October 31 - November 2, 2007, Proceedings*, volume 4724 of *Lecture Notes in Computer Science*, pages 271–282. Springer.

Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358.

Dvorák, W., Greßler, A., and Woltran, S. (2018). Evaluating setafs via answer-set programming. In Thimm, M., Cerutti, F., and Vallati, M., editors, *Proceedings of the Second International*

Workshop on Systems and Algorithms for Formal Argumentation (SAFA 2018) co-located with the 7th International Conference on Computational Models of Argument (COMMA 2018), Warsaw, Poland, September 11, 2018, volume 2171 of *CEUR Workshop Proceedings*, pages 10–21. CEUR-WS.org.

Flouris, G. and Bikakis, A. (2019). A comprehensive study of argumentation frameworks with sets of attacking arguments. *Int. J. Approx. Reason.*, 109:55–86.

McCarthy, J. (1980). Circumscription - A form of non-monotonic reasoning. *Artif. Intell.*, 13(1-2):27–39.

Modgil, S. and Caminada, M. (2009). Proof theories and algorithms for abstract argumentation frameworks. In Simari, G. R. and Rahwan, I., editors, *Argumentation in Artificial Intelligence*, pages 105–129. Springer.

Nielsen, S. H. and Parsons, S. (2006). A generalization of dung's abstract framework for argumentation: Arguing with sets of attacking arguments. In Maudet, N., Parsons, S., and Rahwan, I., editors, *Argumentation in Multi-Agent Systems, Third International Workshop, ArgMAS 2006, Hakodate, Japan, May 8, 2006, Revised Selected and Invited Papers*, volume 4766 of *Lecture Notes in Computer Science*, pages 54–73. Springer.

Sapino, M., Ferretti, E., Dondena, L. M., and Errecalde, M. (2020). Modeling human decision making with an abstract dynamic argumentation framework. In Pesado, P. and Eterovic, J., editors, *Computer Science - CACIC 2020 - 26th Argentine Congress, CACIC 2020, San Justo, Buenos Aires, Argentina, October 5-9, 2020, Revised Selected Papers*, volume 1409 of *Communications in Computer and Information Science*, pages 3–18. Springer.

Shams, Z. and Oren, N. (2016). A two-phase dialogue game for skeptical preferred semantics. In Michael, L. and Kakas, A. C., editors, *Logics in Artificial Intelligence - 15th European Conference, JELIA 2016, Larnaca, Cyprus, November 9-11, 2016, Proceedings*, volume 10021 of *Lecture Notes in Computer Science*, pages 570–576.

Siegel, G. J. (1954). Letter to the editor - the application of the operations-research approach in making a decision of no great importance. *Oper. Res.*, 2(4):446.

Strasser, C. and Antonelli, G. A. (2019). Non-monotonic Logic. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2019 edition.

Verheij, B. (1999). Two approaches to dialectical argumentation: Admissible sets and argumentation stages.

Vreeswijk, G. and Prakken, H. (2000). Credulous and sceptical argument games for preferred semantics. In Ojeda-Aciego, M., de Guzmán, I. P., Brewka, G., and Pereira, L. M., editors, *Logics in Artificial Intelligence, European Workshop, JELIA 2000 Malaga, Spain, September 29 - October 2, 2000, Proceedings*, volume 1919 of *Lecture Notes in Computer Science*, pages 239–253. Springer.

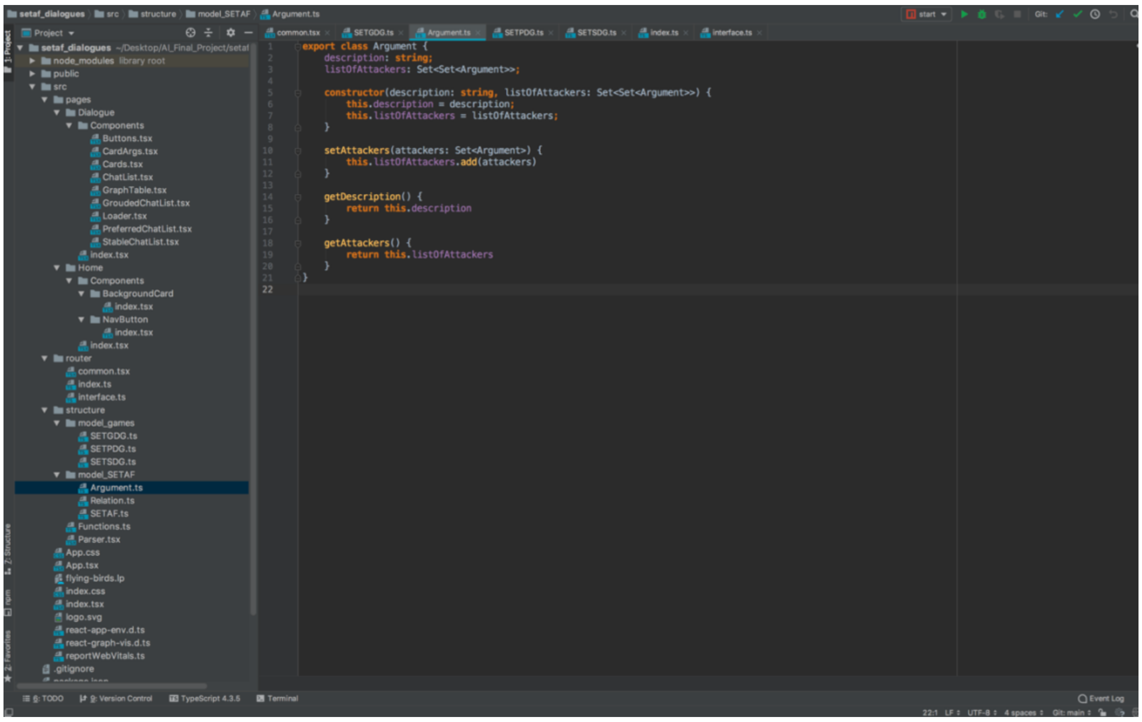
Yun, B., Vesic, S., and Croitoru, M. (2020a). Ranking-based semantics for sets of attacking arguments. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3033–3040. AAAI Press.

Yun, B., Vesic, S., and Croitoru, M. (2020b). Sets of attacking arguments for inconsistent datalog knowledge bases. In Prakken, H., Bistarelli, S., Santini, F., and Taticchi, C., editors, *Computational Models of Argument - Proceedings of COMMA 2020, Perugia, Italy, September 4-11, 2020*, volume 326 of *Frontiers in Artificial Intelligence and Applications*, pages 419–430. IOS Press.

Yun, B., Vesic, S., and Oren, N. (2020c). Representing pure nash equilibria in argumentation. *CoRR*, abs/2006.11020.

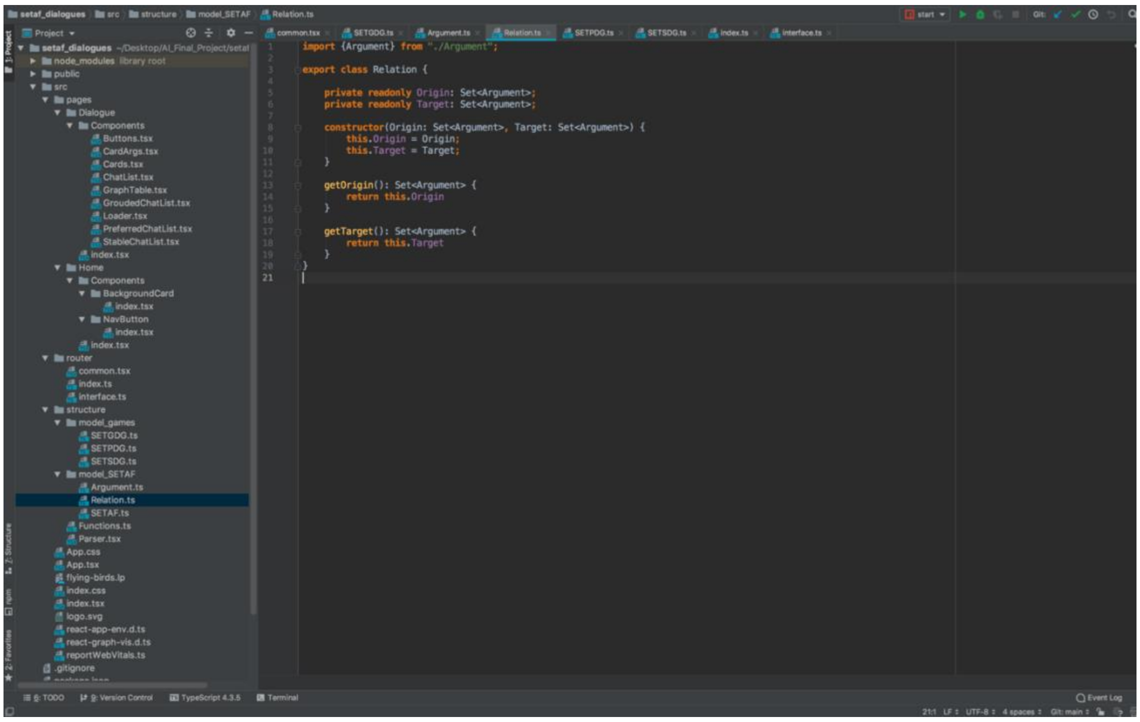
Appendix 1 Steps and Codes for Setting Up SETAF Model

CODE SNIPPETS FOR MODEL SETUP



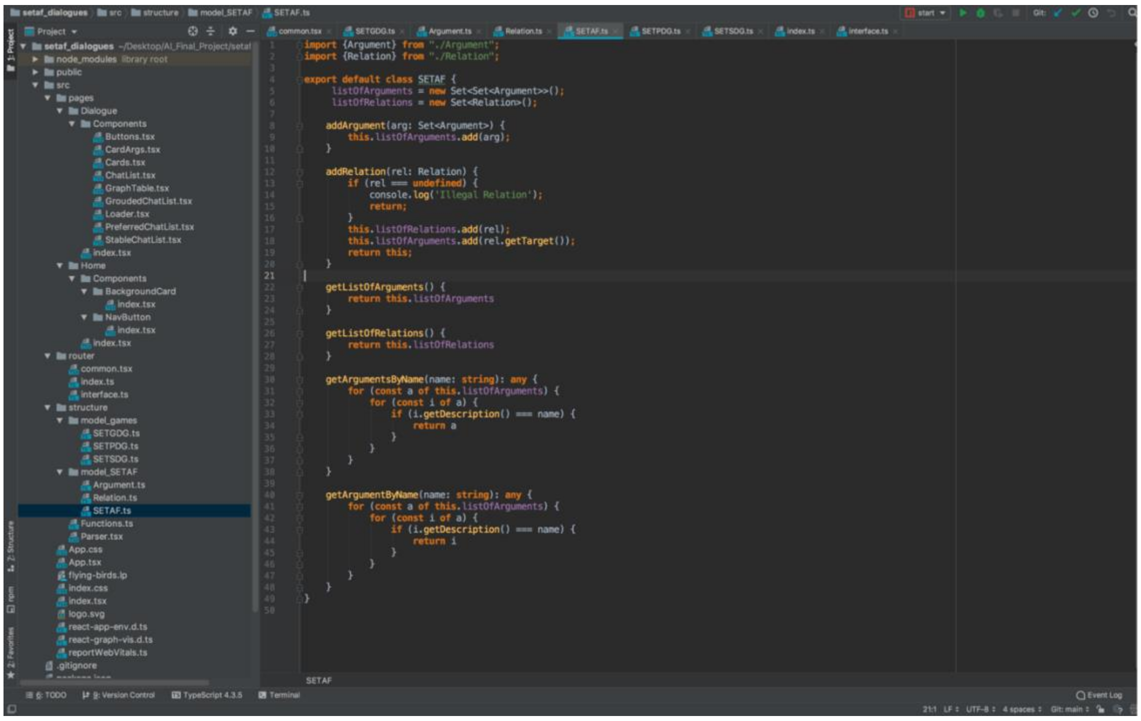
```
export class Argument {
    description: string;
    listOfAttackers: Set<Set<Argument>>;

    constructor(description: string, listOfAttackers: Set<Set<Argument>>) {
        this.description = description;
        this.listOfAttackers = listOfAttackers;
    }

    setAttackers(attackers: Set<Argument>) {
        this.listOfAttackers.add(attackers)
    }

    getDescription() {
        return this.description
    }

    getAttackers() {
        return this.listOfAttackers
    }
}
```

- Properties:
  - description: means the description of the Argument.
  - listOfAttackers: is the attackers set of the Argument.
- Functions
  - setAttackers: is prepared for the initialisation of arguments during file parsing.
  - getDescription: can return the description of the Argument.
  - getAttackers: can return the attackers set of the Argument.



```
import {Argument} from "./Argument";

export class Relation {

    private readonly Origin: Set<Argument>;
    private readonly Target: Set<Argument>;

    constructor(Origin: Set<Argument>, Target: Set<Argument>) {
        this.Origin = Origin;
        this.Target = Target;
    }

    getOrigin(): Set<Argument> {
        return this.Origin
    }

    getTarget(): Set<Argument> {
        return this.Target
    }
}
```

- Properties:
  - Origin: means the attacker set.
  - Target: means the set of target arguments that is attacked in the attacking relation. Please note that the target set can have only one Argument in SETAF. However, it is easier to build a SETAF model using an argument model set instead of an argument model.
- Constructor: is a piece of code that will be executed during model initialisation. Each time the attacking relation model is instantiated, it receives parameters passed in from outside the model and assigns them to its own instance.
- Functions
  - getOrigin: can return the attacker set of the argument instance.
  - getTarget: can return the target argument set of the argument instance.

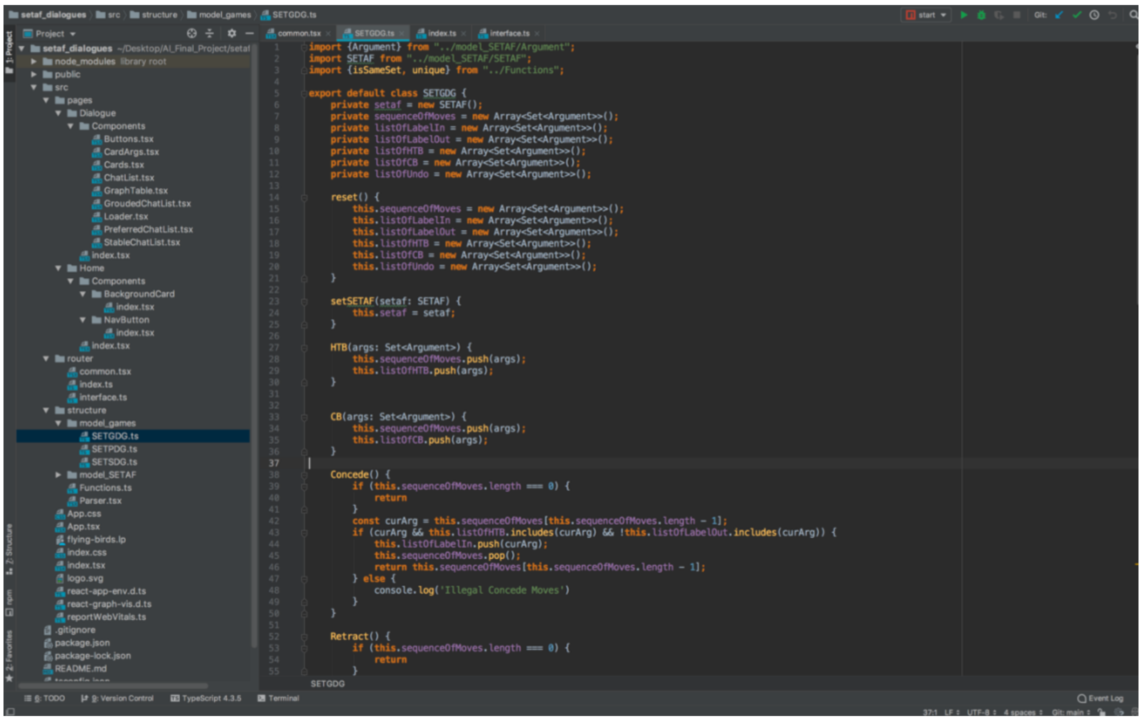


```typescript
import {Argument} from "./Argument";
import {Relation} from "./Relation";

export default class SETAF {
     listOfArguments = new Set<Set<Argument>>();
     listOfRelations = new Set<Relation>();

    addArgument(arg: Set<Argument>) {
        this.listOfArguments.add(arg);
    }

    addRelation(rel: Relation) {
        if (rel === undefined) {
            console.log('Illegal Relation');
            return;
        }
        this.listOfRelations.add(rel);
        this.listOfArguments.add(rel.getTarget());
        return this;
    }

    getListOfArguments() {
        return this.listOfArguments
    }

    getListOfRelations() {
        return this.listOfRelations
    }

    getArgumentsByName(name: string): any {
        for (const a of this.listOfArguments) {
            for (const i of a) {
                if (i.getDescription() === name) {
                    return a
                }
            }
        }
    }

    getArgumentByName(name: string): any {
        for (const a of this.listOfArguments) {
            for (const i of a) {
                if (i.getDescription() === name) {
                    return i
                }
            }
        }
    }
}
```

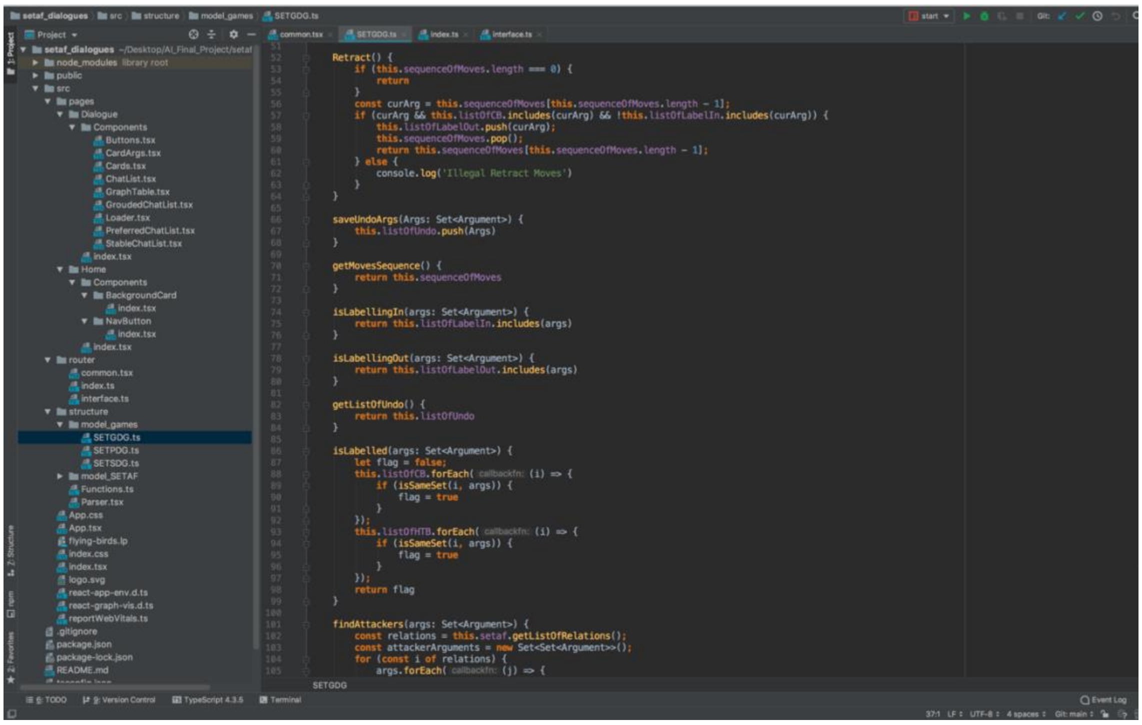
- Properties:
  - listOfArguments: is a container for all the arguments instances.
  - listOfRelations: is a container of all the attacking relations instances.
- Functions
  - addArgument: can add an argument instance to SETAF.
  - addRelation:can add an attacking relation instance to SETAF.
  - getListOfArguments: can return the container for argument instances.
  - getListOfRelations: can return the container for attacking relation instances.
  - getArgumentsByName: can return a set of argument instances found by name or prescription.
  - getArgumentByName: can return an argument instances found by name or description.

Appendix 2 Codes for Discussion Games

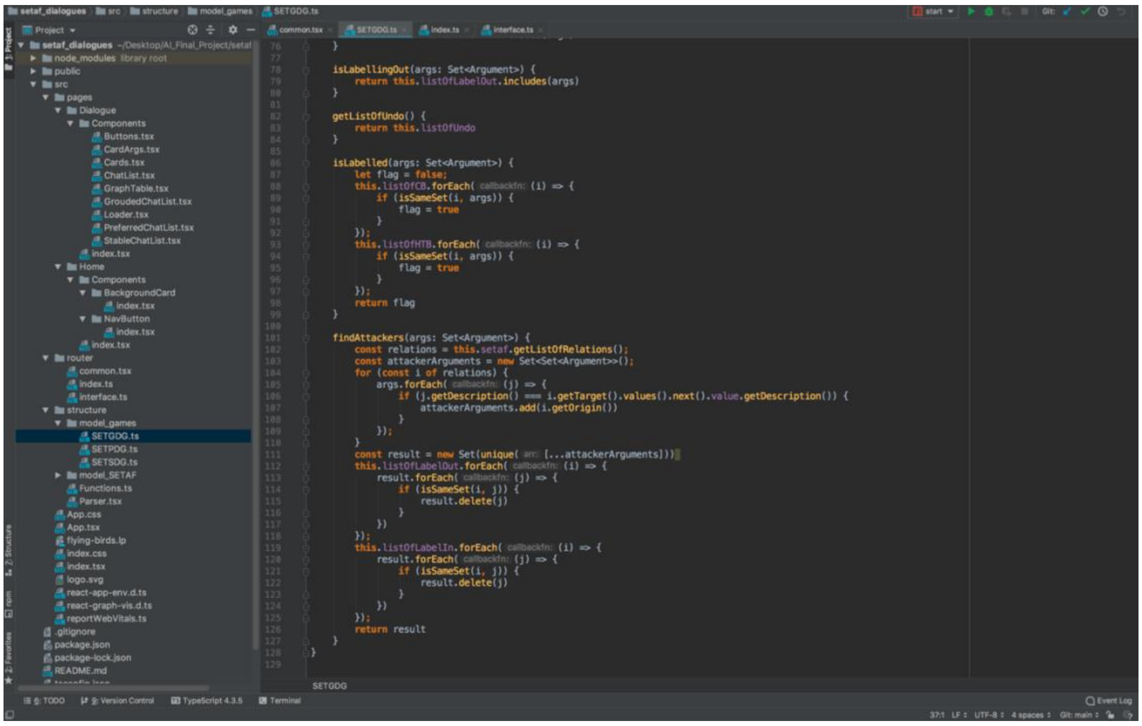
CODE SNIPPETS OF SETGDG



```
import {Argument} from "../model_SETAF/Argument";
import SETAF from "../model_SETAF/SETAF";
import {isSameSet, unique} from "../Functions";

export default class SETGDG {
    private setaf = new SETAF();
    private sequenceOfMoves = new Array<Set<Argument>>();
    private listOfLabelIn = new Array<Set<Argument>>();
    private listOfLabelOut = new Array<Set<Argument>>();
    private listOfHTB = new Array<Set<Argument>>();
    private listOfCB = new Array<Set<Argument>>();
    private listOfUndo = new Array<Set<Argument>>();

    reset() {
        this.sequenceOfMoves = new Array<Set<Argument>>();
        this.listOfLabelIn = new Array<Set<Argument>>();
        this.listOfLabelOut = new Array<Set<Argument>>();
        this.listOfHTB = new Array<Set<Argument>>();
        this.listOfCB = new Array<Set<Argument>>();
        this.listOfUndo = new Array<Set<Argument>>();
    }

    setSETAF(setaf: SETAF) {
        this.setaf = setaf;
    }

    HTB(args: Set<Argument>) {
        this.sequenceOfMoves.push(args);
        this.listOfHTB.push(args);
    }


    CB(args: Set<Argument>) {
        this.sequenceOfMoves.push(args);
        this.listOfCB.push(args);
    }

    Concede() {
        if (this.sequenceOfMoves.length === 0) {
            return
        }
        const curArg = this.sequenceOfMoves[this.sequenceOfMoves.length - 1];
        if (curArg && this.listOfHTB.includes(curArg) && !this.listOfLabelOut.includes(curArg)) {
            this.listOfLabelIn.push(curArg);
            this.sequenceOfMoves.pop();
            return this.sequenceOfMoves[this.sequenceOfMoves.length - 1];
        } else {
            console.log('Illegal Concede Moves')
        }
    }

    Retract() {
        if (this.sequenceOfMoves.length === 0) {
            return
        }
```



```
    Retract() {
        if (this.sequenceOfMoves.length === 0) {
            return
        }
        const curArg = this.sequenceOfMoves[this.sequenceOfMoves.length - 1];
        if (curArg && this.listOfCB.includes(curArg) && !this.listOfLabelIn.includes(curArg)) {
            this.listOfLabelOut.push(curArg);
            this.sequenceOfMoves.pop();
            return this.sequenceOfMoves[this.sequenceOfMoves.length - 1];
        } else {
            console.log('Illegal Retract Moves')
        }
    }

    saveUndoArgs(Args: Set<Argument>) {
        this.listOfUndo.push(Args)
    }

    getMovesSequence() {
        return this.sequenceOfMoves
    }

    isLabellingIn(args: Set<Argument>) {
        return this.listOfLabelIn.includes(args)
    }

    isLabellingOut(args: Set<Argument>) {
        return this.listOfLabelOut.includes(args)
    }

    getListOfUndo() {
        return this.listOfUndo
    }

    isLabelled(args: Set<Argument>) {
        let flag = false;
        this.listOfCB.forEach( callbackfn: (i) => {
            if (isSameSet(i, args)) {
                flag = true
            }
        });
        this.listOfHTB.forEach( callbackfn: (i) => {
            if (isSameSet(i, args)) {
                flag = true
            }
        });
        return flag
    }

    findAttackers(args: Set<Argument>) {
        const relations = this.setaf.getListOfRelations();
        const attackerArguments = new Set<Set<Argument>>();
        for (const i of relations) {
            args.forEach( callbackfn: (j) => {
```



```
    }

    isLabellingOut(args: Set<Argument>) {
        return this.listOfLabelOut.includes(args)
    }

    getListOfUndo() {
        return this.listOfUndo
    }

    isLabelled(args: Set<Argument>) {
        let flag = false;
        this.listOfCB.forEach( callbackfn: (i) => {
            if (isSameSet(i, args)) {
                flag = true
            }
        });
        this.listOfHTB.forEach( callbackfn: (i) => {
            if (isSameSet(i, args)) {
                flag = true
            }
        });
        return flag
    }

    findAttackers(args: Set<Argument>) {
        const relations = this.setaf.getListOfRelations();
        const attackerArguments = new Set<Set<Argument>>();
        for (const i of relations) {
            args.forEach( callbackfn: (j) => {
                if (j.getDescription() === i.getTarget().values().next().value.getDescription()) {
                    attackerArguments.add(i.getOrigin())
                }
            });
        }
        const result = new Set(unique( arr: [...attackerArguments]))
        this.listOfLabelOut.forEach( callbackfn: (i) => {
            result.forEach( callbackfn: (j) => {
                if (isSameSet(i, j)) {
                    result.delete(j)
                }
            })
        });
        this.listOfLabelIn.forEach( callbackfn: (i) => {
            result.forEach( callbackfn: (j) => {
                if (isSameSet(i, j)) {
                    result.delete(j)
                }
            })
        });
        return result
    }
}
```

Here following is the detailed information for SETGDG models.

- Properties:
  - setaf: is the parsed SETAF instance.
  - sequenceOfMoves: is an argument set that records the order of all the moves.
  - listOfLabelIn: is an argument set that is used to store arguments labelled IN.
  - listOfLabelOut: is an argument set that is used to store arguments labelled OUT.
  - listOfHTB: is a set used to store the arguments from HTB moves.
  - listOfCB: is a set used to store the arguments from CB moves.
  - listOfUndo: is used to temporarily store arguments that have not yet been labelled.
- Functions:
  - reset: is a function to reset all the set in SETGDG.
  - setSETAF: is a function to set a SETAF to SETGDG instance.
  - HTB: is the function of HTB moves.
  - CB: is the function of CB moves.
  - Concede: is the function of Concede moves.
  - Retract: is the function of Retract moves.

– saveUndoArgs: is used to save an argument to the set listOfUndo.

– getMovesSequence: can return the set sequenceOfMoves.

– isLabellingIn: can return a boolean about whether the Argument belongs to listOfLabelIn.

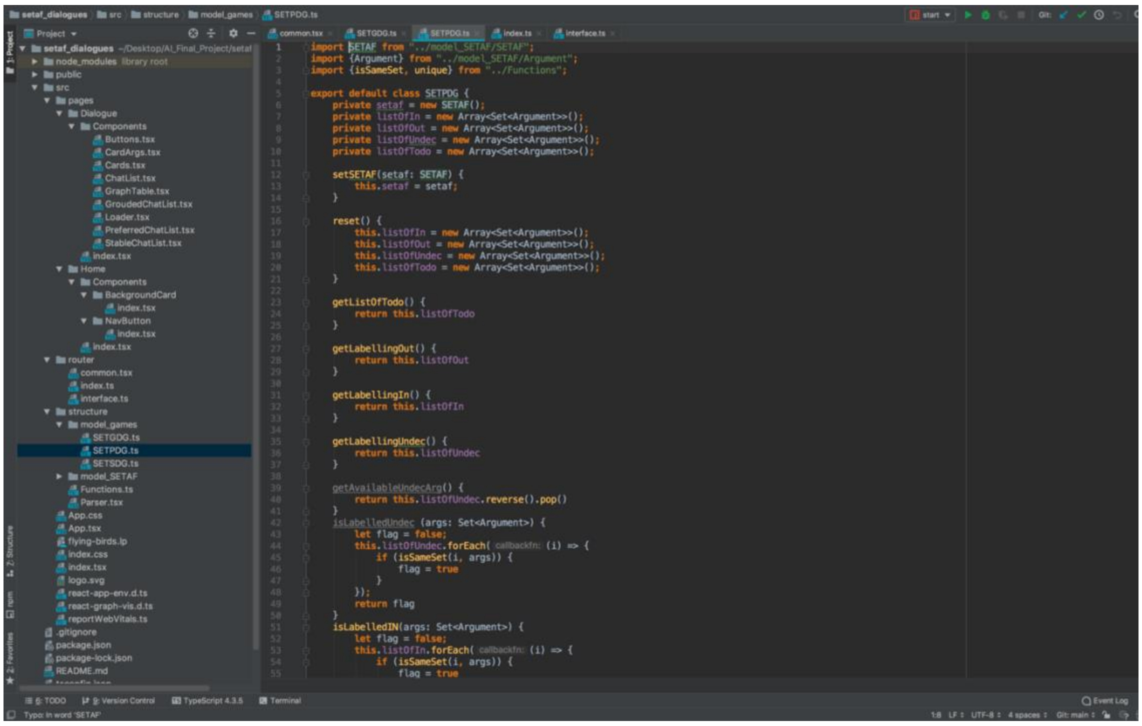
– isLabellingOut: can return a boolean about whether the Argument belongs to listOfLabelOut.

– getListOfUndo: can return the set listOfUndo.

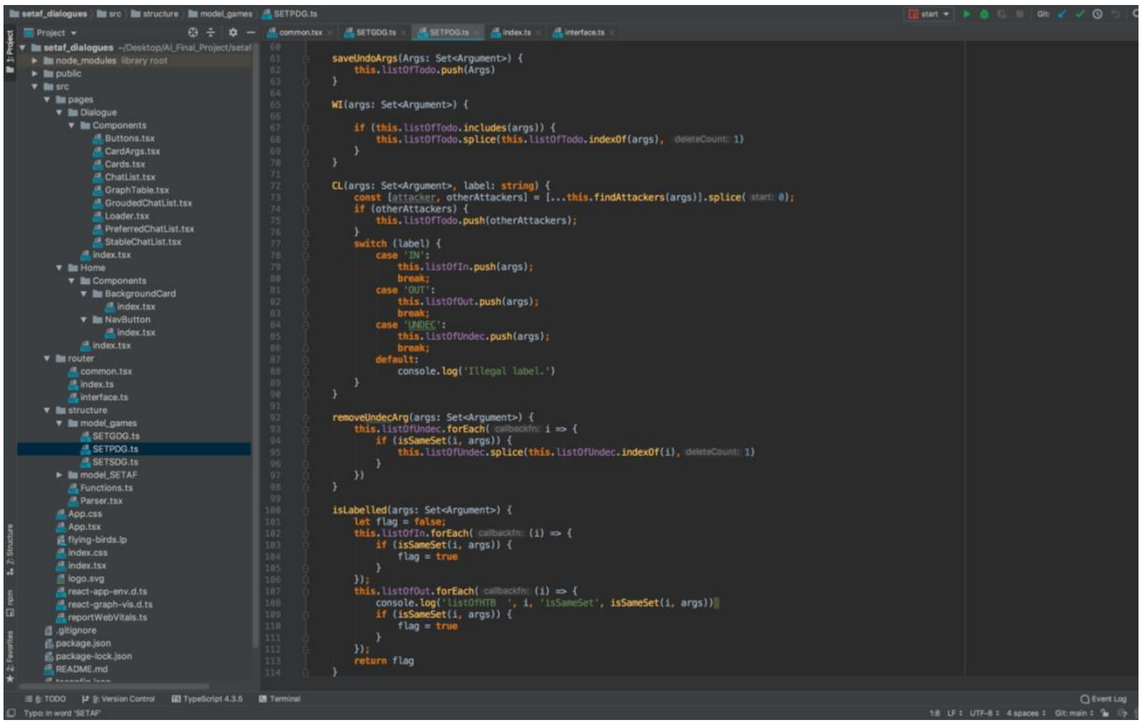
– isLabelled: is used to judge whether an argument has been labelled.

– findAttackers: can return an argument set that has attacked the target argument.

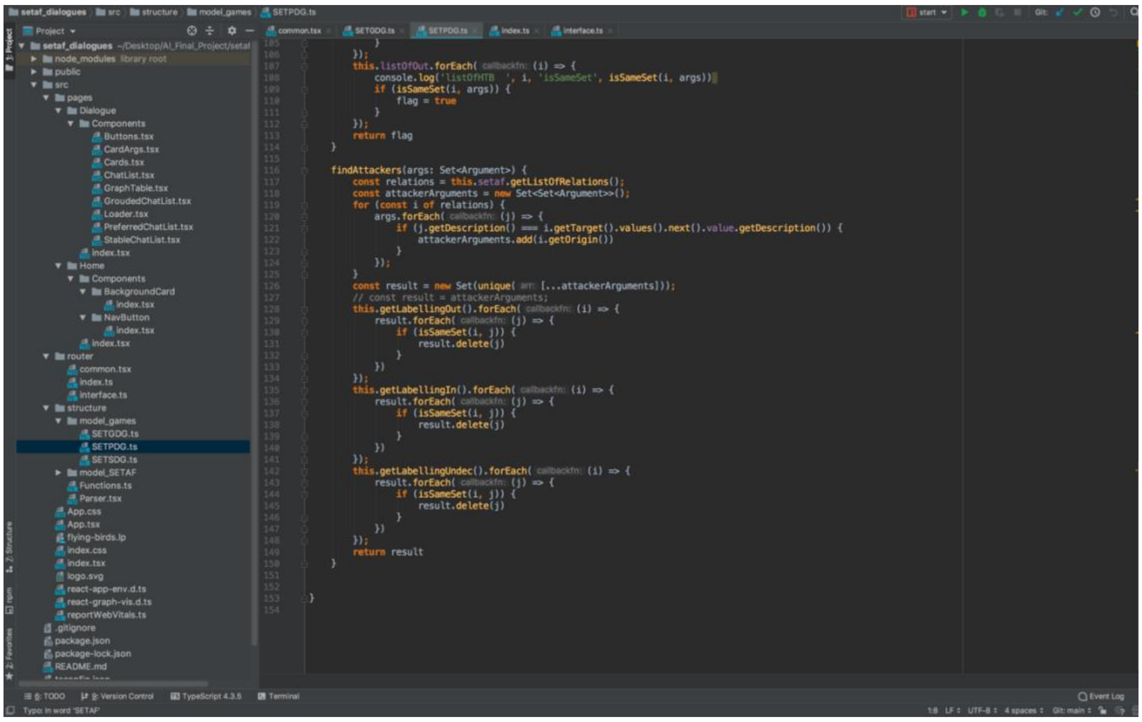
CODE SNIPPETS OF SETPDG



```
import SETAF from "../model_SETAF/SETAF";
import {Argument} from "../model_SETAF/Argument";
import {isSameSet, unique} from "../Functions";

export default class SETPDG {
    private setaf = new SETAF();
    private listOfIn = new Array<Set<Argument>>();
    private listOfOut = new Array<Set<Argument>>();
    private listOfUndec = new Array<Set<Argument>>();
    private listOfTodo = new Array<Set<Argument>>();

    setSETAF(setaf: SETAF) {
        this.setaf = setaf;
    }

    reset() {
        this.listOfIn = new Array<Set<Argument>>();
        this.listOfOut = new Array<Set<Argument>>();
        this.listOfUndec = new Array<Set<Argument>>();
        this.listOfTodo = new Array<Set<Argument>>();
    }

    getListOfTodo() {
        return this.listOfTodo
    }

    getLabellingOut() {
        return this.listOfOut
    }

    getLabellingIn() {
        return this.listOfIn
    }

    getLabellingUndec() {
        return this.listOfUndec
    }

    getAvailableUndecArg() {
        return this.listOfUndec.reverse().pop()
    }
    isLabelledUndec (args: Set<Argument>) {
        let flag = false;
        this.listOfUndec.forEach( callbackfn: (i) => {
            if (isSameSet(i, args)) {
                flag = true
            }
        });
        return flag
    }
    isLabelledIN(args: Set<Argument>) {
        let flag = false;
        this.listOfIn.forEach( callbackfn: (i) => {
            if (isSameSet(i, args)) {
                flag = true
```



```

    saveUndoArgs(Args: Set<Argument>) {
        this.listOfTodo.push(Args)
    }

    WI(args: Set<Argument>) {

        if (this.listOfTodo.includes(args)) {
            this.listOfTodo.splice(this.listOfTodo.indexOf(args),  deleteCount: 1)
        }
    }

    CL(args: Set<Argument>, label: string) {
        const [attacker, otherAttackers] = [...this.findAttackers(args)].splice( start: 0);
        if (otherAttackers) {
            this.listOfTodo.push(otherAttackers);
        }
        switch (label) {
            case 'IN':
                this.listOfIn.push(args);
                break;
            case 'OUT':
                this.listOfOut.push(args);
                break;
            case 'UNDEC':
                this.listOfUndec.push(args);
                break;
            default:
                console.log('Illegal label.')
        }
    }

    removeUndecArg(args: Set<Argument>) {
        this.listOfUndec.forEach( callbackfn: i => {
            if (isSameSet(i, args)) {
                this.listOfUndec.splice(this.listOfUndec.indexOf(i), deleteCount: 1)
            }
        })
    }

    isLabelled(args: Set<Argument>) {
        let flag = false;
        this.listOfIn.forEach( callbackfn: (i) => {
            if (isSameSet(i, args)) {
                flag = true
            }
        });
        this.listOfOut.forEach( callbackfn: (i) => {
            console.log('listOfHTB  ', i, 'isSameSet', isSameSet(i, args))
            if (isSameSet(i, args)) {
                flag = true
            }
        });
        return flag
    }
```



```
        }
    });
    this.listOfOut.forEach( (i) => {
        console.log('listOfHTB  ', i, 'isSameSet', isSameSet(i, args))
        if (isSameSet(i, args)) {
            flag = true
        }
    });
    return flag
}

findAttackers(args: Set<Argument>) {
    const relations = this.setaf.getListOfRelations();
    const attackerArguments = new Set<Set<Argument>>();
    for (const i of relations) {
        args.forEach( (j) => {
            if (j.getDescription() === i.getTarget().values().next().value.getDescription()) {
                attackerArguments.add(i.getOrigin())
            }
        });
    }
    const result = new Set(unique( [...attackerArguments]));
    // const result = attackerArguments;
    this.getLabellingOut().forEach( (i) => {
        result.forEach( (j) => {
            if (isSameSet(i, j)) {
                result.delete(j)
            }
        })
    });
    this.getLabellingIn().forEach( (i) => {
        result.forEach( (j) => {
            if (isSameSet(i, j)) {
                result.delete(j)
            }
        })
    });
    this.getLabellingUndec().forEach( (i) => {
        result.forEach( (j) => {
            if (isSameSet(i, j)) {
                result.delete(j)
            }
        })
    });
    return result
}


}
```

Here below is the detailed information for SETPDG model.

- Properties:
  - setaf: is the parsed SETAF instance.
  - listOfIn: is an arguments set used to store arguments labelled IN.
  - listOfOut: is an arguments set used to store arguments labelled OUT.
  - listOfUndec: is an arguments set used to store arguments labelled UNDEC.
  - listOfTodo: is an arguments set used to store arguments that haven't been labelled.
- Functions:
  - setSETAF: is a function to set SETAF within this model.
  - reset: is a function to reset all the label collections
  - getListOfTodo: returns the set listOfTodo.
  - getLabellingOut: returns the set listOfOut.
  - getLabellingIn: returns the set listOfIn.
  - getLabellingUndec: returns the set listOfUndec.
  - getAvailableUndecArg: removes and returns the last Argument in listOfUndec.
  - isLabelledUndec: returns a boolean that determines whether an argument has been labelled UNDEC.
  - isLabelledIN: returns a boolean that determines whether an argument has been labelled IN.

– saveUndoArgs: is a function to save one Argument to the set listOfTodo.

– WI: is the function of WI move.

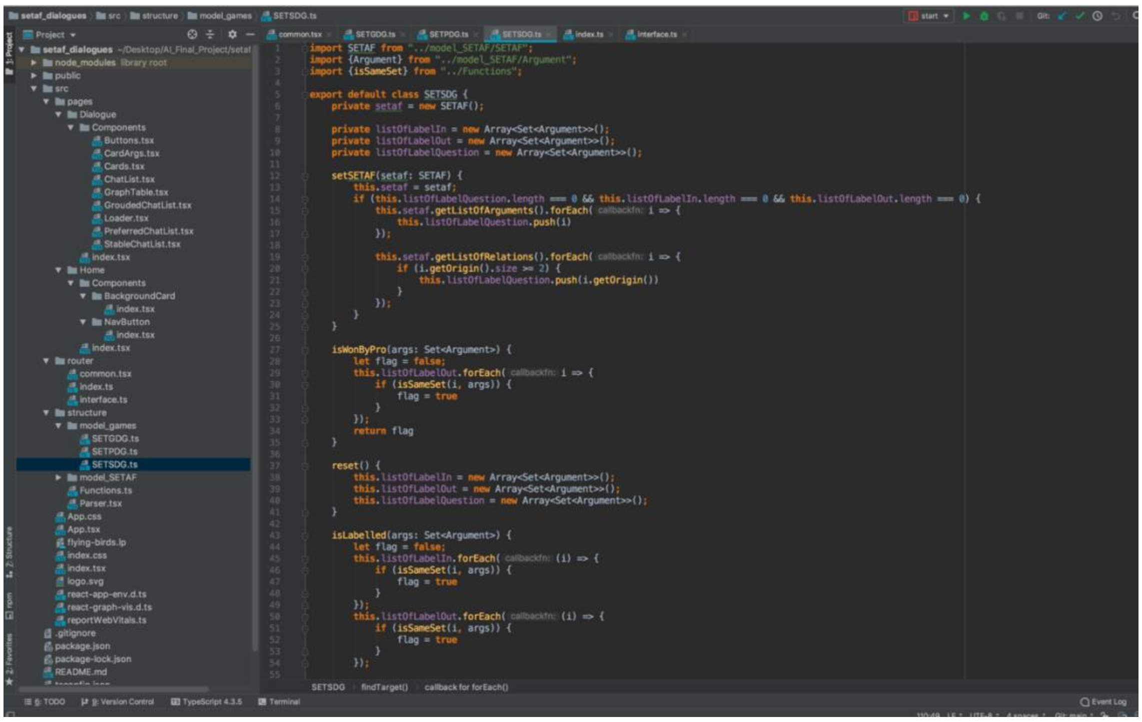
– CL: is the function of CL move, which will add Argument to the specified set according to the label parameter.

– removeUndecArg: removes the specified Argument from listOfUndec.

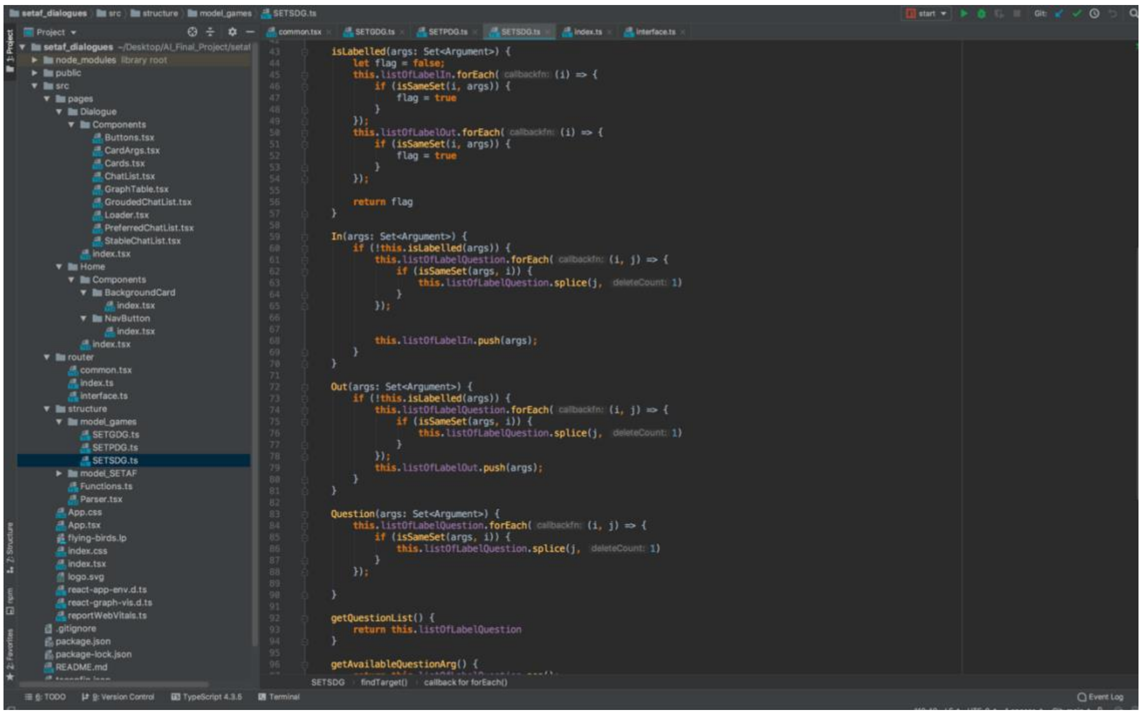
– isLabelled: returns a boolean that determines whether an argument has been labelled IN or OUT.

– findAttackers: can return an argument set that has attacked the tartget Argument.

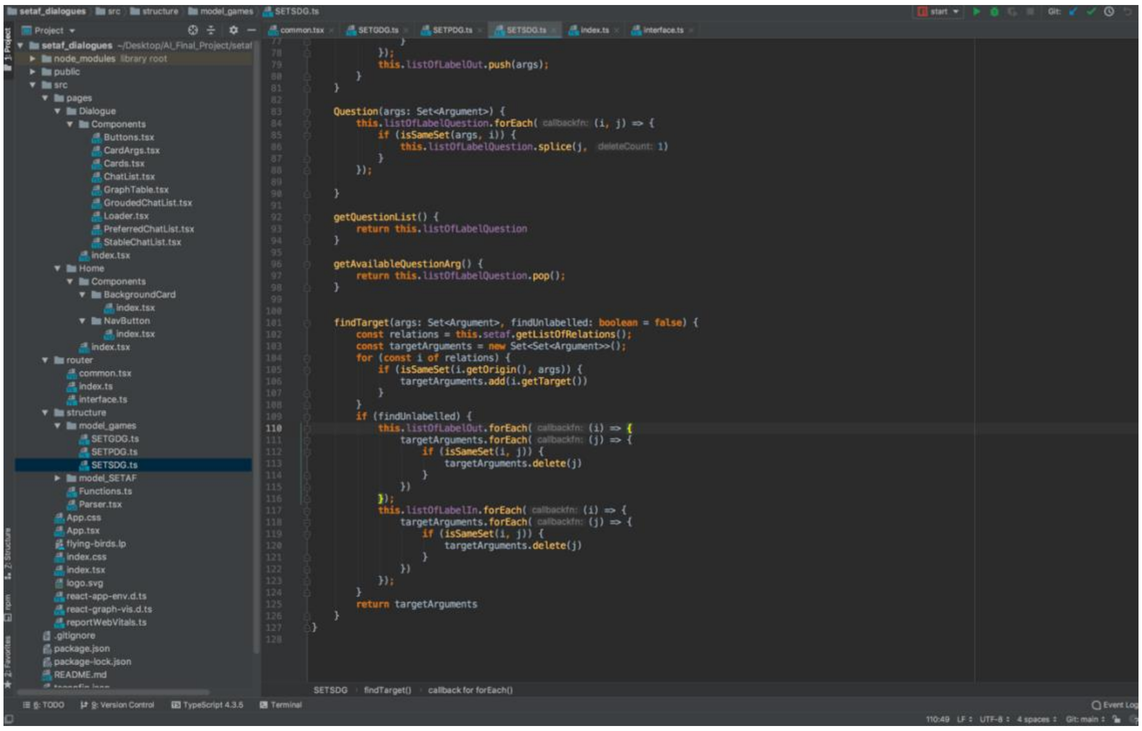
CODE SNIPPETS OF SETPDG



```typescript
import SETAF from "../model_SETAF/SETAF";
import {Argument} from "../model_SETAF/Argument";
import {isSameSet} from "../Functions";

export default class SETSDG {
    private setaf = new SETAF();

    private listOfLabelIn = new Array<Set<Argument>>();
    private listOfLabelOut = new Array<Set<Argument>>();
    private listOfLabelQuestion = new Array<Set<Argument>>();

    setSETAF(setaf: SETAF) {
        this.setaf = setaf;
        if (this.listOfLabelQuestion.length === 0 && this.listOfLabelIn.length === 0 && this.listOfLabelOut.length === 0) {
            this.setaf.getListOfArguments().forEach( callbackfn: i => {
                this.listOfLabelQuestion.push(i)
            });

            this.setaf.getListOfRelations().forEach( callbackfn: i => {
                if (i.getOrigin().size >= 2) {
                    this.listOfLabelQuestion.push(i.getOrigin())
                }
            });
        }
    }

    isWonByPro(args: Set<Argument>) {
        let flag = false;
        this.listOfLabelOut.forEach( callbackfn: i => {
            if (isSameSet(i, args)) {
                flag = true
            }
        });
        return flag
    }

    reset() {
        this.listOfLabelIn = new Array<Set<Argument>>();
        this.listOfLabelOut = new Array<Set<Argument>>();
        this.listOfLabelQuestion = new Array<Set<Argument>>();
    }

    isLabelled(args: Set<Argument>) {
        let flag = false;
        this.listOfLabelIn.forEach( callbackfn: (i) => {
            if (isSameSet(i, args)) {
                flag = true
            }
        });
        this.listOfLabelOut.forEach( callbackfn: (i) => {
            if (isSameSet(i, args)) {
                flag = true
            }
        });
```



```typescript
    isLabelled(args: Set<Argument>) {
        let flag = false;
        this.listOfLabelIn.forEach( callbackfn: (i) => {
            if (isSameSet(i, args)) {
                flag = true
            }
        });
        this.listOfLabelOut.forEach( callbackfn: (i) => {
            if (isSameSet(i, args)) {
                flag = true
            }
        });

        return flag
    }

    In(args: Set<Argument>) {
        if (!this.isLabelled(args)) {
            this.listOfLabelQuestion.forEach( callbackfn: (i, j) => {
                if (isSameSet(args, i)) {
                    this.listOfLabelQuestion.splice(j,  deleteCount: 1)
                }
            });


            this.listOfLabelIn.push(args);
        }
    }

    Out(args: Set<Argument>) {
        if (!this.isLabelled(args)) {
            this.listOfLabelQuestion.forEach( callbackfn: (i, j) => {
                if (isSameSet(args, i)) {
                    this.listOfLabelQuestion.splice(j,  deleteCount: 1)
                }
            });
            this.listOfLabelOut.push(args);
        }
    }

    Question(args: Set<Argument>) {
        this.listOfLabelQuestion.forEach( callbackfn: (i, j) => {
            if (isSameSet(args, i)) {
                this.listOfLabelQuestion.splice(j,  deleteCount: 1)
            }
        });

    }

    getQuestionList() {
        return this.listOfLabelQuestion
    }

    getAvailableQuestionArg() {
```



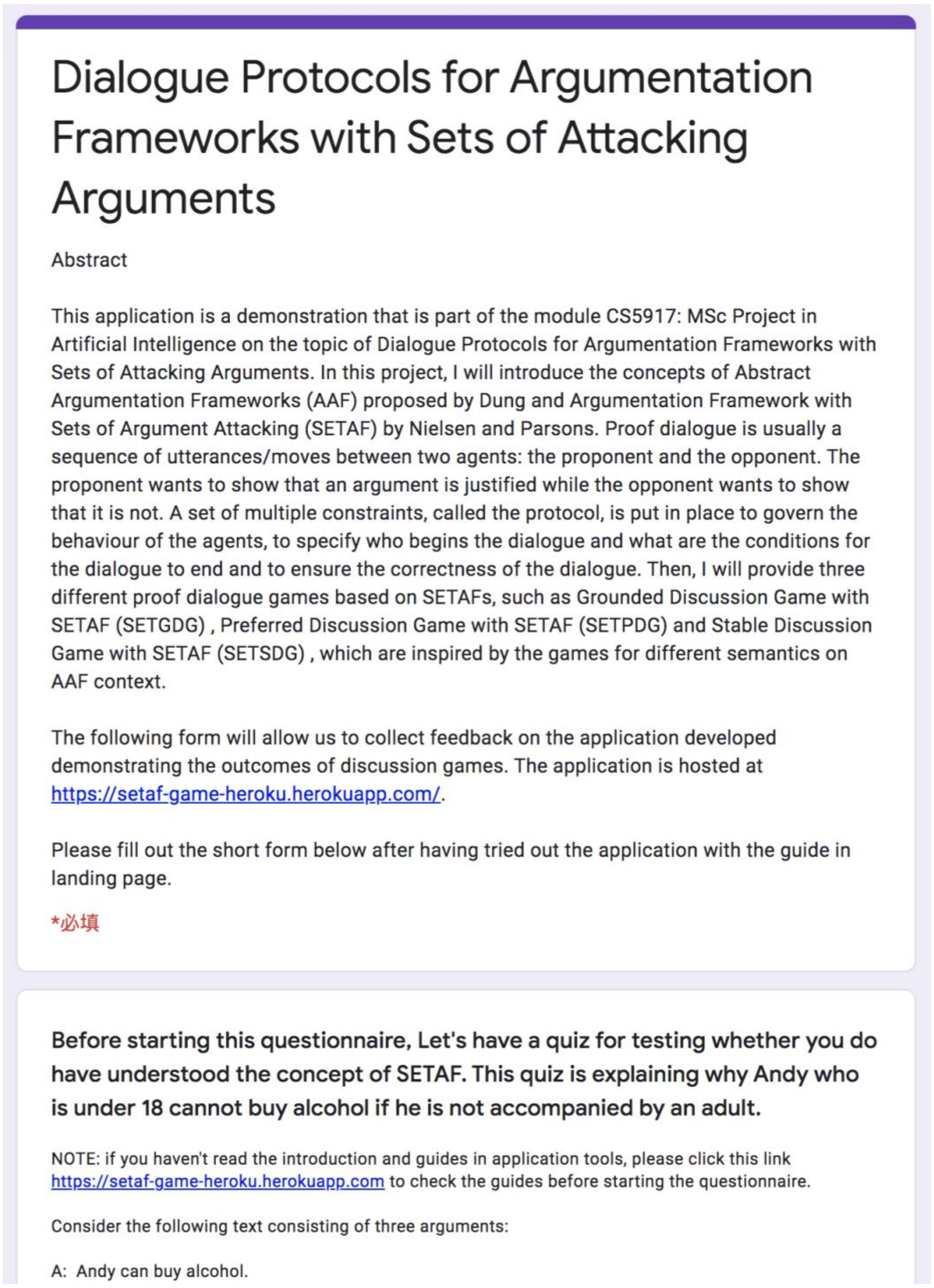
```
                }
            });
            this.listOfLabelOut.push(args);
        }
    }

    Question(args: Set<Argument>) {
        this.listOfLabelQuestion.forEach( callbackfn: (i, j) => {
            if (isSameSet(args, i)) {
                this.listOfLabelQuestion.splice(j,  deleteCount: 1)
            }
        });

    }

    getQuestionList() {
        return this.listOfLabelQuestion
    }

    getAvailableQuestionArg() {
        return this.listOfLabelQuestion.pop();
    }


    findTarget(args: Set<Argument>, findUnlabelled: boolean = false) {
        const relations = this.setaf.getListOfRelations();
        const targetArguments = new Set<Set<Argument>>();
        for (const i of relations) {
            if (isSameSet(i.getOrigin(), args)) {
                targetArguments.add(i.getTarget())
            }
        }
        if (findUnlabelled) {
            this.listOfLabelOut.forEach( callbackfn: (i) => {
                targetArguments.forEach( callbackfn: (j) => {
                    if (isSameSet(i, j)) {
                        targetArguments.delete(j)
                    }
                })
            });
            this.listOfLabelIn.forEach( callbackfn: (i) => {
                targetArguments.forEach( callbackfn: (j) => {
                    if (isSameSet(i, j)) {
                        targetArguments.delete(j)
                    }
                })
            });
        }
        return targetArguments
    }
}
```

Here below is the detailed information for SETSDG model.

- Properties:
  - listOfLabelIn: is an argument set storing the arguments labelled as IN.
  - listOfLabelOut: is an argument set storing the arguments labelled as OUT.
  - listOfLabelQuestion: is an argument set storing the arguments labelled as QUESTION.
- Functions:
  - setSETAF: is a function to set SETAF within this model.
  - isWonByPro: returns a boolean that determines whether this game terminated and win by PRO.
  - reset: is a function to reset three label collections.
  - isLabelled: returns a boolean that determines whether an argument has been labelled IN or OUT.
  - In: is the function of IN moves. It adds an argument into IN label set and delete it from QUESTION label set.
  - Out: is the function of IN moves. It adds an argument into OUT label set and delete it from QUESTION label set.
  - Question: is the function of QUESTION moves. It deletes an argument from QUESTION label set.
  - getQuestionList: returns the set of arguments labelled as QUESTION.
  - getAvailableQuestionArg: returns the last Argument in QUESTION label set and deletes it from the set.

- findTarget: returns the set of one Argument's target arguments.

Appendix 3 Questionnaire for Users



Dialogue Protocols for Argumentation Frameworks with Sets of Attacking Arguments

Abstract

This application is a demonstration that is part of the module CS5917: MSc Project in Artificial Intelligence on the topic of Dialogue Protocols for Argumentation Frameworks with Sets of Attacking Arguments. In this project, I will introduce the concepts of Abstract Argumentation Frameworks (AAF) proposed by Dung and Argumentation Framework with Sets of Argument Attacking (SETAF) by Nielsen and Parsons. Proof dialogue is usually a sequence of utterances/moves between two agents: the proponent and the opponent. The proponent wants to show that an argument is justified while the opponent wants to show that it is not. A set of multiple constraints, called the protocol, is put in place to govern the behaviour of the agents, to specify who begins the dialogue and what are the conditions for the dialogue to end and to ensure the correctness of the dialogue. Then, I will provide three different proof dialogue games based on SETAFs, such as Grounded Discussion Game with SETAF (SETGDG) , Preferred Discussion Game with SETAF (SETPDG) and Stable Discussion Game with SETAF (SETSDG) , which are inspired by the games for different semantics on AAF context.

The following form will allow us to collect feedback on the application developed demonstrating the outcomes of discussion games. The application is hosted at https://setaf-game-heroku.herokuapp.com/.

Please fill out the short form below after having tried out the application with the guide in landing page.

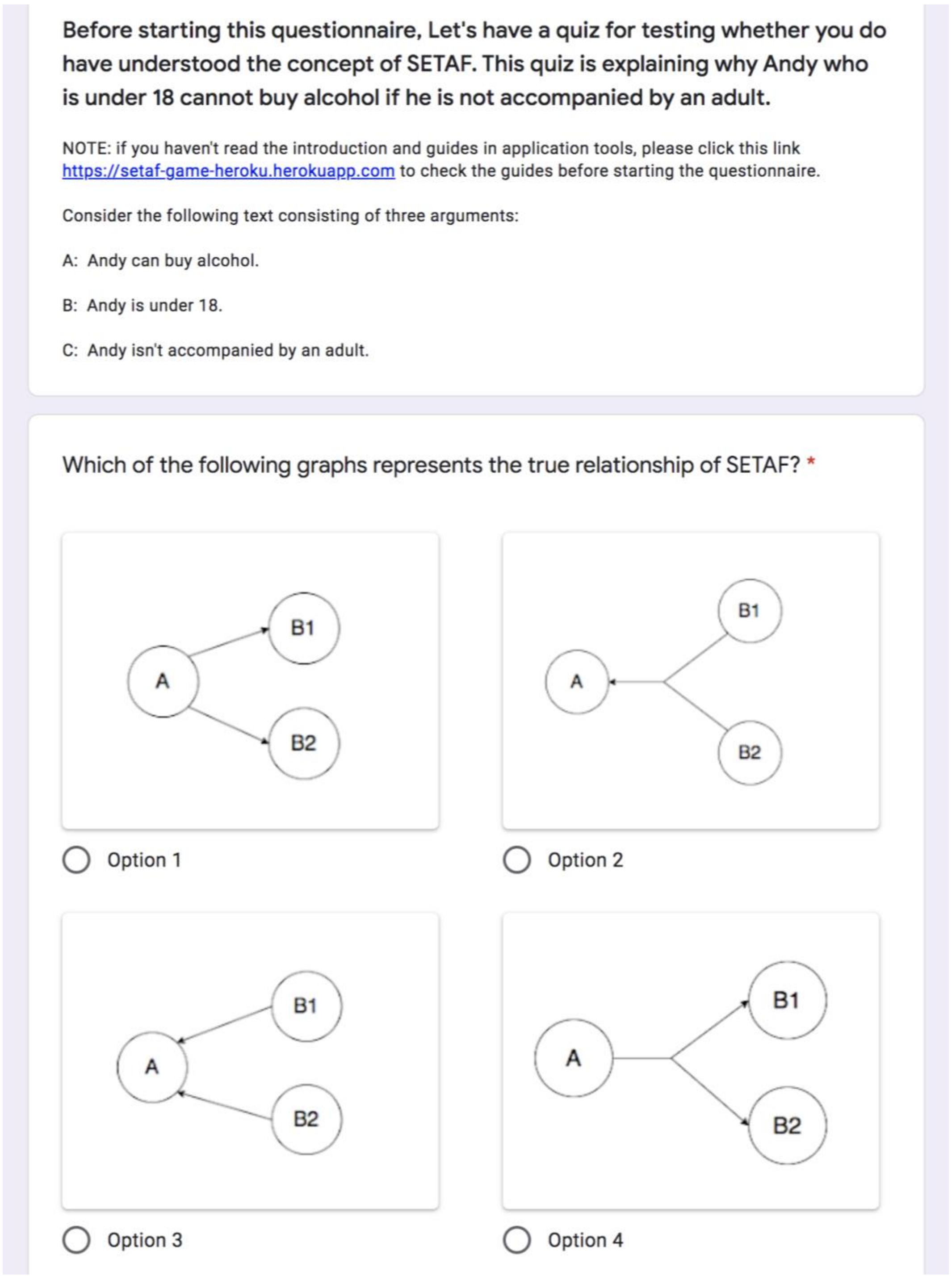
*必填

Before starting this questionnaire, Let's have a quiz for testing whether you do have understood the concept of SETAF. This quiz is explaining why Andy who is under 18 cannot buy alcohol if he is not accompanied by an adult.

NOTE: if you haven't read the introduction and guides in application tools, please click this link https://setaf-game-heroku.herokuapp.com to check the guides before starting the questionnaire.

Consider the following text consisting of three arguments:

A: Andy can buy alcohol.



Before starting this questionnaire, Let's have a quiz for testing whether you do have understood the concept of SETAF. This quiz is explaining why Andy who is under 18 cannot buy alcohol if he is not accompanied by an adult.

NOTE: if you haven't read the introduction and guides in application tools, please click this link https://setaf-game-heroku.herokuapp.com to check the guides before starting the questionnaire.

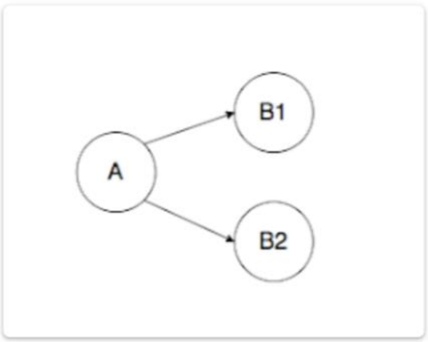
Consider the following text consisting of three arguments:

A: Andy can buy alcohol.

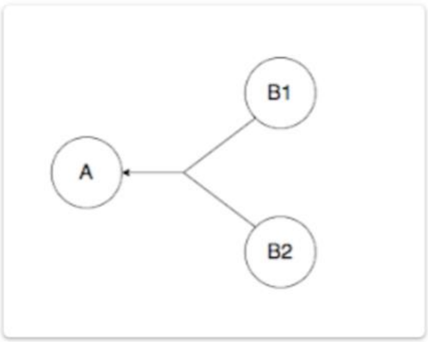
B: Andy is under 18.

C: Andy isn't accompanied by an adult.

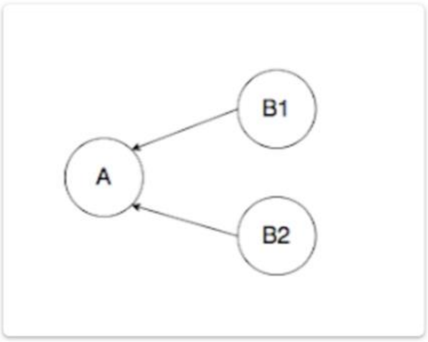
Which of the following graphs represents the true relationship of SETAF? *



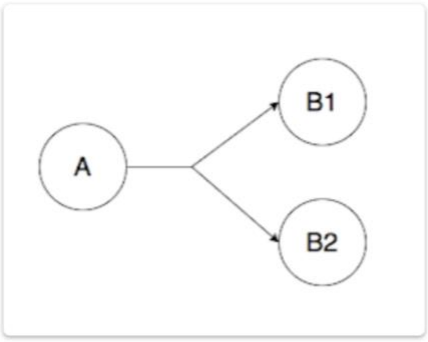
○ Option 1



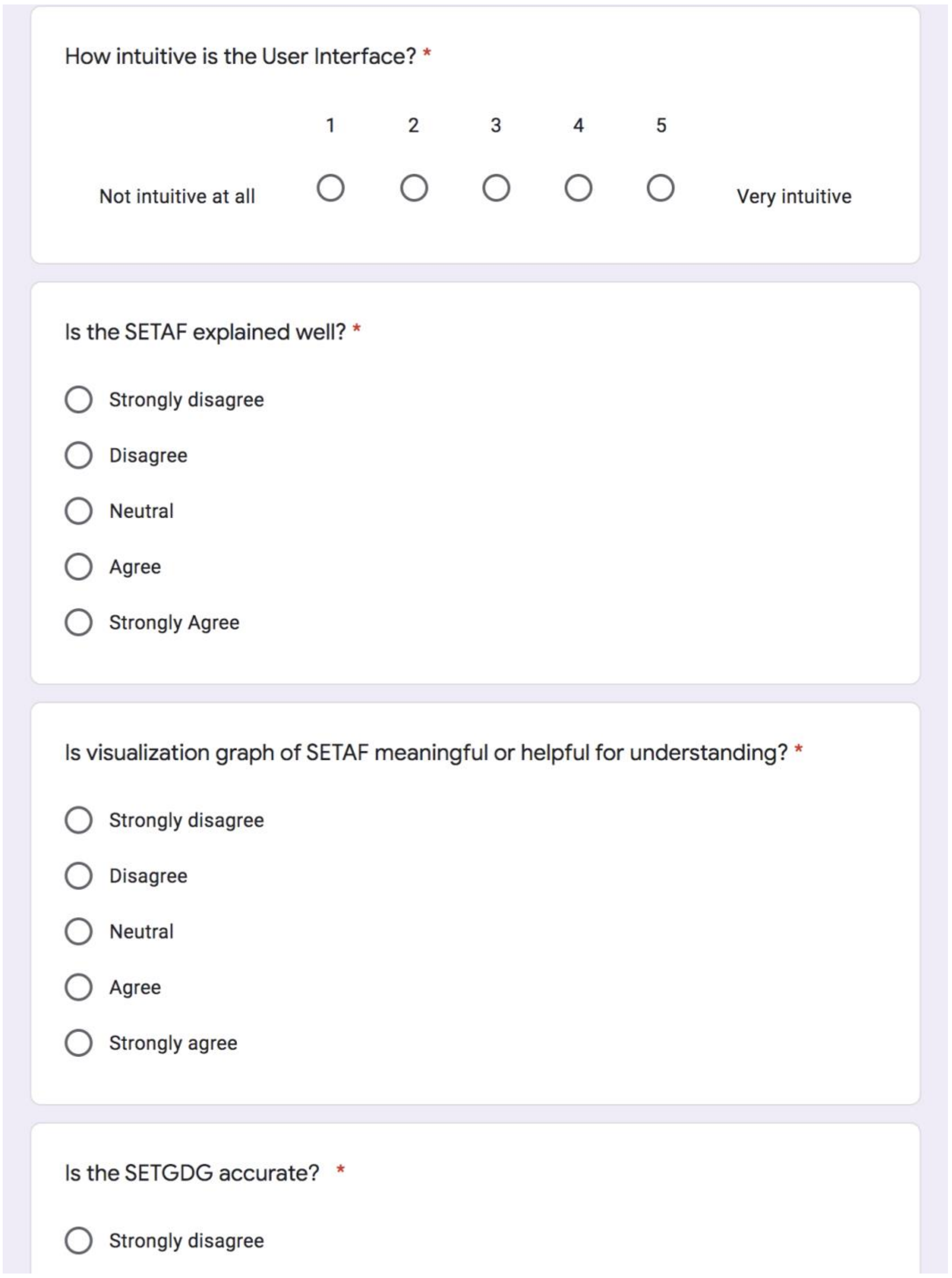
○ Option 2



○ Option 3



○ Option 4



How intuitive is the User Interface? *

	1	2	3	4	5	
Not intuitive at all	○	○	○	○	○	Very intuitive

Is the SETAF explained well? *

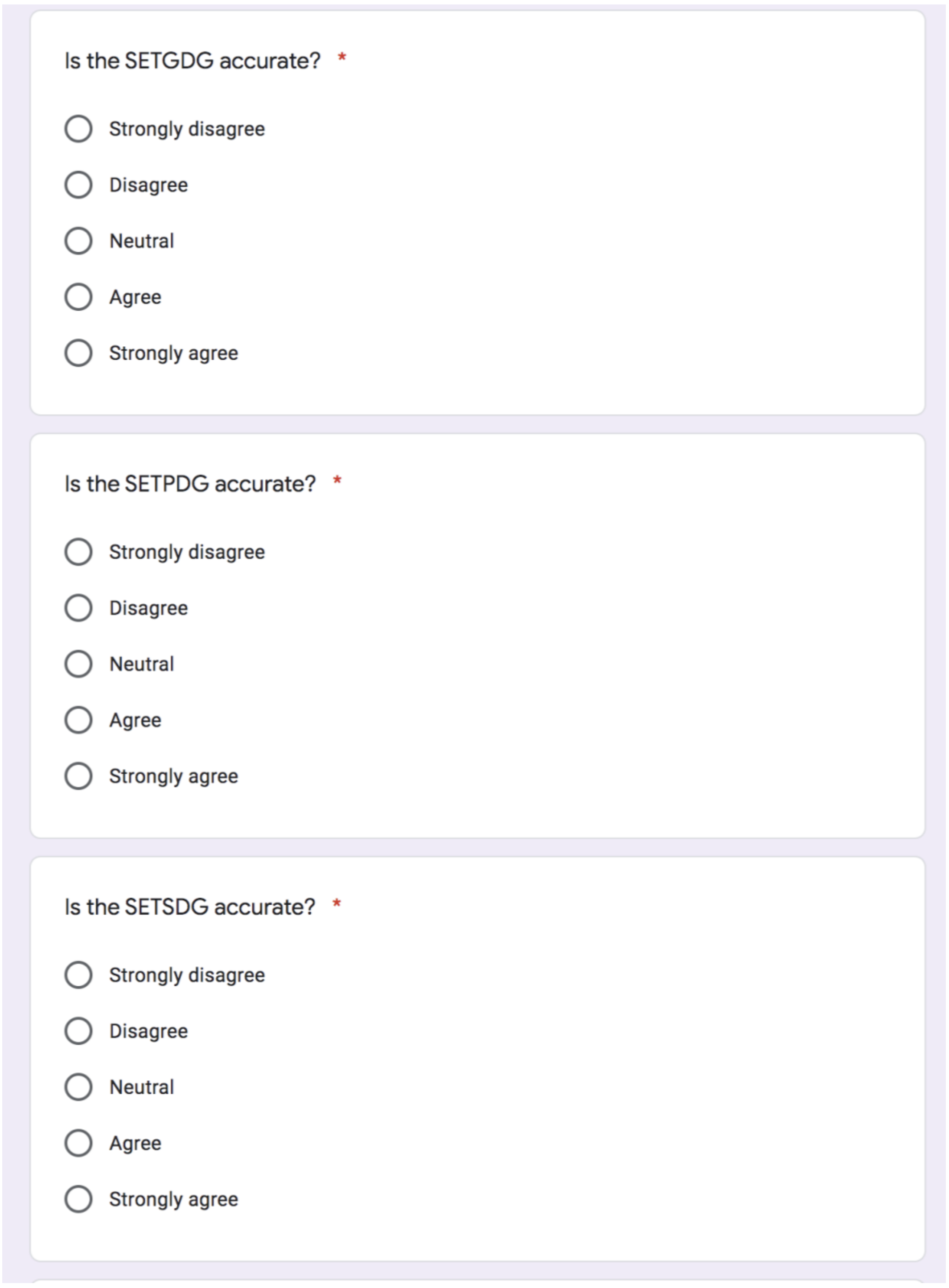
○ Strongly disagree
○ Disagree
○ Neutral
○ Agree
○ Strongly Agree

Is visualization graph of SETAF meaningful or helpful for understanding? *

○ Strongly disagree
○ Disagree
○ Neutral
○ Agree
○ Strongly agree

Is the SETGDG accurate? *

○ Strongly disagree



Is the SETGDG accurate? *

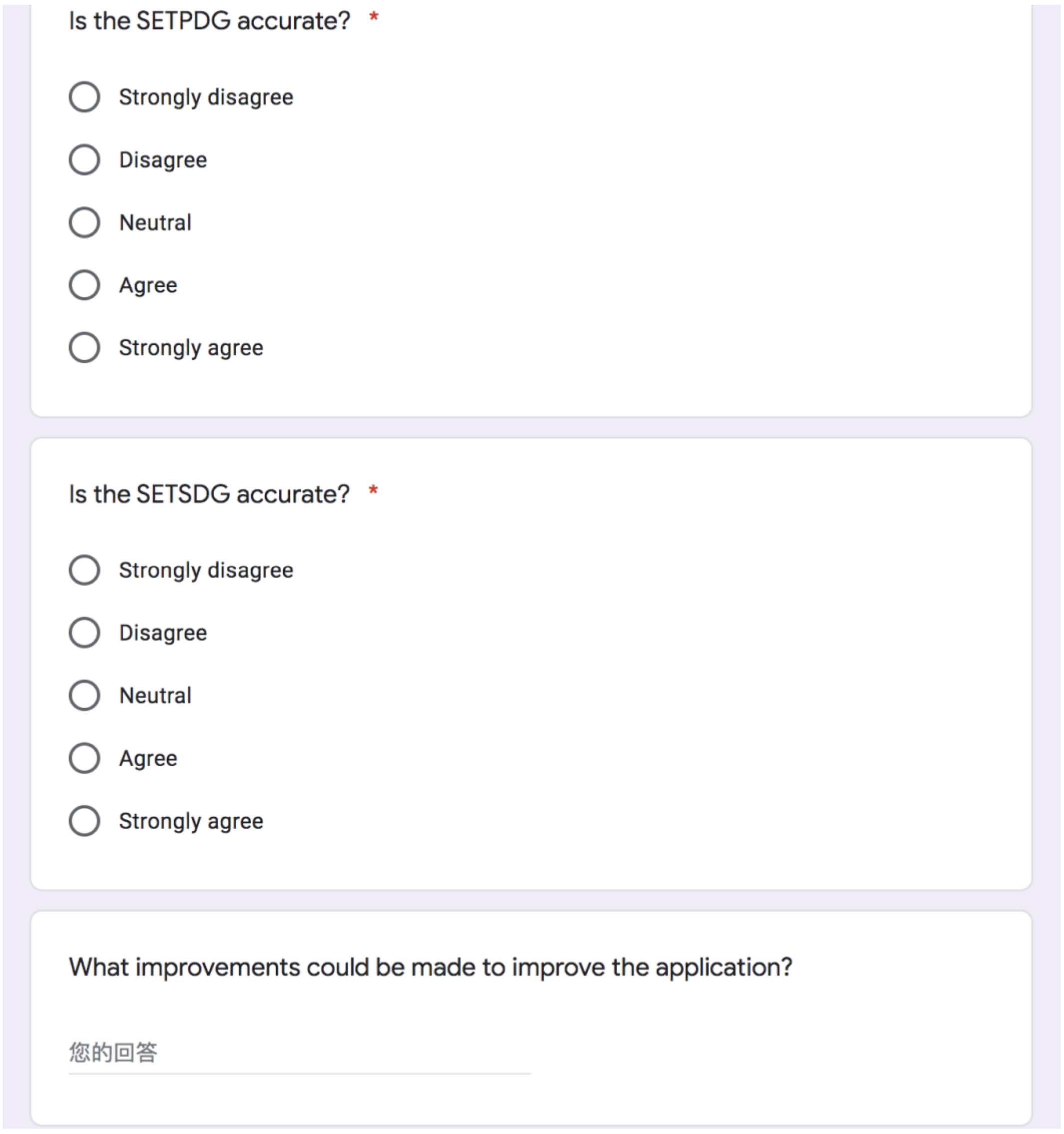
- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree

Is the SETPDG accurate? *

- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree

Is the SETSDG accurate? *

- ◯ Strongly disagree
- ◯ Disagree
- ◯ Neutral
- ◯ Agree
- ◯ Strongly agree



Is the SETPDG accurate? *

○ Strongly disagree
○ Disagree
○ Neutral
○ Agree
○ Strongly agree

Is the SETSDG accurate? *

○ Strongly disagree
○ Disagree
○ Neutral
○ Agree
○ Strongly agree

What improvements could be made to improve the application?

您的回答