AILP (2017) Report

Yijie Chen
S1508767

1st December 2017

1 Introduction

This report describes work done in the AILP course. It gives the aims and hypothesis that guided the work; describes the algorithms that were implemented; reports the results of experiments that were run; and analyses these results.

2 Aims and hypothesis

The aim of the assignment is to understand how the CAES system works, how the burden of proof and critical questions are used in this system and how it is used to overcome disruption on legal level or personal decision, through the evidences from plaintiff and defendant.

The intention of the extension implemented is:

to allow analysis of legal disputations in a way that closely matches what happens in real cases.

It assumes that evidences and arguments which plaintiff and defendant give shows in the input file (example.txt), and the questions they ask will be chosen from cq.txt. Through the argument and evidence in input file, the system can solve the legal disputation. The system will give the result of these arguments and evidence. It will give a reasonable consequence from these arguments rather than the truth. This system is used for :

Court decision
Personal decision
Some reasonable choice

3 Algorithms and implementation

For this purpose, the following extension were carried out

1. Allow user to input their arguments, assumptions to the system. The system will read input from text files (example.txt, cq.txt) and store them into a global variable which they belong to. Also change assumption, main query and premises, exception, result in argument to predicate form.

2. Showing the burden of proof on plaintiff or defendant, giving the current argument they talk and acceptable of the main query (dialogue system)

3. Giving current critical question one side asks and the other side give the response

3.1 Reading from text files

At first, I separate input file through split. I split every line as an element in the list. Then check its type (assumption, proof standard, argument, main query or default standard) and put them into the global variable which they belong to. Also, if any line in the input file is illegal, the system will raise an error. For default standard, main query and assumption, I use a similar method to check, which split the input sentence by their type and colon. Because, these types of sentences are all start with 'type: '. If the sentence contains 'type: ', it will split into 2 parts rather than the same as the original sentence. Then I put the second part in list into a suitable global variable. For arguments, there are 2 colons in a sentence, I split the input sentence by colon. If it belongs to the argument, then the length of the list after splitting should be 3. Then, the system will check whether the third part is a number which is smaller than 1 and larger than 0. If a sentence satisfies all requirements above, it belongs to argument type. The system will put the second part of the list as value and first part as the key into a global dictionary. Besides, system also put the weights (third part) as value and first part as key into a global dictionary for later use. There is a function called predicates, this function change all the propliteral in argument, assumption into predicate form. First, it will check whether the propliteral contains constant(s). If it contains constant, it will regard constant as one variable. Then, check the propliteral is binary or unary. If it is binary, the system will return a form likes predicate (A, B). If it is unary, it will return a form likes predicate (A).

3.2 Implementation of Burden of Proof and dialogue system

I set an empty input argument set as the global variable, and set burden of proof as plaintiff at first. Every time an argument is used by one side, then the argument will be entered into this set, and print on terminal. This is the basic dialogue system. For this part, I set 6 new global variables (contains the input argument set above). Two of them are used to store argument two sides (plaintiff and defendant) said (called plaintiff list and defendant list). Two of them are recorded the status of acceptable of main query. One of them is stored that the result of a argument and its arg id (called result set). For the last one, I separate argument into 3 parts, then split 'If', 'then' and 'unless' and take the second part in the list as value of dictionary and its arg id as key of dictionary. I start at main query, searching whole values in result set, if main query is equal to the value in result set. I take the arg id of this value, and put argument which has this arg id into current argument set, also put into plaintiff list. Then, check the acceptable of main query, if the acceptable of main query change, the burden of proof shift to defendant. Next, print the acceptable of main query, the current argument, burden of proof. For this argument, I search the its premises in result set , if the value matches, then execute the same process as above until it can not find matches of value and premises. If this time the burden of proof changes to defendant, then defendant will check all the argument in plaintiff list. Defendant will take all exception and 'not' + result out, and give evidence (argument) to support it. And do the same process above till it can not find matches of value and premises. This recursive will finish until all the useful argument is used.

3.3 Implementation of Critical Question

I create a new input file cq.txt which contains all critical question schemas. When one side (plaintiff or defendant) gives an argument, the predicates of premises, result and exception of this

argument will match will the critical questions in cq.txt. If the critical question contains any predicates, then start to match the variable and constant. The variables in predicate form of propliteral (pred (A,B) or pred (A)) will match the constant (capital letter in critical question except the first capital letter). It follows the order and replace the constant in critical question with variable in propliteral. And put this critical question into a global variable (called current critical question). Cause in the dialogue system, when one side ask a question, the other side should answer it. I match the predicate in critical question with result set (used in 3.2), if the predicate matches, then I use the arg id as a key, to get the value of the whole argument. For the following recursive, it is the same as 3.2.

4 Experiments and results

4.1 Choice of experiments

I choose my first example as the test case. The first input file has 7 arguments and lots of assumptions, and it is a legal disputation, which is more suitable for this system. These arguments support both positive and negative result. For example, one of the argument is “If Tony killed Sam and on_purpose Tony then murderer Tony unless psychosis Tony or self_defence” , which supports the main query (murderer Tony). And there is another one is “If witness2 said "Jack killed Sam" then not murderer Tony”, which supports the negate of main query (not murderer Tony). Besides, some arguments also contain exception. These things can easy to test all functions of my code (both for extension 2 and extension 3). Moreover, if reader works well, all the assumptions and arguments will be separated to a suitable global variable.

All arguments are:

```
[killed(Tony,Sam), on_purpose(Tony)], ~[psychosis(Tony), self_defence] => murderer(Tony)
[has_evidence(Tony), reliable(witness1), said(witness1,"killed(Tony,Sam)")], ~[] => killed(Tony,Sam)
[detective(Jacky), find(Jacky,evidence_Tony)], ~[] => has_evidence(Tony)
[said(witness2,"killed(Jack,Sam)")], ~[] => -murderer(Tony)
[-has_evidence(Jack), said(witness2,"killed(Jack,Sam)"), unreliable(witness2)], ~[] => -killed(Tony,Sam)
[detective(Jacky), find(Jacky,no_evidence_Jack)], ~[] => -has_evidence(Jack)
[said(witness3,"has(Tony,alibi)"), unreliable(witness3)], ~[] => -has_evidence(Tony)
```

And I put all the critical question scheme in one input file (cq.txt). It contains:

```
Is self_defence ?
Is T has_evidence ?
Is P unsound_mind ?
Does T killed S ?
Does T waste W ?
Does P agrees E ?
Is T has_enough_money ?
Is T has_enough_time ?
Does E contains A ?
Is G not_care ?
Does A has B ?
Is A not_need ?
```

All critical question will search from this file.
It is more flexible that put all the critical question scheme in one file rather than put them in every example files. Because, if one of the premises in example1 and 2 use the same predicates, then the question can be used both in example1 and 2. It makes the system has a higher flexibility.

4.2 Experimental results

4.2.1 Experimental result of extension 1

```
the main query is ['murderer(Tony)']
------------------------------------------------------------
the arg_if_then is {'arg6': 'If witness2 said "Jack killed Sam" then not murderer Tony', 'arg3': 'If detective Jacky and Jacky find evidence_T
ony then has_evidence Tony', 'arg5': 'If Tony killed Sam and on_purpose Tony then murderer Tony unless psychosis Tony or self_defence', 'arg4'
: 'If witness3 said "Tony has alibi" and unreliable witness3 then not has_evidence Tony', 'arg1': 'If witness1 said "Tony killed Sam" and has_
evidence Tony and reliable witness1 then Tony killed Sam', 'arg7': 'If Jacky find no_evidence_Jack and detective Jacky then not has_evidence J
ack', 'arg2': 'If witness2 said "Jack killed Sam" and not has_evidence Jack and unreliable witness2 then not Tony killed Sam'}
------------------------------------------------------------
the assumption is {said(Chen,"not_psychosis Tony"), on_purpose(Tony), reliable(witness1), find(Jacky,no_evidence_Jack), -self_defence, detecti
ve(Jacky), unreliable(witness2), said(witness3,"Tony has alibi"), relative(witness4), said(witness1,"Tony killed Sam"), said(witness2,"Jack ki
lled Sam"), unreliable(witness3), Doctor(Chen), find(Jacky,evidence Tony), find(witness3,evidence Jack)}
```

From the output in terminal, it's easy to find that all arguments and assumptions are separated to the suitable set.

4.2.2 Experimental result of extension 2

```
Current argument:
Main query acceptable: False
Burden of proof: Plaintiff
---------------------------------------
Current argument:
[killed(Tony,Sam), on_purpose(Tony)], ~[psychosis(Tony), self_defence] => murderer(Tony)
Main query acceptable: False
Burden of proof: Plaintiff
---------------------------------------
Current argument:
[killed(Tony,Sam), on_purpose(Tony)], ~[psychosis(Tony), self_defence] => murderer(Tony)
[has_evidence(Tony), reliable(witness1), said(witness1,"killed(Tony,Sam)")], ~[] => killed(Tony,Sam)
Main query acceptable: False
Burden of proof: Plaintiff
---------------------------------------
Current argument:
[killed(Tony,Sam), on_purpose(Tony)], ~[psychosis(Tony), self_defence] => murderer(Tony)
[has_evidence(Tony), reliable(witness1), said(witness1,"killed(Tony,Sam)")], ~[] => killed(Tony,Sam)
[detective(Jacky), find(Jacky,evidence_Tony)], ~[] => has_evidence(Tony)
Main query acceptable: True
Burden of proof: Defendant
---------------------------------------
Current argument:
[killed(Tony,Sam), on_purpose(Tony)], ~[psychosis(Tony), self_defence] => murderer(Tony)
[has_evidence(Tony), reliable(witness1), said(witness1,"killed(Tony,Sam)")], ~[] => killed(Tony,Sam)
[detective(Jacky), find(Jacky,evidence_Tony)], ~[] => has_evidence(Tony)
[said(witness2,"killed(Jack,Sam)")], ~[] => -murderer(Tony)
Main query acceptable: True
Burden of proof: Defendant
---------------------------------------
```

In this experiment, I check whether the burden of proof works well or not. At first, the burden of proof is on plaintiff and the main query is unacceptable. Plaintiff gives an argument to support main query (murderer(Tony)), and then it gives evidences to support premises until there are no more arguments which support premises. At this time, the burden of proof changes to defendant, cause the acceptable of main query change to True. Defendant starts to rebut. Giving an argument supports negate of murderer(Tony). Thus, this experiment shows that all the functions in this extension works well. The dialogue system also establishes.

4.2.3 Experimental result of extension 3

In this experiment, I'd like to test the implementation of critical question. The first 2 dialogue are same as the extension 2, but in the third dialogue, the argument matches a scheme "Does T killed S?". In this argument, T is Tony and S is Sam, and the system replace the constant with variable successfully. The current critical question will be Does Tony killed Sam. Besides, when defendant asks this question. Plaintiff should answer it, so plaintiff gives an argument to show "Tony killed Sam". Also plaintiff should give enough evidence to support this argument until premises are assumptions. In the recursive, defendant also can ask question like "Is Tony has_evidence", and

plaintiff will show it. These two questions are good critical questions, but the last one “Is self_defence?” Is a bad critical question, because self_defence is in assumption. This critical question is meaningless. So this experiment shows the critical question works successfully.

```
Current argument:
Main query acceptable: False
Burden of proof: Plaintiff
Current critical question:
--------------------------------------
Current argument:
[killed(Tony,Sam), on_purpose(Tony)], ~[psychosis(Tony), self_defence] => murderer(Tony)
Main query acceptable: False
Burden of proof: Plaintiff
Current critical question:
--------------------------------------
Current argument:
[killed(Tony,Sam), on_purpose(Tony)], ~[psychosis(Tony), self_defence] => murderer(Tony)
[has_evidence(Tony), reliable(witness1), said(witness1,"Tony killed Sam")], ~[] => killed(Tony,Sam)
Main query acceptable: False
Burden of proof: Plaintiff
Current critical question: Does Tony killed Sam ?
--------------------------------------
Current argument:
[killed(Tony,Sam), on_purpose(Tony)], ~[psychosis(Tony), self_defence] => murderer(Tony)
[has_evidence(Tony), reliable(witness1), said(witness1,"Tony killed Sam")], ~[] => killed(Tony,Sam)
[detective(Jacky), find(Jacky,evidence_Tony)], ~[] => has_evidence(Tony)
Main query acceptable: True
Burden of proof: Defendant
Current critical question: Is Tony has_evidence ?
--------------------------------------
Current argument:
[killed(Tony,Sam), on_purpose(Tony)], ~[psychosis(Tony), self_defence] => murderer(Tony)
[has_evidence(Tony), reliable(witness1), said(witness1,"Tony killed Sam")], ~[] => killed(Tony,Sam)
[detective(Jacky), find(Jacky,evidence_Tony)], ~[] => has_evidence(Tony)
[said(witness2,"Jack killed Sam")], ~[] => -murderer(Tony)
Main query acceptable: True
Burden of proof: Defendant
Current critical question: Is self_defence ?
```

5 Discussion and Conclusion

This system is quite useful for making a decision in some part of daily decision or court decision. If the decision is definitely True or False, this system is really suitable to judge, but if the decision is ambiguous or the answer may be undecided between True or False, this system is not suitable. This system always can give an answer, as long as the input file is correct, but whether the result is the truth or not, this system can not guarantee. But we can improve this by output the percentage confidence on True and False. Then, the ambiguous problem can be solved.Besides, when it solves some real life problems, the weight is not actually existed. The weight is given by the operator of this system, which is really subjective. Thus, an improvement can be made to solve this issue. We can get weight through machine learning. We can set a big database to store all the arguments with their weight, and when we have a new argument, then we search from database and find similar arguments, using these data and machine learning to get a relative objective number. Also, the critical question in this system is inflexible. In real life, the questions which can be asked are always flexible, people ask what they want to know. So maybe a linguistic system should be import to improve this system.