Formalizing a Switch of Burden of Proof by Logic Programming

Ken Satoh[1], Satoshi Tojo[2], Yoshitaka Suzuki[2]

[1] National Institute of Informatics and Sokendai
ksatoh@nii.ac.jp
[2] Japan Advanced Institute of Science and Technology
{tojo, syoshita}@jaist.ac.jp

Abstract. Prakken [2] argues that we cannot formalize a switch of burden of proof in legal reasoning either by argumentation framework or by nonmonotonic reasoning since argumentation semantics can represent any nonmonotonic formalism. In this paper, we argue that a switch of burden of proof can be formalized in nonmonotonic reasoning by formalizing burden of proof in a different way followed by the Japanese civil procedure law.

1 Introduction

In this paper, we consider a switch of burden of proof(SBP)[3] . Prakken [2] claims that we cannot formalize SBP in argumentation semantics such as [1]. He also argues that nonmonotonic reasoning does not formalize SBP since argumentation semantics can represent any nonmonotonic formalism. In this paper, we argue that SBP can be formalized in nonmonotonic reasoning by formalizing a burden of proof in a different way from Prakken's formalization. We formalize a burden of proof as a decision making used by a judge whereas the argumentation semantics formalizes a burden of proof as a dialogue game between two sides which adversarially attacks each other.

In the Japanese civil procedure law, a burden of proof is a tool of decision making by a judge when an ultimate fact is "non liquet", that is, the state which we cannot conclude that the ultimate fact is true or not even after giving all available pieces of evidence [6]. This burden is decided a priori in a general way and there is no influence of context. In this paper, the status of "non liquet" is formalized as the non-existence of any information about the ultimate fact or

[3] Prakken uses a terminology "a shift of the burden of proof in [2], but we use "a switch" in this paper to avoid confusion since "a shift of the burden of proof" is sometimes used for a change of the responsibility on who proves a certain fact in a certain context. For example, in Japan in tort dispute, plaintiff has a burden of proving the existence of causality of defendant's act and damage whereas in traffic accident dispute which is a subclass of tort, defendant has a burden of proving the non-existence of causality when some damage occurs. In Japan, we call it *a shift of burden of proof*.

the co-existence of plausible information which contradicts each other. The latter case often arises when each side provides a contradicting evidence each of which will lead to the different truth-value. When an ultimate fact is "non liquet", the default value of the fact is decided and as a side-effect, if some default value is beneficiary to one side, then the other side must prove the negation of the default value to prevent the status of "non liquet".

We give an example of the burden of proof which is used in [2].

- Suppose that plaintiff claims that a contract between him and defendant exists because plaintiff offered defendant to sell his car, and defendant accepted.
- Plaintiff has the burden of proving that there was such an offer and acceptance.

This means that for each of ultimate facts (offer and acceptance), if there is no evidence for the fact or there are contradictory pieces of evidence for the fact (in other words, in "non liquet" status) at the final decision of judgement, we assume that the truth value of the fact is false and therefore, plaintiff will lose this case. Therefore plaintiff needs to strictly prove offer and acceptance by giving a piece of evidence of offer and acceptance and rebutting a contradictory evidence given by defendant.

Then, a switch of burden of a proof occurs as follows.

- Suppose that defendant concedes plaintiff's claims concerning offer and acceptance, and instead attacks plaintiff's argument by claiming an exception, viz. that she was insane when she accepted plaintiff's offer.
- In this case, a defendant has the burden of proving insanity. The judge must be convinced of her insanity.

Therefore, in this situation an allocation of proof is switched to defendant and as far as the issue of defendant's insanity is concerned, if there is no evidence from defendant or there are contradictory pieces of evidence for insanity, then the truth value of insanity becomes false and defendant will lose this case.

Prakken claims that in existing argumentation systems, we cannot formalize a switch of burden of proof since this switch is not automatically decided in such a way that burden of proof is switched alternately. Moreover, Prakken claims that nonmonotonic reasoning cannot formalize a switch of burden of proof since the argumentation system can represent any nonmonotonic reasoning. Then, To solve the problem, he introduces a function which allocates a burden of proof for each ultimate fact which is outside the argument system.

We believe that the cause of Prakken's claim is that he understands burden of proof in a dialogue game. Therefore, if we understand burden of proof differently, there would be possibility to formalize burden of proof in nonmonotonic reasoning. In this paper, we understand burden of proof as a decision making for the judge. However, there is a problem of this kind since we have to let adversarial arguments co-exist because judge should decide a conclusion after seeing all the arguments. To solve the problem, we introduce a `p`-predicate which means

that there is an evidence of ultimate facts. This is an essential difference of our formalism from Prakken's formalism since in his formalism, these adversarial arguments cannot co-exist and therefore, cannot represent "non liquet" status. By allowing such co-existence of contradictory argument, we can show that not only burden of proof but also a switch of burden of proof can be formalized.

2 Logic Programming Formalization of Burden of Proof

Firstly, we introduce a predicate `p` in order to reify plausibility of `P` and the meaning of `p(P)` for an ultimate fact in a law is that there is a piece of evidence that a fact `P` is plausible. To simplify our discussion, we assume that a piece of evidence can be either just a statement of `p(P)` or some derived result from subsidiary rules which express a relation between an ultimate fact and evidentiary facts.

A proof of an ultimate fact is expressed using a predicate `proved` as follows:

```
proved(P) :- p(P), not p(-P).
```

where

- `-P` means a explicit negation of `P` as used in answer set programming.
- `not` represents "negation as failure" which means the failure of showing evidence.

Then, using the same terminology, we can represent "non liquet" status as "`not p(P),not p(-P)`" or "`p(P), p(-P)`" where

- The first disjunct means that there is no evidence for a fact `P`.
- The second disjunct means that there are contradictory pieces of evidence for a fact `P`.

Suppose that the default value of `P` is true in "non liquet" status. Then, this fact becomes true either by proving `P` or by making `P` "non liquet" status. To distinguish from a fact which requires a strict proof and a fact which can use the default value, we introduce another predicate `default` to express this situation. Therefore, the rules for giving the default truth value of `P` are the following three:

```
default(P) :- not p(P), not p(-P).
default(P) :- p(P), p(-P).
default(P) :- p(P), not p(-P).
```

We can simplify these rules into the following:

```
default(P) :- p(P).
default(P) :- not p(-P).
```

since

– The first rule of the simplified rule set, `default(P) :- p(P).`, is equivalent to the combination of `default(P) :- p(P), p(-P).` and `default(P) :- p(P), not p(-P).` in the original rule set since `default(P) :- p(P), p(-P). default(P) :- p(P), not p(-P).` is equivalent to `default(P) :- (p(P), p(-P)) or (p(P), not p(-P)).` and "`p(P), p(-P)` or `p(P), not p(-P)`" is equivalent to `p(P)`.
– Similarly, the second rule of the simplified rule set, `default(P) :- not p(-P).`, is equivalent to the combination of `default(P) :- not p(P), not p(-P).` and `default(P) :- p(P), not p(-P).` in the original rule set since `default(P) :- not p(P), not p default(P) :- p(P), not p(-P).` is equivalent to `default(P) :- (not p(P), not p(-P)) or (p(` and "`not p(P), not p(-P)` or `p(P), not p(-P)`" is equivalent to `not p(-P)`.

We can furtherly simplify the above two rules into one rule by using the predicate `proved` as follows:

```
default(P) :- not proved(-P).
```

since the definition of `proved(-P)` is "`p(-P), not p(P)`", by the negating the definition, we obtain "`not p(-P)` or `p(P)`" which is equivalent to the previous two rules. This means that if this default value is beneficiary to one side, the other side must prove the negation of the default value strictly. In this case, we can say that the burden of proof of proving the negation of `P` is allocated to the opponent.

Note that if a side cannot succeed in showing `proved(P)`, the value of `P` is assumed to be false. Therefore, if we write `proved(P)`, the burden of proof of proving `P` is allocated to the same side. This means that the burden of proof of every ultimate fact is allocated to either side.

In a general case, we make the following rule. For any conclusion, we divide ultimate facts used to prove conclusions into the two categories; one of them should be proved by the proponent (we represent as `P_1,...,P_n`) and the negation of the other of them should be proved by the opponent (we present these conditions as `-O_1,...,-O_m`) to rebut the conclusion. Then, we have the following rule:

```
p(C) :-
     proved(P_1), proved(P_2),..., proved(P_n),
     default(O_1),...,default(O_m).
```

3 Reasoning Examples

Now, we show examples that the above formulation really works. In order to show the power of our formulation, we slightly modify the example of "offer and acceptance" used in Section 1 where we introduce more ultimate facts that "`document_insane`", "`authentic_document`" and "`correct_stamp`" meaning respectively that the court issues a document which declares that defendant was insane at the time of contract, that the document is authentic and that the document has a correct stamp. These facts are also used in [2] as follows.

– To prove defendant's insanity, the defendant must prove that there is an official-looking document containing a judicial decision declaring her insane.
– In order to claim that this document is not authentic by proving that the correct stamp is not put on the document, the burden of its proof is on the one who claims this, so a switch of burden of proof occurs again.

Considering all the rules above, we have the following logic program. Note that the burden of proof is already decided in advance as stated in Section 1, therefore we can write the following rules beforehand. Then, according to the context of available information, we add some contingent facts to make a decision.

```
p(valid_contract):-proved(offer),proved(acceptance),default(sane).

p(-sane):-proved(document_insane),default(authentic_document).

p(-authentic_document):-proved(-correct_stamp).

proved(X):-p(X), inverse(X,InX), not p(InX).

default(X):-inverse(X,InX), not proved(InX).

inverse(-X,X).
inverse(X,-X):-atomic(X).
```

Note that we introduce the predicate `inverse` since we will consider both of positive atom and negated atom.

– The first rule,
"`p(valid_contract) :- proved(offer),proved(acceptance),default(sane).`"
means that in order to prove a piece of evidence strictly that there is a valid contract, we have to show that there is an offer and acceptance and a person in the other party is sane. However, since the default value of sanity is true (or in other words, the burden of proving "insanity" is on the opponent), we do not need to prove strictly that the person is sane.
– The second rule,
"`p(-sane) :- proved(document_insane),default(authentic_document).`"
means that in order to show a piece of evidence that the person is not sane at the time of contract, we have to show that there is a document issued by a court that the person is not sane and the document is authentic. However, since the burden of proving authenticity of document is on the opponent, we do not need to show that the document is authentic.
– Finally, the third rule,
"`p(-authentic_document):-proved(-correct_stamp).`"
means that in order to show a piece of evidence that the document is not authentic, we have to strictly prove that there is no correct stamp from the court.

We give reasoning examples of judgement in various information contexts as follows:

Example 1. Suppose that plaintiff gave a piece of evidence of offer and acceptance and there were no other information available and we make a judgement in this context. This means that we add `p(offer)` and `p(acceptance)` into the program. Then, `proved(valid_contract)` becomes true since the default value of `sane` is true and there is no evidence from the defendant that `sane` is false.

Example 2. Suppose that plaintiff gave a piece of evidence of offer and acceptance and the defendant gave a piece of evidence of non-existence of the acceptance and we make a judgement in this context. This means that we add `p(offer)` and `p(acceptance)` and `p(-acceptance)` into the program. Then, `proved(valid_contract)` becomes false since `proved(acceptance)` is failed since there are contradictory pieces of evidence about `acceptance`.

Example 3. Suppose that the plaintiff gave a piece of evidence of offer and acceptance and the defendant gave a piece of evidence of the document which declares that the defendant was insane at the time of contract and we make a judgement in this context. This means that we add `p(offer)`, `p(acceptance)` and `p(document_insane)` into the above program. Then, `proved(valid_contract)` becomes false since the default value of `sane` is overwritten by the fact of `p(document_insane)`.

Example 4. Suppose that the plaintiff gave a piece of evidence of offer and acceptance and the defendant gave a piece of evidence of the document which declares that the defendant was insane at the time of contract and the plaintiff gave a piece of evidence that the document does not have correct stamp and we make a judgement in this context. This means that we add `p(offer)`, `p(acceptance)` and `p(document_insane)` and `p(-correct_stamp)` into the above program. Then, authenticity of the document is not satisfied since `default(document_insane)` is failed because `p(-document_insane)` is proved. Therefore, `sane` again becomes true and `proved(valid_contract)` becomes true.

4 Further Refinement of Formalization

In the above formalization, which side has the burden of proof is not so clear. We can express who has the burden of proof by adding arguments for each predicate as follows. In the following program, the second argument in the `p`-predicate expresses the side which has the burden of proof of the first argument and the third argument in `default` expresses the side which has the burden of proof of the negation of the first argument.

```
p(valid_contract,P,O) :-
    proved(offer,P,O),proved(acceptance,P,O),default(sane,P,O).
p(-sane,P,O):-
    proved(document_insane,P,O),default(authentic_document,P,O).
```

```
p(-authentic_document,P,O):-
    proved(-correct_stamp,P,O).

proved(X,P,O):-
    p(X,P,O),inverse(X,InX),not p(InX,O,P).
default(X,P,O):-
    inverse(X,InX),not proved(InX,O,P).

inverse(-X,X).
inverse(X,-X).
```

Then, we can extract information how the burden of proof is switched by adding some more extra commands for printing into the program. We show the program in Appendix. Here, we show the trace of calling `proved(valid_contract,plaintiff,defendant)` in Example 4 in Section 3.

```
proved(valid_contract,plaintiff,defendant).
  plaintiff is now proving "valid_contract" against defendant.
    plaintiff is now proving "offer" against defendant.
      plaintiff claims that there is a piece of evidence for "offer".
    plaintiff has proved "offer" against defendant.
    plaintiff is now proving "acceptance" against defendant.
      plaintiff claims that there is a piece of evidence for "acceptance".
    plaintiff has proved "acceptance" against defendant.
    *** A shift of a burden of proof occurs. ***
    In order to rebut "valid_contract",
      it is sufficient for defendant to prove "-sane".
    defendant is now proving "-sane" against plaintiff.
      defendant is now proving "document_insane" against plaintiff.
        defendant claims that there is a piece of evidence for "document_insane".
      defendant has proved "document_insane" against plaintiff.
      *** A shift of a burden of proof occurs. ***
      In order to rebut "-sane",
        it is sufficient for plaintiff to prove "-authentic_document".
      plaintiff is now proving "-authentic_document" against defendant.
        plaintiff is now proving "-correct_stamp" against defendant.
          plaintiff claims that there is a piece of evidence for "-correct_stamp".
        plaintiff has proved "-correct_stamp" against defendant.
      plaintiff has proved "-authentic_document" against defendant.
    defendant has failed in proving "-sane" against plaintiff.
  plaintiff has proved "valid_contract" against defendant.
```

5 Related Work

Sartor[5, Page 729] considers Prakken's criticism against logical models of defeasible reasoning in terms of burden of proof. Sartor says that the weak point

of defeasible reasoning is that when there is a balance between arguments for and against a fact, we are unable to build a justified argument for conclusion of C according to a rule using the fact in its condition.

We think that a similar weakness also exists in nonmonotonic reasoning if we cannot express the coexistence of the contradictory arguments. For example, consider the following logic program

```
a :- not b.
b :- not a.
c :- not a.
```

Our intuition here is that we would like to prove `c` when we fail in proving `a`. However, in the current semantics of logic programming, we cannot express such intension.

If we consider the well-founded semantics, we cannot prove `c`; the truth-value of `c` is undefined. If we consider the stable model semantics, although in one stable model, we can conclude `c` by assuming `a` is false, we have another model where we fail to conclude `c` by assuming `b` is false. Therefore in both models, we cannot conclude `c` decisively.

Therefore, what we could like to have a rule like:

```
a :- not b.
b :- not a.
c :- {if we do not decide the truth value of a}
```

Unfortunately, the condition of `{if we do not decide the truth value of a}` cannot be expressed in usual nonmonotonic reasoning formalism, therefore, Prakken's argument is somehow correct.

However, as shown in this paper, this failure can be mended by changing the formalization of burden of proof so that we can allow co-existence of contradictory arguments.

In [3, 4], Prakken and Sartor developed their work on burden of proof furtherly and they propose a formalization of distinguish two kinds of burden of proof; one is the *burden of production* and the other is the *burden of persuasion*. In burden of production, one side has a responsibility to raise the issue but the other side may have a burden of proof to negate the issue. In the current formalism, we only consider burden of persuasion and therefore, to formalize burden of production is one of important future work.

6 Conclusion

In this paper, we give a formalization of burden of proof and show that a switch of burden of proof can be expressed correctly in the nonmonotonic formalization without any additional functions which Prakken used in [2]. We believe that the reason why our attempt is successful is because we do not formalize burden of proof by dialogue game, but formalize it in terms of decision making by a judge after all the evidence was provided as Japanese civil procedure suggests. As future work, we would like to consider the following:

– According to our formalism, we could infer a lack of evidence if we cannot obtain the desired result in the current context. In a naive way, we just add various pieces of evidence and see whether we can obtain the desired result. We would like to reason about these pieces of evidence by abduction in an efficient manner.
– We would like to extend our formalism including a shift of burden of proof such as using presumption and two kinds of burden of proof studied in [3, 4].

Acknowledgements We are very grateful to anonymous reviewers for their constructive comments to improve the paper.

References

1. Dung, P., M., On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games, *Artif. Intell.* 77(2), pp. 321-358 (1995).
2. Prakken, H., Modelling Defeasibility in Law: Logic or Procedure?, *Fundam. Inform.*, 48(2-3), pp. 253-271 (2001).
3. Prakken, H., and Sartor, G., Presumptions and Burdens of Proof, *Proc. of Legal Knowledge and Information Systems, JURIX 2006: The Nineteenth Annual Conference*, pp. 21 – 30 (2006).
4. Prakken, H., and Sartor, G., Formalising Arguments about the Burden of Persuasion, *Proc. of the Eleventh International Conference on Artificial Intelligence and Law*, to appear (2007).
5. Sartor, G., Legal Reasoning, A Cognitive Approach to the Law, *A Treatize of Legal Philosophy and General Jurisprudence*, Enrico Pattaro (ed.), Springer (2005).
6. Shindo, K., New Civil Procedure Law (Revised 3rd Edition), Kobundo publisher (in Japanese) (2005).

Appendix: Trace Program

```
:-dynamic indent/1.

demo:-proved(valid_contract,plaintiff,defendant).

p(valid_contract,P,O) :-
        proved(offer,P,O),
        proved(acceptance,P,O),
        default(valid_contract,sane,P,O).
p(offer,P,O):-
        print_trace(offer,P,O).
p(acceptance,P,O):-
        print_trace(acceptance,P,O).
p(-sane,P,O):-
        proved(document_insane,P,O),
```

```
        default(-sane,authentic_document,P,O).
p(document_insane,P,O):-
        print_trace(document_insane,P,O).
p(-authentic_document,P,O):-
        proved(-correct_stamp,P,O).
p(-correct_stamp,P,O):-
        print_trace(-correct_stamp,P,O).

default(C,X,P,O):-
        indent(N),N2 is N + 2,tab(N2),
        print('*** A shift of a burden of proof occurs. ***'),nl,
        tab(N2),
        print('In order to rebut '),print_prop(C),print(', '),nl,
        N4 is N + 4,tab(N4),
        print('it is sufficient for '),print(O),print(' to prove '),
        inverse(X,InX),print_prop(InX),print('.'),nl,
        \+ proved(InX,O,P).

proved(X,P,O):-retract(indent(N)),
        N2 is N + 2,
        tab(N2),
        print(P),print(' is now proving '),print_prop(X),
        print(' against '),print(O),print('.'),nl,
        assert(indent(N2)),fail.
proved(X,P,O):-p(X,P,O),inverse(X,InX),\+ p(InX,O,P),
        retract(indent(N2)),tab(N2),
        print(P),print(' has proved '),print_prop(X),
        print(' against '),print(O),print('.'),nl,
        N is N2-2,assert(indent(N)),!.
proved(X,P,O):-retract(indent(N2)),tab(N2),
        print(P),print(' has failed in proving '),print_prop(X),
        print(' against '),print(O),print('.'),nl,
        N is N2-2,assert(indent(N)),fail.

inverse(-X,X):-!.
inverse(X,-X):-!.

indent(0).

print_trace(X,P,_):-indent(N), N2 is N + 2, tab(N2), print(P),
        print(' claims that there is a piece of evidence for '),
        print_prop(X),print('.'),nl.

print_prop(-X):-!,print('"'),print('-'),print(X),print('"').
print_prop(X):-print('"'),print(X),print('"').
```