A Formalization for Burden of Production in Logic Programming

Ken Satoh[1]

National Institute of Informatics and Sokendai
ksatoh@nii.ac.jp

Abstract. In legal system, there are mainly two kinds of burdens of proof; one is a burden of persuasion and the other is a burden of production. These burdens have usually been formalized by an argumentation between a plaintiff/prosecutor and a defendant. In the previous work, we formalized a burden of persuasion from a different point of view; a decision making by a judge. If a burden of persuasion is not fulfilled, a judge will make a decision which is not favorable to the party having the burden. And as by-product, both parties are trying to fulfill their burden of persuasion. In this paper, we follow this idea in order to formalize a burden of production as well. We introduce a new predicate which a judge uses in order to check whether a party with the burden of production raised the issues and provided an enough evidence during the proceedings.

1 Introduction

In legal system, there are mainly two kinds of burdens of proof; a burden of persuasion and a burden of production[3]. A burden of persuasion 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 [8]. A burden of production is a duty upon a party to raise a favorable issue to the party to make the judge consider the issue. The burden usually includes not only raising the issue, but also providing enough evidence to support the issue.

For example, in a murder case, if a prosecutor would like to assert that a man committed a murder, the prosecutor must show the following.

- The man had an intention of killing.
- The man made a killing action.
- As a result the killed died.

If any of the above conditions are "non liquet", they are regarded as false and the man is not punished. In this case, we say that the prosecutor has a burden of persuasion of these facts since if the prosecutor fails to prove these facts, the conclusion is not in favor of the prosecutor.

On the other hand, even if the above facts are decided to be true, the defendant can argue that the man was insane at the murder or the man did so in order to defense himself. In this case, if the defendant keeps silent, these facts are not considered by the judge. But once it is raised by the defendant properly, the judge must decide whether these facts are true or not. In this case, we say that the defendant has a burden of persuasion of these facts since if the defendant does not raise these facts, the conclusion is not in favor of the defendant. However, in this case, the burden of persuasion still resides in the prosecutor. That is, if one of these facts are "non liquet" then by the principle of "in dubio pro reo", the man is not punished. This means that "non liquet" facts such as insanity or self defense are regarded as true in this case.

This is a special phenomenon for a criminal law case since in a civil law case the burden of persuasion and the burden of production usually resides in the same party. In [6], we formalized a switch of burden of proof where the burden of proof actually means the burden of persuasion. This is because it is enough to consider a burden of persuasion in a civil law case. In this paper, we formalize a burden of production in terms of logic programming. We introduce a notion of allegement which means that a party which has a burden raised an issue and provided enough evidence for the issue. It is because a burden of persuasion is used for the decision making after all evidences are provided at the end of proceedings whereas a burden of production is used for checking whether one party raised the issue and provided a sufficient pieces of evidence.

2 Previous Formalization of Burden of Persuasion

In this section, we review our previous formalization of a burden of persuasion [6, 7]. We formalized a burden of persuasion as a tool of decision making by judge when an ultimate fact is "non liquet". The formalization is quite different from a usual formalization of the burden of persuasion; in a usual one the burden of persuasion is formalized as a dialogue game. In our formalism, we do not formalize a dynamical nature of dialogue, but how to decide the case according to a set of given evidence. When an ultimate fact is "non liquet", the default value of the fact is used by the judge and as a side-effect, if the default value is beneficiary to one party, then the other party must prove the negation of the default value to prevent the status of "non liquet".

For this purpose, we introduced `plausible` predicate[1] . `plausible(Fact)` means that given the current evidence, plausibility of the fact `Fact` is above the degree of proof. We express the situation that `Fact` is proved as "`prove(Fact)`" and defined in logic programming as follows

```
  prove(Fact):- plausible(Fact).
```

Moreover, "non liquet" status can be expressed as "`not plausible(Fact)`, `not plausible(-Fact)`" where `not` is negation as failure in logic programming and `-Fact` is the negation of `Fact`. Then, we express the default value, `default`

[1] In [6, 7], we called it p-predicate. Note that we follow the definition of [7] because of easiness of explanation.

(`Fact`) of `Fact` with the consideration of the burden of proof that `Fact` is really plausible or the opponent proves otherwise can be represented as follows.

```
  default(Fact):- plausible(Fact).
  default(Fact):- not plausible(Fact), not plausible(-Fact).
```

This is actually equivalent as follows.

```
  default(Fact):- not prove(-Fact).
```

This means that if a party which is not favorable for `Fact` cannot show the negation of `Fact`, then `Fact` is regarded as proved.

In a general case, we use 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 `F_1,...,F_n`) and the negation of the other of them should be proved by the opponent (we present these conditions as `-E_1,...,-E_m`) to rebut the conclusion. Then, we have the following rule:

```
prove(C) :-
     prove(F_1), prove(F_2),..., prove(F_n),
     default(E_1),...,default(E_m).
```

This is equivalent to:

```
prove(C) :-
     prove(F_1), prove(F_2),..., prove(F_n),
     not prove(-E_1),...,not prove(-E_m).
```

Moreover, we introduced the second argument to the `prove` predicate in order to clarify which party has the burden. So, `prove(Fact,Party)` means that party `Party` have the burden to prove `Fact`. Then, the above rule become:

```
prove(C,P) :-
    prove(F_1,P), prove(F_2,P),..., prove(F_n,P),opposite(P,O),
    not prove(-E_1,O),...,not prove(-E_m,O).
```

where `opposite(P,O)` expresses that `P` and `O` are opposite parties each other.

However, we could not formalize the murder example with the `prove` predicate. Suppose that we use the above formalization because `-self_defense` is a default.

```
prove(murder,prosecutor):-
    prove(murder_intention,prosecutor),
    prove(murder_action,prosecutor),
    prove(death,prosecutor),
    not prove(self_defense,defendant).
```

Suppose
`prove`(`murder_intention`,`prosecutor`),
`prove`(`murder_action`,`prosecutor`) and

`prove (death,prosecutor)` hold.
Consider that the defendant produced evidence for `self_defense` in order to accomplish his burden of production and that then the prosecutor produced evidence for `-self_defense` which makes the truth value of `self_defense` becomes "non-liquet", but is not powerful enough to negate `self_defense`. Then, `not prove (self_defense,defendant)` becomes true and therefore, `prove (murder,prosecutor)` is derived. However, it is against the principle of "in dubio pro reo" so the person must not be punished.

The alternative formalization would be

```
prove(murder,prosecutor):-
    prove(murder_intention,prosecutor),
    prove(murder_action,prosecutor),
    prove(death,prosecutor),
    prove(-self_defense,prosecutor).
```

where we use `prove (-self_defense,prosecutor)` instead of default. Then, the burden of persuasion resides in the prosecutor. However, Suppose again that
`prove (murder_intention,prosecutor)`,
`prove (murder_action,prosecutor)` and
`prove (death,prosecutor)` hold
and that the prosecutor and the defendant remained silent about `self_defense`, then `prove (-self_defense,prosecutor)` does not hold so `prove (murder,prosecutor)` is not derived. However, `prove (murder,prosecutor)` should be proved since defendant did not accomplished his burden of production for `self_defense`, so `self_defense` should be decided to be false.

This problem is actually caused by trying to use `prove` predicate in order to check whether an issue is raised or not. But, the intended functionality of `prove` is to use them *during the decision making process*, but the burden of production is considered *before the decision making process is started.* Therefore, we need another predicate to formalize a burden of production during the fact finding process.

3 Formalization of Burden of Production

To formalize a burden of production, we introduce `allege` predicate. We use the same idea of formalization of burden in the previous work; the burden is used as a tool of decision making by a judge. So, a formalization is used when the proceedings is finished and a judge is about to make a decision. `allege (Fact,Party)` means that a party (defendant or plaintiff or prosecutor) `Party` alleged `Fact` and gave some reasonable pieces of evidence of `Fact` in the proceedings. So, if a rule has `allege (Fact,Party)` in the body, then without `Party`'s raising the issue of `Fact`, the rule is not considered by the judge since `Party` did not fulfill their burden of production. Therefore, if one party has the burden of production of `Fact`, the party must make `allege (Fact,Party)` true in the proceedings. In civil law, the burden of persuasion and the burden of production usually coincide at least in Japan, so we normally do not consider the burden of production. However, in criminal law, all the burdens of persuasion reside in the prosecutor, but the burden of production is divided into two parts and one part expressing the exceptional situation should be raised by the defendant.

Now, we show a formalization of the murder example in Introduction using `allege` predicate[2] .

```
prove(murder,prosecutor):-
    allege(murder_intention,prosecutor),
    allege(murder_action,prosecutor),
    allege(death,prosecutor),
    prove(murder_intention,prosecutor),
    prove(murder_action,prosecutor),
    prove(death,prosecutor),
    not defense(murder,defendant).

defense(murder,defendant):-
    allege(insane,defendant),
    not prove(-insane,prosecutor).
defense(murder,defendant):-
    allege(self_defense,defendant),
    not prove(-self_defense,prosecutor).
```

Suppose that a prosecutor fulfilled his burden of production for `murder_intention`, `murder_action` and `death`[3] and his burden of persuasion for these facts[4] . Moreover, suppose that the defendant alleged `self_defense` and provided evidence for `self_defense` to let `allege` (`self_defense`,`defendant`) hold. Then, suppose that the prosecutor produced evidence for `-self_defense` which makes the truth value of `self_defense` becomes "non-liquet", but is not powerful enough to negate `self_defense`. Then, since `prove` (`-self_defense`,`prosecutor`) does not hold, `defense` (`murder`,`defendant`) holds and therefore `prove`(`murder`,`prosecutor`) is not derived. This exactly reflects the principle of "in dubio pro reo".

On the other hand, suppose that the prosecutor fulfilled his burden of production for `murder_intention`, `murder_action` and `death` and his burden of persuasion for these facts and that the defendant remained silent about `self_defense`,

[2] Some reader might think that the order of predicates (`allege` and `prove` predicates) matters in the rule and we use a PROLOG-like right-to-left evaluation scheme to represent such an order. However, it is not actually true. `allege` predicate means whether a party with the burden of production raised the issue and provided enough pieces of evidence in the proceedings and therefore the truth value of `allege` is evaluated after the proceedings is finished when a judge is about to make a decision over cases. So, it can be evaluated correctly even if it is put after `prove` predicate in the body of the rule.

[3] This is expressed by the situation `allege` (`murder_intention`,`prosecutor`), `allege` (`murder_action`,`prosecutor`) and `allege` (`death`,`prosecutor`) hold.

[4] This is expressed by the situation `prove` (`murder_intention`,`prosecutor`), `prove` (`murder_action`,`prosecutor`) and `prove` (`death`,`prosecutor`) hold.

then allege(self_defense,defendant) does not hold and defense (murder,defendant) does not hold and therefore prove (murder,prosecutor) is derived. This also corresponds with actual legal conclusion.

Note that in both cases, since the defendant did not allege insane, insane is never considered at the decision making process.

In a general case, we make the following rules. For a conclusion C, we divide ultimate facts used to prove conclusions into the two categories; one of them should be alleged and proved by the proponent (we represent as F_1,...,F_n) and the other of them should be alleged by the opponent proved by the proponent (we present these conditions as -E_1,...,-E_m) to raise exceptional claims. Then, we have the following rule:

```
prove(C,P) :-
     allege(F_1,P), allege(F_2,P),...,allege(F_n,P),
     prove(F_1,P), prove(F_2,P),..., prove(F_n,P),
     opposite(P,O),
     not defense(C,O).

defense(C,O):-
     allege(E_1,O), opposite(P,O), prove(-E_1,P).
...
defense(C,O):-
     allege(E_m,O), opposite(P,O), prove(-E_m,P).
```

In Appendix A we show a PROLOG program in which we add a function of printing an execution trace to the above program and in Appendices B and C, we show execution traces for examples in the murder case discussed above.

4 Related Work

In [6], we gave a formalization of burden of persuasion and showed that a switch of burden of persuasion can be expressed correctly in the nonmonotonic formalization without any additional functions which Prakken used in [2] by using an interpretation of burden of proof in the Japanese civil procedure law [8]. In [7], we extended [6] to introduce abductive predicate to choose the preferred theory for the proponent if the theories of interpretation of burden of persuasions are split.

In [3–5], Prakken and Sartor formalized a burden of persuasion and a burden of production in their IS logic. In IS logic, they used naming of rules and introduce meta-reasoning for application of rules. In this paper, we do not use such an advanced mechanism and show that introducing a new predicate in logic programming is enough to capture a burden of production.

Gordon et al. handled not only a burden of persuasion but also a burden of production in their Carneades system[1]. They formalized a burden of proof as a kind of dynamic process of arguments and therefore, they needed to introduce additional mechanisms to handle these burdens. Although these additional mechanisms may be necessary for capturing a notion of a burden of proofs in dynamic process, we showed that a simple mechanism of "negation as failure" in logic programming is sufficient for a burden of production for a decision making process.

5 Conclusion

In this paper, we formalize a burden of production by introducing a new predicate `allege` which is used to express that the opponent claims a fact as a defense and provide some evidence for the fact.

As a future research, we would like to study how to divide ultimate facts into normal predicate and exceptional predicate and how to combine burden of proof and fact finding process.

References

1. Gordon, T. F., Prakken, H., Walton, D., "The Carneades Model of Argument and Burden of Proof", Artif. Intell., Vol. 171, Nos. 10-15, pp. 875 – 896 (2007).
2. Prakken, H., Modelling Defeasibility in Law: Logic or Procedure?, *Fundam. Inform.*, 48(2-3), pp. 253 – 271 (2001).
3. Prakken, H., Sartor, G., Presumptions and Burdens of Proof, Proc. of JURIX 20006, pp. 21 – 30 (2006).
4. Prakken, H., Sartor, G., Formalising Arguments about the Burden of Persuasion, Proc. of ICAIL 2007, pp. 97 – 106 (2007).
5. Prakken, H., Sartor, G., "More on Presumptions and Burdens of Proof", Proc. of JURIX 2008, pp. 176 – 185 (2008).
6. Satoh, K., Tojo, S., Suzuki, Y.,
"Formalizing a Switch of Burden of Proof by Logic Programming", Proc. of the 1st International Workshop on Juris-Informatics (JURISIN 2007), pp. 76 – 85 (2007). (`http://research.nii.ac.jp/~ksatoh/papers/jurisin2007.pdf`)
7. Satoh, K., Tojo, S., Suzuki, Y., "Abductive Reasoning for Burden of Proof", Proc. of the 2nd International Workshop on Juris-Informatics (JURISIN 2008), pp. 93 – 102 (2008). (`http://research.nii.ac.jp/~ksatoh/papers/jurisin2008.pdf`)
8. Shindo, K., New Civil Procedure Law (Revised 3rd Edition), Kobundo publisher (in Japanese) (2005).

Appendix A: PROLOG PROGRAM

```
:-dynamic indent/1.
:-dynamic avoidance/2.

:- op(1100, xfx, user:(<=)).

demo:- abolish(indent/1),
       assert(indent(0)),
       prove_main(murder,prosecutor).

prove_main(murder,P):-
    allege(murder_intention,P),
    allege(murder_action,P),
    allege(death,P),
    print_message(proof_start,P,murder),
    prove(murder_intention,P),
    prove(murder_action,P),
    prove(death,P),
    opposite(P,O),
   ((\+defense(murder,O))->
   (print_message(proof_success,P,murder),!);
   (print_message(proof_failure,P,murder),!,fail)).

prove(X,P):-
    print_message(proof_start,P,X),
  ((plausible(X,P),
    print_message(plausible_evidence,P,X))->
        (print_message(proof_success,P,X),!);
        (print_message(proof_failure,P,X),!,fail)).

defense(X,O):-
    avoidance(X,Defense),
    prove_avoidance(X,Defense,O).

prove_avoidance(C,X,O):-
   allege(X,O),
   print_message(avoidance_start,O,C,X),
  ((inverse(X,InX),opposite(O,P),
    \+prove(InX,P))->
        (print_message(avoidance_success,O,C,X),!);
        (print_message(avoidance_fail,O,C,X),!,fail)).

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

```
opposite(prosecutor,defendant):-!.
opposite(defendant,prosecutor):-!.

print_message(proof_start,P,X):-
    increment_indent(N2), tab(N2),
    print(P), print(' tried to prove '),
    print_proc(X), print('.'), nl.
print_message(proof_success,P,X):-
    decrement_indent(N2), tab(N2),
    print(P), print(' successfully proved '),
    print_proc(X), print('.'), nl.
print_message(proof_failure,P,X):-
    decrement_indent(N2), tab(N2),
    print(P),print(' failed to prove '),
    print_proc(X), print('.'), nl.
print_message(plausible_evidence,_,X):-
    indent(N), N2 is N + 2, tab(N2),
    print_proc(X), print(' is determined to be plausible.'),nl.
print_message(avoidance_start,O,A,X):-
    increment_indent(N2), tab(N2),
    print(O),print(' alleges '),print_proc(X),
    print(' as a defense against '), print_proc(A),
    print('.'),nl.
print_message(avoidance_success,O,C,X):-
    decrement_indent(N2), tab(N2),
    print(O), print(' successfully proved '),
    print_proc(X), print(' as a defense against '),
    print_proc(C), print('.'),nl.
print_message(avoidance_fail,O,C,X):-
    decrement_indent(N2), tab(N2),
    print(O), print(' failed to claim '),
    print_proc(X), print(' as a defense against '),
    print_proc(C), print('.'),nl.
print_message(alleged,X,P):-
    indent(N), N2 is N + 2, tab(N2),
    print(P), print(' alleges '),
    print_proc(X),print('.'),nl.

increment_indent(N2):-
    retract(indent(N)), N2 is N+2,
    assert(indent(N2)).
decrement_indent(N2):- retract(indent(N2)),
    N is N2-2, assert(indent(N)).

print_proc(-X):-print('"-'),print(X),print('"').
print_proc(X):- print('"'),print(X),print('"').

%%%%%%%%%%%%%% avoidances for rule application %%%%%%%%%%%%%%

avoidance(murder,insanity).
avoidance(murder,self_defense).

%%%%%%%%%%%%%%%% allegement, plausibility of cases %%%%%%%%%%%%%%
allege(murder_intention,prosecutor).
plausible(murder_intention,prosecutor).

allege(murder_action,prosecutor).
plausible(murder_action,prosecutor).

allege(death,prosecutor).
plausible(death,prosecutor).

allege(self_defense,defendant).
% plausible(-self_defense,prosecutor).
```

Appendix B: EXECUTION TRACE for the above program

```
  prosecutor tried to prove "murder".
    prosecutor tried to prove "murder_intention".
      "murder_intention" is determined to be plausible.
    prosecutor successfully proved "murder_intention".
    prosecutor tried to prove "murder_action".
      "murder_action" is determined to be plausible.
    prosecutor successfully proved "murder_action".
    prosecutor tried to prove "death".
      "death" is determined to be plausible.
    prosecutor successfully proved "death".
    defendant alleges "self_defense"
      as a defense against "murder".
      prosecutor tried to prove "-self_defense".
      prosecutor failed to prove "-self_defense".
    defendant successfully claimed "self_defense"
      as a defense against "murder".
  prosecutor failed to prove "murder".
```

Appendix C: EXECUTION TRACE for the above program + "`plausible(-self_defense,prosecutor).`"

```
prosecutor tried to prove "murder".
  prosecutor tried to prove "murder_intention".
    "murder_intention" is determined to be plausible.
  prosecutor successfully proved "murder_intention".
  prosecutor tried to prove "murder_action".
    "murder_action" is determined to be plausible.
  prosecutor successfully proved "murder_action".
  prosecutor tried to prove "death".
    "death" is determined to be plausible.
  prosecutor successfully proved "death".
  defendant alleges "self_defense"
    as a defense against "murder".
    prosecutor tried to prove "-self_defense".
      "-self_defense" is determined to be plausible.
    prosecutor successfully proved "-self_defense".
  defendant failed to claim "self_defense"
    as a defense against "murder".
prosecutor successfully proved "murder".
```