Private-Safe (Logic-based) Decision Systems for Energy Assignment in Agricultural Cooperatives

Luciana Fidilio-Allende Joaquín Arias

Grupo de Inteligencia Artificial de la URJC
Center for Intelligent Information Technologies (CETINIA)
Móstoles, Madrid

26-28 Junio 2024 (PAAMS'24)

Introduction

- Automated Decision Makers, even if they are value-aware, can only be trustworthy if they are capable of explaining their decisions (XAI).
- Logic-based systems, such as s(CASP) [3], a goal-directed execution of Answer Set Programming (ASP) [7], can provide justifications.
  - But the justifications (and the ASP models themselves) may expose sensitive information (violating privacy and/or legislation).

Introduction

- Automated Decision Makers, even if they are value-aware, can only be trustworthy if they are capable of explaining their decisions (XAI).
- Logic-based systems, such as s(CASP) [3], a goal-directed execution of Answer Set Programming (ASP) [7], can provide justifications.
  - But the justifications (and the ASP models themselves) may expose sensitive information (violating privacy and/or legislation).
- Alternatives: (i) Manipulate the justifications [1]

Introduction

- Automated Decision Makers, even if they are value-aware, can only be trustworthy if they are capable of explaining their decisions (XAI).
- Logic-based systems, such as s(CASP) [3], a goal-directed execution of Answer Set Programming (ASP) [7], can provide justifications.
  - But the justifications (and the ASP models themselves) may expose sensitive information (violating privacy and/or legislation).
- Alternatives: (i) Manipulate the justifications [1]

Example from [9]

```
1 person(X) :- ustaff(X).
2 ustaff(X) :- professor(X).
3 professor(mary).
```

Justification for `?- person(mary)`

```
1 person(mary) :-
2   ustaff(mary) :-
3       professor(mary).
```

Introduction

- Automated Decision Makers, even if they are value-aware, can only be trustworthy if they are capable of explaining their decisions (XAI).
- Logic-based systems, such as s(CASP) [3], a goal-directed execution of Answer Set Programming (ASP) [7], can provide justifications.
  - But the justifications (and the ASP models themselves) may expose sensitive information (violating privacy and/or legislation).
- Alternatives: (i) Manipulate the justifications [1]

Example from [9]

```
person(X) :- ustaff(X).
ustaff(X) :- professor(X).
professor(mary).
```

Justification for `?- person(mary)`

```
'person' holds (for mary), because
  'ustaff' holds (for mary), because
      'professor' holds (for mary).
```

Introduction

- Automated Decision Makers, even if they are value-aware, can only be trustworthy if they are capable of explaining their decisions (XAI).
- Logic-based systems, such as s(CASP) [3], a goal-directed execution of Answer Set Programming (ASP) [7], can provide justifications.
  - But the justifications (and the ASP models themselves) may expose sensitive information (violating privacy and/or legislation).
- Alternatives: (i) Manipulate the justifications [1]

Example from [9]

```
person(X) :- ustaff(X).
ustaff(X) :- professor(X).
professor(mary).
```

Justification for `?- person(mary)`

```
'person' holds (for mary), because

      'professor' holds (for mary).
```

Introduction

- Automated Decision Makers, even if they are value-aware, can only be trustworthy if they are capable of explaining their decisions (XAI).
- Logic-based systems, such as s(CASP) [3], a goal-directed execution of Answer Set Programming (ASP) [7], can provide justifications.
  - But the justifications (and the ASP models themselves) may expose sensitive information (violating privacy and/or legislation).
- Alternatives: (i) Manipulate the justifications [1], or (ii) apply forgetting, a syntactic transformation that forgets predicates in ASP programs [9].

Example from [9]

```
person(X) :- ustaff(X).
ustaff(X) :- professor(X).
professor(mary).
```

```
{ ..., person(mary) }
```

Result after forgetting `ustaff/1`

```
person(X) :- professor(X).
professor(mary).
```

```
{ ..., person(mary) }
```

Introduction

- Automated Decision Makers, even if they are value-aware, can only be trustworthy if they are capable of explaining their decisions (XAI).
- Logic-based systems, such as s(CASP) [3], a goal-directed execution of Answer Set Programming (ASP) [7], can provide justifications.
  - But the justifications (and the ASP models themselves) may expose sensitive information (violating privacy and/or legislation).
- Alternatives: (i) Manipulate the justifications [1], or (ii) apply forgetting, a syntactic transformation that forgets predicates in ASP programs [9].

Limitations of forgetting proposals/operators

(i) They have technical limitations.

(ii) They are focused on propositional logic, or need to ground the variables.

Limitations of state-of-the-art Forgetting techniques

	(UP)	(SP)	Loops	Commutative	Variables	Constraints
f_{SU} [8]	Yes	No	Yes	No	No	No
f_{SP} [6]	Yes	Limited	No	No	No	No
f^*_{SP} [4]	Yes	Limited	Yes	Yes	No	No
f_{AC} [5]	Yes	Yes	Yes	Yes	No	No

- (UP): Uniform Persistence means that the original program and the forgetting result are equivalent even if we add new facts.
- (SP): Strong Persistence is similar to (UP) but adding new rules.
- Loops: Deal with even/odd loops (by adding auxiliary predicates).
- Commutative: Allow iterative application, regardless of the order.

Limitations of state-of-the-art Forgetting techniques (cont.)

	(UP)	(SP)	Loops	Commutative	Variables	Constraints
f_{SU} [8]	Yes	No	Yes	No	No	No
f_{SP} [6]	Yes	Limited	No	No	No	No
f^*_{SP} [4]	Yes	Limited	Yes	Yes	No	No
f_{AC} [5]	Yes	Yes	Yes	Yes	No	No
f_{CASP}	Yes	Yes	Yes	Yes	WiP	WiP

Our proposal f_{CASP} based on f_{SU}

- It is simple, iterable, and commutable.
- Uses the compiler of s(CASP) to generate dual rules.
- Supports loops and denials.

. . . and we believe that it can satisfy SP and, in the future, support Variables and Constraints.

Background: ASP

- Answer Set Programming (ASP) is based on the stable model semantics [7], supporting non-stratified negation:

```
p :- not q.
q :- not p.
```

Even loop: `{p}, {q}`

```
p :- not q.
q :- p.
```

Odd loop: `no models`

- In this work, we extended ASP with **double default negations** [10]: The clause `p :- not not p.` generates two models: `{p}` and `{}`.
  - Double negations can be modeled as even loops. For example, the predicate `p :- not not p` is transformed into:

```
p :- not neg_p.
neg_p :- not p.
```

Background: s(CASP)

- It is a top-down, goal-directed interpreter of ASP with Constraints [3].
- Can provide justifications (in natural language) [1].
  - They can be manipulated (to hide sensitive information) using the directive `#show` and the flag `--short`.
- It solves negated atoms (`not p`) against the dual rules of the program (the negation of the rules present in the program) [2]. E.g.:

```
1 % Original program
2 p(0).
3 p(X) :- q(X), not t(X,Y).
```

Background: s(CASP)

- It is a top-down, goal-directed interpreter of ASP with Constraints [3].
- Can provide justifications (in natural language) [1].
  - They can be manipulated (to hide sensitive information) using the directive `#show` and the flag `--short`.
- It solves negated atoms (`not p`) against the dual rules of the program (the negation of the rules present in the program) [2]. E.g.:

```
% Original program
p(0).
p(X) :- q(X), not t(X,Y).
```

```
% Dual rules
not p(X) :- not p1(X), not p2(X).
not p1(X) :- X\=0.
not p2(X) :- forall(Y, not p2_(X,Y)).
not p2_(X,Y) :- not q(X).
not p2_(X,Y) :- q(X), t(X,Y).
```

Background: The forgetting technique f_{CASP}

f_{CASP} consists on five steps that can be iteratively repeated to forget multiple predicates (consider we want to forget the predicate `p`):

1. Add auxiliary predicates due to even loops, facts and/or missing predicate.

   If `p` is part of an even loop, `not p` is replaced, and `neg_x :- not p` is added.

2. Generate the simplified dual rule(s) using s(CASP).

```
% Clauses of p
p :- t, not u.
p :- not r.
```

```
% s(CASP) dual rules
not p :- not p_1, not p_2.
not p_1 :- not t.
not p_1 :- u.
not p_2 :- r.
```

```
% Simplified dual rule(s)
not p :- not t, r.
not p :- u, r.
```

3. Forget the predicate and its negation.
4. Clean true/false and add double negations to preserve even loops.

   Repeat steps 1 to 4 to forget the next predicate...

5. (Optional) Transform double negations into even loops.

Energy assignment in agricultural cooperatives

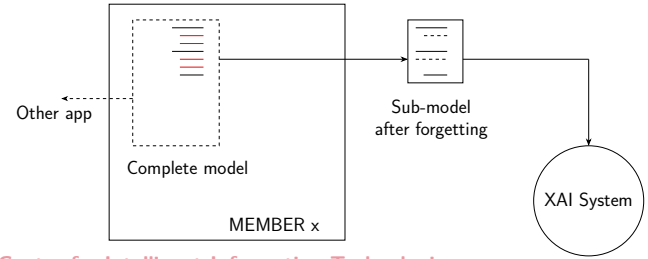
Ethical values in energy management

- In agricultural cooperatives, farmers share their assets to obtain a better and cheaper access to resources. The sharing of locally-generated energy can be a stable alternative supply in rural areas.
- Energy assignment can encourage better practices if we base the distribution criteria of the generated energy on values.

- We can consider values based on the 10 CAP objectives as criteria.
- For example, we can consider how fairly the farmer's workers are paid.

Fair energy assignment

- We propose an automated decision-making system for energy assignment in agricultural cooperatives using fair income as criteria.
  - Its decisions must be fair.
  - To trust a decision, a justification is required (XAI).
  - Additionally, the members of the cooperative may want to preserve business secrets (e.g., salary complements).



Fair energy assignment

- We propose an automated decision-making system for energy assignment in agricultural cooperatives using fair income as criteria.
  - Its decisions must be fair.
  - To trust a decision, a justification is required (XAI).
  - Additionally, the members of the cooperative may want to preserve business secrets (e.g., salary complements).

```
% Original clauses
salary(eric, Salary):-
  base_salary(eric, S0),
  distance_home_work(eric, S1),
  has_children(eric, S2),
  Salary is S0 + S1 + S2.
```

```
% Result after forgetting
salary(eric, Salary):-
  S0 = 1200,
  S1 = 100,
  S2 = 100,
  Salary is S0 + S1 + S2.
```

Fair energy assignment

- In other scenarios the clauses involve even loops.

```
% Original clauses
over_40_bea :-
  not neg_over_40_bea.
neg_over_40_bea:-
  not over_40_bea.

generational_renewal(bea, 0):-
  over_40_bea.
generational_renewal(bea, 100):-
  not over_40_bea.

salary(bea, Salary):-
  base_salary(bea, S0),
  generational_renewal(bea, S1),
  holiday_worked(bea, S2),
  Salary is S0 + S1 + S2.
```

Fair energy assignment

- In other scenarios the clauses involve even loops.

```
% Original clauses
over_40_bea :-
  not neg_over_40_bea.
neg_over_40_bea:-
  not over_40_bea.

generational_renewal(bea, 0):-
  over_40_bea.
generational_renewal(bea, 100):-
  not over_40_bea.

salary(bea, Salary):-
  base_salary(bea, S0),
  generational_renewal(bea, S1),
  holiday_worked(bea, S2),
  Salary is S0 + S1 + S2.
```

Fair energy assignment

- In other scenarios the clauses involve even loops.

```
% Original clauses
over_40_bea :-
  not neg_over_40_bea.
neg_over_40_bea:-
  not over_40_bea.

generational_renewal(bea, 0):-
  over_40_bea.
generational_renewal(bea, 100):-
  not over_40_bea.

salary(bea, Salary):-
  base_salary(bea, S0),
  generational_renewal(bea, S1),
  holiday_worked(bea, S2),
  Salary is S0 + S1 + S2.
```

```
% Result after forgetting
neg_1 :- not neg_2.
neg_2 :- not neg_1.

salary(bea, Salary):-
  S0 = 900,
  neg_2,
  S1 = 0,
  S2 = 0,
  Salary is S0 + S1 + S2.
salary(bea, Salary):-
  S0 = 900,
  neg_1,
  S1 = 100,
  S2 = 0,
  Salary is S0 + S1 + S2.
```

Fair energy assignment

- Let's see the justifications for `?- salary(bea).`

```
% Answer 1
salary(bea,1000) :-
    base_salary(bea,900),
    generational_renewal(bea,100) :-
        not over_40_bea :-
            neg_over_40_bea :-
              chs(not over_40_bea).
    holiday_worked(bea,0),
    1000 is 900+100+0.
```

```
% Answer 2
salary(bea,900) :-
    base_salary(bea,900),
    generational_renewal(bea,0) :-
        over_40_bea :-
            not neg_over_40_bea :-
              chs(over_40_bea).
    holiday_worked(bea,0),
    900 is 900+0+0.
```

Fair energy assignment

- Let's see the justifications for `?- salary(bea).`

```
% Answer 1
salary(bea,1000) :-
    base_salary(bea,900),
    generational_renewal(bea,100) :-
        not over_40_bea :-
            neg_over_40_bea :-
              chs(not over_40_bea).
    holiday_worked(bea,0),
    1000 is 900+100+0.
```

```
% Answer 1
salary(bea,1000) :-
  neg_1 :-
      not neg_2 :-
          chs(neg_1).
  1000 is 900+100+0.
```

Conclusions

- We have presented a (simple) use case of the forgetting operator f_{CASP}, modelling a decision system for a fair distribution of the energy in an agricultural cooperative.
- In this use case, the application of the operator has made possible to preserve the confidentiality of the farmers

  ...while maintaining the explainability of the model.

Conclusions

- We have presented a (simple) use case of the forgetting operator f_{CASP}, modelling a decision system for a fair distribution of the energy in an agricultural cooperative.
- In this use case, the application of the operator has made possible to preserve the confidentiality of the farmers

  ... while maintaining the explainability of the model.

Future Work

- Expand f_{CASP} to support generic CASP programs.
- Test the technical limits of the operator.
- Provide a formal proof of the f_{CASP} algorithm's correctness.

Conclusions

- We have presented a (simple) use case of the forgetting operator f_{CASP}, modelling a decision system for a fair distribution of the energy in an agricultural cooperative.
- In this use case, the application of the operator has made possible to preserve the confidentiality of the farmers

  ...while maintaining the explainability of the model.

Future Work

- Expand f_{CASP} to support generic CASP programs.
- Test the technical limits of the operator.
- Provide a formal proof of the f_{CASP} algorithm's correctness.

THANKS!

Bibliography I

[1] Arias, Joaquín, Carro, Manuel, Chen, Zhuo, and Gupta, Gopal (2020). **Justifications for Goal-Directed Constraint Answer Set Programming**. In: *Proceedings 36th International Conference on Logic Programming (Technical Communications)*. Vol. 325. EPTCS. Open Publishing Association, pp. 59–72. DOI: 10.4204/EPTCS.325.12.

[2] — (2022). **Modeling and Reasoning in Event Calculus using Goal-Directed Constraint Answer Set Programming**. In: *Theory and Practice of Logic Programming* 22.1, pp. 51–80. DOI: 10.1017/S1471068421000156.

[3] Arias, Joaquín, Carro, Manuel, Salazar, Elmer, Marple, Kyle, and Gupta, Gopal (2018). **Constraint Answer Set Programming without Grounding**. In: *Theory and Practice of Logic Programming* 18.3-4, pp. 337–354. DOI: 10.1017/S1471068418000285.

[4] Berthold, Matti (2022). **On Syntactic Forgetting with Strong Persistence**. In: *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022*. URL: https://proceedings.kr.org/2022/5/.

Bibliography II

[5] Berthold, Matti, Gonçalves, Ricardo, Knorr, Matthias, and Leite, João (2019a). **A Syntactic Operator for Forgetting that Satisfies Strong Persistence**. In: vol. 19. 5-6, pp. 1038–1055. DOI: 10.1017/S1471068419000346.

[6] — (2019b). **Forgetting in Answer Set Programming with Anonymous Cycles**. In: Lecture Notes in Computer Science 11805, pp. 552–565. DOI: 10.1007/978-3-030-30244-3_46.

[7] Gelfond, Michael and Lifschitz, Vladimir (1988). **The Stable Model Semantics for Logic Programming**. In: *5th International Conference on Logic Programming*, pp. 1070–1080. DOI: 10.2307/2275201.

[8] Gonçalves, Ricardo, Janhunen, Tomi, Knorr, Matthias, and Leite, João (2021). **On Syntactic Forgetting Under Uniform Equivalence**. In: *Logics in Artificial Intelligence - 17th European Conference, JELIA 2021*. Vol. 12678. Lecture Notes in Computer Science. Springer, pp. 297–312. DOI: 10.1007/978-3-030-75775-5_20.

Bibliography III

[9] Gonçalves, Ricardo, Knorr, Matthias, and Leite, João (2023). **Forgetting in Answer Set Programming - A Survey**. In: *Theory Pract. Log. Program.* 23.1, pp. 111–156. DOI: 10.1017/S1471068421000570. URL: https://doi.org/10.1017/S1471068421000570.

[10] Lifschitz, Vladimir, Tang, Lappoon R, and Turner, Hudson (1999). **Nested expressions in logic programs**. In: *Annals of Mathematics and Artificial Intelligence* 25, pp. 369–389. DOI: 10.1023/A:1018978005636.