CETINIA

Universidad Rey Juan Carlos

Modeling Administrative Discretion Using Goal-Directed Answer Set Programming

Joaquín Arias[1] Mar Moreno-Rebato[1]
Jose A. Rodriguez-García[1] Sascha Ossowski[1]

[1]CETINIA, Universidad Rey Juan Carlos

CAEPIA'21 – Málaga, September 24, 2021

Introduction

- Formal representation & automated reasoning of legal texts:
  - Interest in smart contracts, and public administrations [6; 8; 14].
  - For deterministic rules: proposals on logic-based programming languages [11; 13].
- However, none of them are able to represent the ambiguity and/or administrative discretion.

Introduction

- Formal representation & automated reasoning of legal texts:
  - Interest in smart contracts, and public administrations [6; 8; 14].
  - For deterministic rules: proposals on logic-based programming languages [11; 13].
- However, none of them are able to represent the ambiguity and/or administrative discretion.

Our Proposal: s(LAW)

- Based on s(CASP) [Arias et al. 2018], a non-monotonic reasoner:
  - Evaluates Answer Set Programs with Constraints.
- Allows modeling legal rules involving ambiguity:
  - E.g. awarding school places in “Comunidad de Madrid” is determined based on proximity to a family’s home, except in cases of *force majeure*.
- Supports reasoning and infers conclusion based on these rules.
- Provides justifications of the inferences (in NL) [Arias et al. 2020].

Background: (Constraint) Logic Programming

- Based on 1^{st} order logic.
- LP concatenates list X and Y to obtain Z

```
1  append(    [], Ys,     Ys).        ?- append([1],[2,3], Z).
2  append([X|Xs], Ys, [X|Zs]) :-        Z = [1,2,3] ?
3       append(Xs, Ys, Zs).
```

Background: (Constraint) Logic Programming

- Based on 1st order logic.
- LP concatenates list X and Y to obtain Z

...and much more

```
?- append(X, Y, [1,2,3]).
```

```
1  append(    [], Ys,     Ys).
2  append([X|Xs], Ys, [X|Zs]) :-
3       append(Xs, Ys, Zs).
```

```
?- append([1],[2,3], Z).
  Z = [1,2,3] ?
```

Background: (Constraint) Logic Programming

- Based on 1st order logic.
- LP concatenates list X and Y to obtain Z

...and much more

```
1  append(    [], Ys,     Ys).
2  append([X|Xs], Ys, [X|Zs]) :-
3       append(Xs, Ys, Zs).
```

```
?- append([1],[2,3], Z).
  Z = [1,2,3] ?
```

```
?- append(X, Y, [1,2,3]).

  X = [], Y = [1,2,3] ?;
  X = [1],  Y = [2,3] ?;
  X = [1,2],  Y = [3] ?;
  X = [1,2,3], Y = [] ?
```

Background: (Constraint) Logic Programming

- Based on 1st order logic.
- LP concatenates list X and Y to obtain Z

...and much more

```
1  append(    [], Ys,     Ys).
2  append([X|Xs], Ys, [X|Zs]) :-
3        append(Xs, Ys, Zs).
```

```
?- append([1],[2,3], Z).
   Z = [1,2,3] ?
```

```
?- append(X, Y, [1,2,3]).
   X = [], Y = [1,2,3] ?;
   X = [1],  Y = [2,3] ?;
   X = [1,2],  Y = [3] ?;
   X = [1,2,3], Y = [] ?
```

- Under CLP a mortgage relation can be defined as:

P=principal, T=time periods, R=repayment each period, I=interest rate, B=balance owing.

```
mg(P, T, _, _, B) :- T #= 0, B #= P.
mg(P, T, R, I, B) :- T #>= 1, NP #= P + P*I - R, NT #= T - 1, mg(NP, NT, R, I, B).
```

Background: (Constraint) Logic Programming

- Based on 1st order logic.
- LP concatenates list X and Y to obtain Z

...and much more

```
1  append(    [], Ys,     Ys).
2  append([X|Xs], Ys, [X|Zs]) :-
3        append(Xs, Ys, Zs).
```

```
?- append([1],[2,3], Z).
  Z = [1,2,3] ?
```

```
?- append(X, Y, [1,2,3]).
  X = [], Y = [1,2,3] ?;
  X = [1],  Y = [2,3] ?;
  X = [1,2],  Y = [3] ?;
  X = [1,2,3], Y = [] ?
```

- Under CLP a mortgage relation can be defined as:

P=principal, T=time periods, R=repayment each period, I=interest rate, B=balance owing.

```
mg(P, T, _, _, B) :- T #= 0, B #= P.
mg(P, T, R, I, B) :- T #>= 1, NP #= P + P*I - R, NT #= T - 1, mg(NP, NT, R, I, B).
```

```
?- mg(1000, 10, 150, 0.10, B).
  B = 203.13 ?
```

```
?- mg(P, 10, 150, 0.10, 0).
  P = 921.68 ?
```

Background: (Constraint) Logic Programming

- Based on 1st order logic.
- LP concatenates list X and Y to obtain Z

...and much more

```
1  append(    [], Ys,     Ys).
2  append([X|Xs], Ys, [X|Zs]) :-
3        append(Xs, Ys, Zs).
```

```
?- append([1],[2,3], Z).
  Z = [1,2,3] ?
```

```
?- append(X, Y, [1,2,3]).
  X = [], Y = [1,2,3] ?;
  X = [1],  Y = [2,3] ?;
  X = [1,2],  Y = [3] ?;
  X = [1,2,3], Y = [] ?
```

- Under CLP a mortgage relation can be defined as:

P=principal, T=time periods, R=repayment each period, I=interest rate, B=balance owing.

```
mg(P, T, _, _, B) :- T #= 0, B #= P.
mg(P, T, R, I, B) :- T #>= 1, NP #= P + P*I - R, NT #= T - 1, mg(NP, NT, R, I, B).
```

```
?- mg(1000, 10, 150, 0.10, B).
  B = 203.13 ?
```

```
?- mg(P, 10, 150, 0.10, 0).
  P = 921.68 ?
```

```
?- mg(P, 10, R, 0.10, B).
  P = 6.14*R + 0.38*B ?
```

Background: s(CASP)

- s(CASP), based on stable model semantics, supports non-stratified negation.
- With s(CASP) we can define different worlds (models).
  - E.g., on saturday, Bob either goes to the opera or stays home.

```
opera(saturday) :- not home(saturday).
home(saturday) :- not opera(saturday).
```

```
%% Model 1
{ opera(saturday) }
```

```
%% Model 2
{ home(saturday) }
```

Background: s(CASP)

- s(CASP), based on stable model semantics, supports non-stratified negation.
- With s(CASP) we can define different worlds (models).
  - E.g., on saturday, Bob either goes to the opera or stays home.

```
opera(saturday) :- not home(saturday).
home(saturday) :- not opera(saturday).
```

```
%% Model 1
{ opera(saturday) }
```

```
%% Model 2
{ home(saturday) }
```

  - With a more complex example...

```
opera(D) :- not home(D).
home(D) :- not opera(D).
home(monday).

:- baby(D), opera(D).

baby(tuesday).

?- opera(D).
```

```
% A day D, Bob either goes to the opera...
%                        ... or stays home.
% On Monday, Bob stays at home.

% When Bob's best friend comes with her baby, it is
% not a good idea to take the baby to the opera.
% They come on Tuesday.

% QUERY: When might Bob go to the opera?
```

Background: s(CASP)

- s(CASP), based on stable model semantics, supports non-stratified negation.
- With s(CASP) we can define different worlds (models).
  - E.g., on saturday, Bob either goes to the opera or stays home.

```
opera(saturday) :- not home(saturday).
home(saturday) :- not opera(saturday).
```

```
%% Model 1
{ opera(saturday) }
```

```
%% Model 2
{ home(saturday) }
```

  - With a more complex example...

```
opera(D) :- not home(D).
home(D) :- not opera(D).
home(monday).

:- baby(D), opera(D).

baby(tuesday).

?- opera(D).
```

```
% A day D, Bob either goes to the opera...
%                        ... or stays home.
% On Monday, Bob stays at home.

% When Bob's best friend comes with her baby, it is
% not a good idea to take the baby to the opera.
% They come on Tuesday.

% QUERY: When might Bob go to the opera?
```

```
{ opera(D | {D \= monday,D \= tuesday}),  not home(D | {D \= monday,D \= tuesday}),
not baby(Var1 | {Var1 \= tuesday}),  baby(tuesday),  not opera(tuesday),  home(tuesday) }
```

Background: s(CASP)

- s(CASP), based on stable model semantics, supports non-stratified negation.
- With s(CASP) we can define different worlds (models).
  - E.g., on saturday, Bob either goes to the opera or stays home.

```
opera(saturday) :- not home(saturday).
home(saturday) :- not opera(saturday).
```

```
%% Model 1
{ opera(saturday) }
```

```
%% Model 2
{ home(saturday) }
```

- With a more complex example... ...we can analyse the justification.

```
JUSTIFICATION_TREE:
Bob goes to the opera on D not equal monday, nor tuesday, because
    there is no evidence that Bob stays home on D not equal monday, nor tuesday, because
        it is assumed that Bob goes to the opera on D not equal monday, nor tuesday.
The global constraints hold, because
    the global constraint number 1 holds, because
        there is no evidence that they came with the babe on Var1 not equal tuesday, and
        they came with the babe on tuesday, and
        there is no evidence that Bob goes to the opera on tuesday, because
            Bob stays home on tuesday, because
                it is assumed that there is no evidence that Bob goes to the opera on tuesday.
```

Background: Applications of s(CASP)

Modelling and Reasoning in Event Calculus using s(CASP): [1]

- *Commonsense Reasoning* requires the ability to model continuous characteristics.
- *Event Calculus* represents continuous change and capture the law of inertia.

XAI of a Physician Advisor System (PAS): [7; 2]

- It recommends treatment choices for chronic heart failure (CHF).
  - Management of chronic diseases (CHF and others): major problem in health care.
  - The system encodes near 80 pages of rules into an ASP program.

Knowledge-driven Natural Language Understanding of English Text: [4]

- Presents two natural language understanding systems.
  - SQuARE: Semantic-based Question Answering and Reasoning Engine.
  - StaCACK: Stateful Conversational Agent using Commonsense Knowledge.
- They “truly understand” the natural language text they process:
  - Provide natural language explanations for their responses.

s(LAW): Administrative and judicial discretion reasoner

This work makes two main contributions:

- Set of patterns to translate legal rules into ASP.
  - Natural language patterns to generate readable justifications.
- Framework to model, reason, and justify conclusions based on:
  - The evidence provided by the user.
  - The applicable legislation.

s(LAW): Administrative and judicial discretion reasoner

This work makes two main contributions:

- Set of patterns to translate legal rules into ASP.
  - Natural language patterns to generate readable justifications.
- Framework to model, reason, and justify conclusions based on:
  - The evidence provided by the user.
  - The applicable legislation.

Use case: Procedure for awarding school places in the “Comunidad de Madrid” (CM):

- Representing ambiguity, discretion or incomplete information (key concepts in legal cases).

s(LAW): Administrative and judicial discretion reasoner

This work makes two main contributions:

- Set of patterns to translate legal rules into ASP.
  - Natural language patterns to generate readable justifications.
- Framework to model, reason, and justify conclusions based on:
  - The evidence provided by the user.
  - The applicable legislation.

Use case: Procedure for awarding school places in the "Comunidad de Madrid" (CM):

- Representing ambiguity, discretion or incomplete information (key concepts in legal cases).

Related Work:

- *Human-Understandable* explanation for AI advice:
  - Not possible for ML-based systems.
- *Current ASP explanation frameworks* [10; 5; 12]:
  - Only support grounded programs,
  - ... or do not justify negated literals,
  - ... and, do not support constraints and/or dense domains.

s(LAW): Patterns to translate law into ASP

Requirement For Applying

- Disjunction — s/he obtains a school place if one of the following common requirements are met
- Conjunction — In addition, some of the specific requirements must be met

Exceptions For Applying — Students coming from non-bilingual schools, need to accredit B1 level of English

Ambiguity — In case of *force majeure*, students may be reassigned to a school from another district

This pattern generates two models:

- One where `force_majeure` is assumed to hold.
- Another model where there is **no** evidence that `force_majeure` holds.

Discretion To Act — The School Council may add another complementary criterion

Unknown Information — It may be unclear whether the documents we have are valid or not

Handles the absence of information:

- `evidence/1`: States that some information is certain, e.g., `evidence(large_family)`.
- `-evidence/1`: The *strong* negation (`-`) is used to specify that we have evidences supporting the falsehood of some information, e.g., `-evidence(large_family)`.

s(LAW): The framework

ArticleESO.pl

- Contains the legislation rules in Fig. 1 following the patterns described.

ArticleESO.pre.pl

- Contains the natural language patterns to provide readable justifications.
- The directive `#pred` defines the natural language patterns, e.g.:

```
#pred obtain_place :: 's/he may obtain a school place'.
```

- Additionally, we can obtain a readable code in NL by invoking `scasp --code --human`.

StudentXX.pl

- Last module in Fig. 2 encodes the evidences of a student and links the previous modules.
- The code `XX` corresponds to the 'id' of each student (from 01 to 06).
- Table 1 shows the data corresponding to the candidates and the conclusion generated by s(LAW) for the query `?- obtain_place`.
  - Students 01, 03, 04, and 05 obtain a place at the school while students 02 and 06 do not.

s(LAW): The framework - ArticleESO.pl

```
%% Obtain a school place if...
obtain_place :-
    met_requirement,
    not exception.
met_requirement :-
    met_common_requirement,
    met_specific_requirement.
%% Common requirements:
met_common_requirement :-
    large_family.

met_common_requirement :-
    recipient_social_benefits.
recipient_social_benefits :-
    renta_minima_insercion.
recipient_social_benefits :-
    ingreso_minimo_vital.

met_common_requirement :-
    disability_status.
disability_status :-
    disabled_parent.
disability_status :-
    disabled_sibling.
%%  Specific requirements:
met_specific_requirement :-
    sibling_enroll_center.
met_specific_requirement :-
    legal_guardian_work_center.
met_specific_requirement :-
    relative_former_student.
met_specific_requirement :-
    school_proximity.
school_proximity :-
    same_education_district.
school_proximity :-
    not same_education_district,
    force_majeure.    % Ambiguity

force_majeure :-
    not n_force_majeure.
n_force_majeure :-
    not force_majeure.

%% Exceptions:
exception :-
    come_non_bilingual,
    want_bilingual_section(Course),
    not accredit_english_level(Course).

accredit_english_level('1st ESO') :-
    b1_certificate.
accredit_english_level('2nd ESO') :-
    b1_certificate.
accredit_english_level('3rd ESO') :-
    b2_certificate.
accredit_english_level('4th ESO') :-
    b2_certificate.
%% Discretion To Act:
obtain_place :-
    not met_requirement,
    met_complementary_criterion(CC).
obtain_place :-
    met_requirement, exception,
    met_complementary_criterion(CC).

met_complementary_criterion(CC) :-
    school_criteria(CC),
    purpose(CC), not unlawful(CC),
    not n_met_complementary_criterion(CC).
n_met_complementary_criterion(CC) :-
    not met_complementary_criterion(CC).

purpose(CC) :- promote_diversity(CC).
unlawful(CC) :- sex_discrimination(CC).
unlawful(CC) :- race_discrimination(CC).
unlawful(CC) :- religion_discrimination(CC).

school_criteria(foreign_student) :-
    foreign_student.
school_criteria(specific_etnia) :-
    specific_etnia.

promote_diversity(foreign_student).
promote_diversity(specific_etnia).
race_discrimination(specific_etnia).
```

Figure 1: Translation of the procedure for awarding school places under s(LAW).

s(LAW): The framework - Student01.pl

```
1  #include('ArticleESO.pl').
2  #include('ArticleESO.pred.pl').
3
4  come_non_bilingual.
5  want_bilingual_section('2nd ESO').
6
7  evidence(large_family).
8  evidence(renta_minima_insercion).
9  evidence(sibling_enroll_center).
10 evidence(same_education_district).
11 evidence(b1_certificate).
12 -evidence(foreign_student).
13 -evidence(specific_etnia).
```

Figure 2: Encoding corresponding to student 01.

Evaluation: Reasoning and Deduction with Real Use-Cases

Table 1: Case of different students evaluated using s(LAW).
Note: '+' is a positive evidence, '−' is a negative evidence, '?' means unkown.

	Student01	Student02	Student03	Student04	Student05	Student06
large_family	+	+	+	−	−	−
renta_minima_insercion	+	+	+	?	−	−
sibling_enroll_center	+	+	−	+	−	−
same_education_district	+	+	−	+	−	−
b1_certificate	+	−	+	?	−	−
foreign_student	−	−	−	−	+	−
specific_etnia	−	−	−	−	−	+
?- obtain_place	**yes**	no	**yes**	**yes**	**yes**	no

Evaluation: A Priori Deduction

- The student 01 meets common and specific requirements and avoids the exception (having level b1 in English). Therefore, s(LAW) returns the partial model:

```
{ obtain_place, large_family, sibling_enroll_center, come_non_bilingual,
  want_bilingual_section(2nd ESO), b1_certificate }
```

Evaluation: A Priori Deduction

- The student 01 meets common and specific requirements and avoids the exception (having level b1 in English). Therefore, s(LAW) returns the partial model:

```
{ obtain_place, large_family, sibling_enroll_center, come_non_bilingual,
  want_bilingual_section(2nd ESO), b1_certificate }
```

... and the corresponding justification in NL.

```
1  s/he may obtain a school place, because
2      a common requirement is met, because
3          s/he is part of a large family.
4      a specific requirement is met, because
5          s/he has siblings enrolled in the center.
6      there is no evidence that an exception applies, because
7          s/he came from a non-bilingual public school, and
8          s/he wish to study 2nd ESO in the Bilingual Section, and
9          s/he accredit required level of English for 2nd ESO, because
10             in the four skills certificate level b1.
```

Figure 3: Justification in Natural Language for the evaluation of `student01.pl`.

Evaluation: A Posteriori Deduction

- The query `?- not force_majeure, obtain_place` avoids the assumption of force majeure.
  - Under this assumption the student 03 does not obtain a place.
- The query `?- not obtain_place` explains why a student does not obtain a place.

Evaluation: A Posteriori Deduction

- The query `?- not force_majeure, obtain_place` avoids the assumption of force majeure.
  - Under this assumption the student 03 does not obtain a place.
- The query `?- not obtain_place` explains why a student does not obtain a place.

```
there is no evidence that s/he may obtain a school place, because
    there is no evidence that a common requirement is met, because
        there is no evidence that s/he is part of a large family, and
        there is no evidence that s/he is a recipient of the RMI, and
        there is no evidence that a parent or sibling has disability status.
    there is no evidence that the criterion foreign_student is met, because
        there is no evidence that s/he meets the criteria foreign_student, because
            there is no evidence that s/he is a foreign student.
    there is no evidence that the criterion specific_etnia is met, because
        s/he meets the criteria specific_etnia, because
            s/he belongs to a specific etnia.
        specific_etnia follows the purpose of the procedure, because
            specific_etnia promotes the diversity.
        specific_etnia is illegal, because
            specific_etnia discriminates based on race.
```

Figure 4: Justification in Natural Language for the evaluation of `student06.pl`.

Conclusions

- Using goal-directed ASP, s(LAW) is capable of modeling discretion and ambiguity.
  - Exhibits the property of modelling vague concepts.
- The deduction based on s(LAW) allows:
  - The consideration of different conclusions (multiple models):
    - which can be analyzed by humans thanks to the justification generated in natural language.
  - The reasoning about the set of these conclusions/models.

Conclusions

- Using goal-directed ASP, s(LAW) is capable of modeling discretion and ambiguity.
  - Exhibits the property of modelling vague concepts.
- The deduction based on s(LAW) allows:
  - The consideration of different conclusions (multiple models):
    - which can be analyzed by humans thanks to the justification generated in natural language.
  - The reasoning about the set of these conclusions/models.

Future work

- Complete the modeling of the legislation by tabulation for each of the criteria.
- Exploit the underlying constraint solver of s(CASP) to check the tabulation:
  - ...considering ambiguity, administrative discretion and unknown information.
- Analyze “Epistemic Specifications” [9]:
  - ...what is true in all/some models, which partial models share assumptions, etc.

Conclusions

- Using goal-directed ASP, s(LAW) is capable of modeling discretion and ambiguity.
  - Exhibits the property of modelling vague concepts.
- The deduction based on s(LAW) allows:
  - The consideration of different conclusions (multiple models):
    - which can be analyzed by humans thanks to the justification generated in natural language.
  - The reasoning about the set of these conclusions/models.

Future work

- Complete the modeling of the legislation by tabulation for each of the criteria.
- Exploit the underlying constraint solver of s(CASP) to check the tabulation:
  - ...considering ambiguity, administrative discretion and unknown information.
- Analyze “Epistemic Specifications” [9]:
  - ...what is true in all/some models, which partial models share assumptions, etc.

THANKS!

Bibliography I

[1] Arias, J., Carro, M., Chen, Z., and Gupta, G. (2019). Constraint Answer Set Programming without Grounding and its Applications. In Alviano, M. and Pieris, A., editors, *3rd Int'l. Workshop on the Resurgence of Datalog in Academia and Industry (Datalog 2.0)*, volume 2368, pages 22–26. CEUR-WS.

[2] Arias, J., Carro, M., Chen, Z., and Gupta, G. (2020). Justifications for goal-directed constraint answer set programming. In *Proceedings 36th International Conference on Logic Programming (Technical Communications)*, volume 325 of *EPTCS*, pages 59–72. Open Publishing Association.

[3] Arias, J., Carro, M., Salazar, E., Marple, K., and Gupta, G. (2018). Constraint Answer Set Programming without Grounding. *Theory and Practice of Logic Programming*, 18(3-4):337–354.

[4] Basu, K., Varanasi, S., Shakerin, F., Arias, J., and Gupta, G. (2021). Knowledge-driven natural language understanding of english text and its applications. *AAAI'21*.

[5] Cabalar, P., Fandinno, J., and Fink, M. (2014). Causal Graph Justifications of Logic Programs. *Theory and Practice of Logic Programming*, 14(4-5):603–618.

[6] Cerrillo i Martínez, A. (2019). El derecho para una inteligencia artificial centrada en el ser humano y al servicio de las instituciones: Presentación del monográfico. *IDP: Revista de Internet, Derecho y Politica*, (30).

Bibliography II

[7] Chen, Z., Marple, K., Salazar, E., Gupta, G., and Tamil, L. (2016). A Physician Advisory System for Chronic Heart Failure Management Based on Knowledge Patterns. *Theory and Practice of Logic Programming*, 16(5-6):604–618.

[8] Cobbe, J. (2019). Administrative law and the machines of government: judicial review of automated public-sector decision-making. *Legal Studies*, 39(4):636–655.

[9] Gelfond, M. (1994). Logic programming and reasoning with incomplete information. *Annals of mathematics and artificial intelligence*, 12(1):89–116.

[10] Pontelli, E., Son, T. C., and El-Khatib, O. (2009). Justifications for Logic Programs under Answer Set Semantics. *Theory and Practice of Logic Programming*, 9(1):1–56.

[11] Ramakrishna, S., Górski, Ł., and Paschke, A. (2016). A dialogue between a lawyer and computer scientist: the evaluation of knowledge transformation from legal text to computer-readable format. *Applied Artificial Intelligence*, 30(3):216–232.

[12] Schulz, C. and Toni, F. (2016). Justifying Answer Sets Using Argumentation. *Theory and Practice of Logic Programming*, 16(1):59–110.

[13] Sergot, M. J., Sadri, F., Kowalski, R. A., Kriwaczek, F., Hammond, P., and Cory, H. T. (1986). The british nationality act as a logic program. *Communications of the ACM*, 29(5):370–386.

[14] Solé, J. P. (2019). Inteligencia artificial, derecho administrativo y reserva de humanidad: algoritmos y procedimiento administrativo debido tecnológico. *Revista general de Derecho administrativo*, 50.