Codifying Logical Fragments in ASP - A General Knowledge Base Approach

Dominic Deckert

4. Februar 2021

Motivation

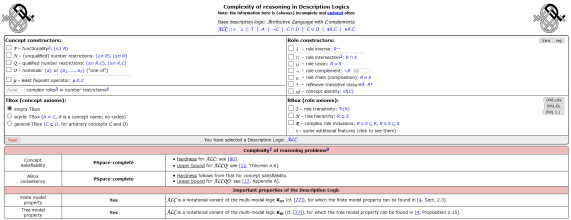
The fact that all Mathematics is Symbolic Logic is one of the greatest discoveries of our age; and when this fact has been established, the remainder of the principles of mathematics consists in the analysis of Symbolic Logic itself.

Bertrand Russell, "The Principles of Mathematics"

Overview

Previous Works

- DL database by Evgeny Zolin (2005 - 2013)
- DL database by Mohamed Ibrahim (2019)



Why ASP?

- **A**nswer **S**et **P**rogramming
- semantics based on Gelfond-Lifschitz reduct
  $\rightarrow$ calculates all answer sets of a given program
- Advantages of ASP:
  - allows for arbitrary arity of predicates
  - analytical tools
  - feasibility of calculations
  - tests for ambiguous rules

Aims of Our Approach

- model different logic fragments and DLs in a *unified manner*
- model connections between logic fragments of different "families"
- *infer* all possible complexity results
- *query* the knowledge base
- find *explanations* for inferred results

Aims of Our Approach

- model different logic fragments and DLs in a *unified manner*
- model connections between logic fragments of different "families"
- *infer* all possible complexity results
- *query* the knowledge base
- find *explanations* for inferred results

Aims of Our Approach

- model different logic fragments and DLs in a *unified manner*
- model connections between logic fragments of different "families"
- *infer* all possible complexity results
- *query* the knowledge base
- find *explanations* for inferred results

Aims of Our Approach

- model different logic fragments and DLs in a *unified manner*
- model connections between logic fragments of different "families"
- *infer* all possible complexity results
- *query* the knowledge base
- find *explanations* for inferred results

Aims of Our Approach

- model different logic fragments and DLs in a *unified manner*
- model connections between logic fragments of different "families"
- *infer* all possible complexity results
- *query* the knowledge base
- find *explanations* for inferred results

Families of Fragments

- *families* of logic fragments utilize the same basic semantics (e.g. DLs, FO, modal logic, ...)
- fragments are described by their traits (e.g. functionality, qualified number restriction)
  $\Rightarrow$ traits may subsume other traits
- traits are classified in restrictions and features of the fragment (relative to chosen base fragment)

Families of Fragments

- *families* of logic fragments utilize the same basic semantics (e.g. DLs, FO, modal logic, ...)
- fragments are described by their traits (e.g. functionality, qualified number restriction)
  $\Rightarrow$ traits may subsume other traits
- traits are classified in restrictions and features of the fragment (relative to chosen base fragment)

Families of Fragments

- *families* of logic fragments utilize the same basic semantics (e.g. DLs, FO, modal logic, ...)
- fragments are described by their traits (e.g. functionality, qualified number restriction)
  $\Rightarrow$ traits may subsume other traits
- traits are classified in restrictions and features of the fragment (relative to chosen base fragment)

Ex. Traits

	Features	Restrictions
DL	number restriction nominals inverse roles ...	horn logic no negation empty TBox ...
FOL	equality symbol	2 – variable fragment quantifier prefixes ...

Expressivity of Logic Fragments

- some fragments are more expressive than others
  i.e. all satisfiability problems in one fragment can be expressed in the other
- for fragments within the same family, this can be easy to determine:
  the fragment with more *features* and less *restrictions* is the more expressive
- expressivity is also informed by translation functions between different fragments

Expressivity of Logic Fragments

- some fragments are more expressive than others
  i.e. all satisfiability problems in one fragment can be expressed in the other
- for fragments within the same family, this can be easy to determine:
  the fragment with more *features* and less *restrictions* is the more expressive
- expressivity is also informed by translation functions between different fragments

Expressivity of Logic Fragments

- some fragments are more expressive than others
  i.e. all satisfiability problems in one fragment can be expressed in the other
- for fragments within the same family, this can be easy to determine:
  the fragment with more *features* and less *restrictions* is the more expressive
- expressivity is also informed by translation functions between different fragments

Ex. Fragments

family(dl; fol).

fragment(fo2, fol).
has_trait(fo2, two_variable).
trait(fo2, restriction, fol).

fragment(alc_EC, dl).
has_trait(alc_EC, (empty_TBox; concept_SAT)).
trait(concept_SAT, restriction, dl).

Ex. Fragments

family(dl; fol).

fragment(fo2, fol).
has_trait(fo2, two_variable).
trait(fo2, restriction, fol).

fragment(alc_EC, dl).
has_trait(alc_EC, (empty_TBox; concept_SAT)).
trait(concept_SAT, restriction, dl).

Ex. Fragments

family(dl; fol).

fragment(fo2, fol).
has_trait(fo2, two_variable).
trait(fo2, restriction, fol).

fragment(alc_EC, dl).
has_trait(alc_EC, (empty_TBox; concept_SAT)).
trait(concept_SAT, restriction, dl).

Ex. Fragments

translation(alc_EC, fo2).

translation(X, Z) :- translation(X, Y), translation(Y, Z).

has_extra_features(L_1, L_2):- has_trait(L_1,A), -has_trait(L_2, A), trait(A, feature, F), fragment(L_1, F), fragment(L_2, F).

higher_logic(X, Y) :- has_extra_features(X, Y), not has_extra_features(Y, X), not has_extra_restrictions(X, Y).

Ex. Fragments

translation(alc_EC, fo2).

translation(X, Z) :- translation(X, Y), translation(Y, Z).

has_extra_features(L_1, L_2):- has_trait(L_1,A), -has_trait(L_2, A), trait(A, feature, F), fragment(L_1, F), fragment(L_2, F).

higher_logic(X, Y) :- has_extra_features(X, Y), not has_extra_features(Y, X), not has_extra_restrictions(X, Y).

Ex. Fragments

translation(alc_EC, fo2).

translation(X, Z) :- translation(X, Y), translation(Y, Z).

has_extra_features(L_1, L_2):- has_trait(L_1,A), -has_trait(L_2, A), trait(A, feature, F), fragment(L_1, F), fragment(L_2, F).

higher_logic(X, Y) :- has_extra_features(X, Y), not has_extra_features(Y, X), not has_extra_restrictions(X, Y).

Ex. Fragments

translation(alc_EC, fo2).

translation(X, Z) :- translation(X, Y), translation(Y, Z).

has_extra_features(L_1, L_2):- has_trait(L_1,A), -has_trait(L_2, A), trait(A, feature, F), fragment(L_1, F), fragment(L_2, F).

higher_logic(X, Y) :- has_extra_features(X, Y), not has_extra_features(Y, X), not has_extra_restrictions(X, Y).

Model Properties

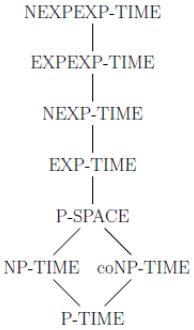
- fragments can also have certain model properties (e.g. finite model property)
- expressivity can propagate model properties
  has_model_property(L_2, P, S) :- has_model_property(L_1, P, S), higher_logic(L_1, L_2).

Model Properties

- fragments can also have certain model properties (e.g. finite model property)
- expressivity can propagate model properties
  has_model_property(L_2*, P, S) :- has_model_property(*L_1*, P, S), higher_logic(*L_1*,* L_2*).*

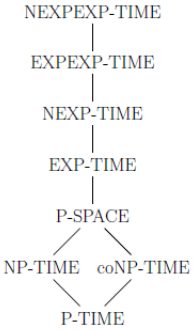
Complexity Classes

- Complexity Classes: sets of problems with the same asymptotic complexity
- inclusion ordering hierarchy between complexity classes
- *smaller_complexity(X,X):- complexity_class(X).*
  :- smaller_complexity(X,Y), smaller_complexity(Y,X).
  smaller_complexity(X,Z):- smaller_complexity(X,Y), smaller_complexity(Y,Z).
- equality results can be incorporated



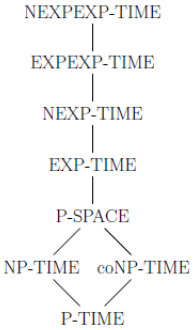
Complexity Classes

- Complexity Classes: sets of problems with the same asymptotic complexity
- inclusion ordering hierarchy between complexity classes
- *smaller_complexity(X,X):- complexity_class(X).*
  :- smaller_complexity(X,Y), smaller_complexity(Y,X).
  smaller_complexity(X,Z):- smaller_complexity(X,Y), smaller_complexity(Y,Z).
- equality results can be incorporated



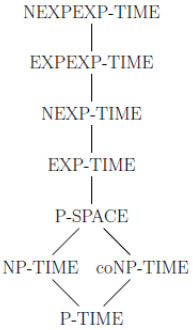
Complexity Classes

- Complexity Classes: sets of problems with the same asymptotic complexity
- inclusion ordering hierarchy between complexity classes
- *smaller_complexity(X,X):- complexity_class(X).*
  :- smaller_complexity(X,Y), smaller_complexity(Y,X).
  smaller_complexity(X,Z):- smaller_complexity(X,Y), smaller_complexity(Y,Z).
- equality results can be incorporated



Complexity Classes

- Complexity Classes: sets of problems with the same asymptotic complexity
- inclusion ordering hierarchy between complexity classes
- *smaller_complexity(X,X):- complexity_class(X).*
  :- smaller_complexity(X,Y), smaller_complexity(Y,X).
  smaller_complexity(X,Z):- smaller_complexity(X,Y), smaller_complexity(Y,Z).
- equality results can be incorporated



Propagating Complexity Results

- some complexity results are known for logic fragments (depending on fragment and semantic)
- furthermore, expressivity propagates complexity results:
  - Hardness: more expressive fragments contain the hard problems of less expressive fragments
    is_hard$(L_1, C, S) \coloneq$ *is_hard* (L_2, C, S), *higher_logic*(L_1, L_2).
  - Inclusion: less expressive fragments can be solved with the same resources as more expressive fragments
    has_complexity $(L_2, C, S) \coloneq$ *has_complexity* (L_1, C, S), *higher_logic*(L_1, L_2), C *!= undecidable*.
- program can identify most meaningful complexity results

Propagating Complexity Results

- some complexity results are known for logic fragments (depending on fragment and semantic)
- furthermore, expressivity propagates complexity results:
  - Hardness: more expressive fragments contain the hard problems of less expressive fragments
    is_hard(L_1, C, S) :- is_hard (L_2, C, S), higher_logic(L_1, L_2).
  - Inclusion: less expressive fragments can be solved with the same resources as more expressive fragments
    has_complexity (L_2, C, S) :- has_complexity (L_1, C, S), higher_logic(L_1, L_2), C != undecidable.
- program can identify most meaningful complexity results

Propagating Complexity Results

- some complexity results are known for logic fragments (depending on fragment and semantic)
- furthermore, expressivity propagates complexity results:
  - Hardness: more expressive fragments contain the hard problems of less expressive fragments
    is_hard(L_1*, C, S) :- is_hard (*L_2*, C, S), higher_logic(*L_1*,* L_2*).*
  - Inclusion: less expressive fragments can be solved with the same resources as more expressive fragments
    has_complexity (L_2*, C, S) :- has_complexity (*L_1*, C, S), higher_logic(*L_1*,* L_2*), C != undecidable.*
- program can identify most meaningful complexity results

Propagating Complexity Results

- some complexity results are known for logic fragments (depending on fragment and semantic)
- furthermore, expressivity propagates complexity results:
  - Hardness: more expressive fragments contain the hard problems of less expressive fragments
    is_hard(L_1*, C, S) :- is_hard (*L_2*, C, S), higher_logic(*L_1*,* L_2*).*
  - Inclusion: less expressive fragments can be solved with the same resources as more expressive fragments
    has_complexity (L_2*, C, S) :- has_complexity (*L_1*, C, S), higher_logic(*L_1*,* L_2*), C != undecidable.*
- program can identify most meaningful complexity results

Working with the Knowledge Base

- Querying for information:
  facts can be retrieved from the knowledge base by filtering the inferred results (using the `target_logic` in query.lp)
- additions to the knowledge base:
  new facts can be added to the knowledge base using a python program

Working with the Knowledge Base

- Querying for information:
  facts can be retrieved from the knowledge base by filtering the inferred results (using the `target_logic` in query.lp)
- additions to the knowledge base:
  new facts can be added to the knowledge base using a python program

Demonstration

Demonstration

Finding Justifications

- Justifications: minimal sets of facts that satisfy a given consequence
- useful metric for debugging and analysis of a query result
- Calculation:
  - single justification can be calculated by successive removing of extraneous facts
  - all justifications can be obtained by modifying the program and using asprin (very costly calculation)

Finding Justifications

- Justifications: minimal sets of facts that satisfy a given consequence
- useful metric for debugging and analysis of a query result
- Calculation:
  - single justification can be calculated by successive removing of extraneous facts
  - all justifications can be obtained by modifying the program and using asprin (very costly calculation)

Finding Justifications

- Justifications: minimal sets of facts that satisfy a given consequence
- useful metric for debugging and analysis of a query result
- Calculation:
  - single justification can be calculated by successive removing of extraneous facts
  - all justifications can be obtained by modifying the program and using asprin (very costly calculation)

Conclusion

We

- have modelled different logic families in the same framework
- can impose an ordering of the expressivity on the different fragments
- can infer “new” results using this hierarchy
- can extend the knowledge base with new results
- can find a justification to provide some explanation for a result

What can still be done?

- from proof of concept to well-filled knowledge base
- graphical interface
- more general approach for justifications