Automating Common Sense Reasoning

Gopal Gupta, Elmer Salazar, Sarat Chandra Varanasi, Kinjal Basu, Farhad Shakerin, Fang Li and Huaduo Wang

The University of Texas at Dallas

Joaquín Arias

Universidad Rey Juan Carlos

Acknowledgement: NSF, DARPA, Amazon, Atos, EIT Digital, MICINN

Overview

- **What?** : formalize human thought process (automate commonsense reasoning)
- **Why?** : If we can formalize the human thought process, then ***everything*** can be automated (medical treatment, self-driving cars, automated s/w certification)
- **How?** : With ASP, and its goal-directed implementation s(CASP)
- Humans are sophisticated & effective thinkers: operate with just 12 watts of power:
  - and they can perform deductive, abductive and inductive reasoning
- Classical logics are limited: can perform reasoning for only well-founded or inductively constructed objects
- Humans employ both inductive (well-founded) and co-inductive (circular or assumption-based) reasoning; they also draw conclusions from failure of proofs (NAF)
- Negation-as-failure (NAF): a key concept for emulating the human thought process
- s(CASP): a goal-directed implementation of ASP that supports inductive, coinductive and abductive reasoning: crucial for automating commonsense reasoning

Formalizing Human Thought

- Formalizing the human thought process has been considered hard
  - History: Syllogisms, Boolean logic, Predicate Logic, ……. Other advanced logics
  - These logics have not been able to model the sophistication/effectiveness of human reasoning …
- Early problems of naïve set theory and predicate logic (Russell's paradox) led mathematicians and logicians to focus only on inductive sets & reasoning
  - ….. which is insufficient to model commonsense reasoning
- Classical systems of logic cannot reason about themselves (Tarski)
  - A logic cannot have its own truth predicate: need meta logic, meta meta logic, *ad infinitum*
  - Classical logic cannot reason about its proof failure, for instance
  - Kripke 1975 showed that a language can have its own truth predicate *and* consistency
  - Led to idea of co-induction and modeling of circular reasoning (that humans do employ)

Formalizing Human Thought

- Reason for automating commonsense reasoning: Human abilities are limited:
  - Humans can handle only 4 variables at a time (Halford et al, *Psych. Sci.*, 2005)

    "If the number of variables to be considered exceeds human processing capacity, then the worker will drop his or her mental bundle and become unable to proceed."

    "worker may revert to a simplified version of the task that does not take all aspects into account and therefore may make the wrong decision."
- Individual statements easy to comprehend; jointly they become hard to understand

  Paul will go to Mexico **if** Sally will not go to Mexico
  Sally will go to Mexico **if** Rob will not go to Mexico.
  Rob will go to Mexico **if** Paul will not go to Mexico.
  Rob will go to Mexico **if** Sally will not go to Mexico.

- Who will go to Mexico?
- Formalization/Automation is hence important;
- We accomplish it via *answer set programming* and the s(CASP) system

Common Sense Reasoning (CSR)

- Standard Logic Prog. fails at performing human-style commonsense reasoning
- In fact, most formalisms have failed; problem: monotonicity of classical logic
- Commonsense reasoning requires:
  1. Non-monotonicity: the system can revise its earlier conclusion in light of new information (contradictory information discovered later does not break down things as in classical logic)
     - We work with the knowledge we have; be ready to revise a conclusion if new knowledge appears
     - If Tweety is a bird, it can fly; conclusion to be retracted if Tweety is found to be a penguin
  2. Draw conclusions from absence of information:
     - Can't tell if it is raining outside. If I see no one holding an umbrella, so it must not be raining
  3. Global constraints: Not pursue reasoning violating a global constraint + invariants must hold
     - We know that it's impossible to walk and sit at the same time; a human must breathe to stay alive
  4. Cyclical or assumption-based reasoning: allow multiple worlds (non-inductive semantics)
     - Fish can talk in the cartoon world but not in the real world (two possible worlds)
- Commonsense Reasoning requires: *negation as failure & cyclical reasoning*

Classical Negation vs Negation as Failure

- Classical negation (CN)
  - represented as -p
  - e.g, -sibling(john, jim) %states that John and Jim are not siblings
  - An explicit proof of falsehood of predicate p is needed
  - -murdered(oj, nicole) holds true only if there is an explicit proof of OJ's innocence (seen in Boston airport, Nicole's body was found in LA)
- Negation as failure (NAF)
  - represented as not(p)
  - e.g., not sibling(john, jim) %states that *no evidence* John and Jim are siblings
  - We try to prove a proposition p, if we fail, we conclude not(p) is true
  - No evidence of p then conclude not(p)
  - not(murdered(oj, nicole)) holds true if we fail to find any proof that OJ killed Nicole

FAIL TO PROVE (NAF) vs **EXPLICITLY PROVING FALSEHOOD (CN)**

Failure of Classical Reasoning Methods for CSR - I

- Given a set of axioms A_1, A_2, …, A_n, a decision procedure based on classical logic gives us a method for proving a theorem T.
  - $A_1, A_2, \ldots, A_n \models T$
- What if the proof of T fails (i.e., we get stuck & cannot progress)?
  - Classical logic-based decision procedures do not give us any insight when a proof fails
- In many circumstances we may be able to conclude that the proof of T is not possible and so $\neg$T should hold.
- Most of commonsense reasoning is of this form:
  - If we *fail to prove* T (now), then T must be false (though T may become true later)
  - Dual also true: if we *prove* T (now), then T is true (though it may become false later)

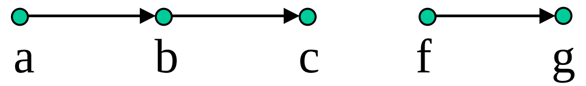
Failure of Classical Reasoning Methods for CSR - I

- Example: Transitive closure: edge(a, b). edge(b, c). edge(f, g).

  reach(X, Y) :- edge(X, Y).

  reach(X, Y) :- edge(X, Z), reach(Z, Y).

  ?- reach(a, f).



Query fails (no proof); Under classical theorem proving we can’t conclude that f is unreachable from node a.

- Need axioms for **un**reachability, only then we can conclude –reach(a, f).
  - That is, we have to explicitly define rules for –reach(X, Y).
- Failure is not of logic or the decision procedure, rather how we interpret the rules.
- Interpret implications as causal relations: A if B means A iff B

  reach(X, Y) $\leftrightarrow$ (edge(X, Y) $\vee$ (edge(X, Z), reach(Z, Y))).

- Now with NAF we can write: unreachable(X, Y) :- not reachable(X, Y).
- Note: when humans write “if A then B”, they mean “A iff B” most of the time;
  - E.g.: breaks_object :- drop_object we automatically mean not_breaks_object :- not_drop_object.

Failure of Classical Reasoning Methods for CSR - II

- Reliance of classical logic solely on inductive or well-founded semantics
- Frege discovers (naïve) set theory or predicate logic
- Russell finds a problem with Frege's formulation (Russell's paradox).
  - Problem is caused by circularity; Russell's solution: ban circularity
  - Russell: henceforth, every structure (including sets) must be inductive, i.e., start from the smallest element and then successively build larger elements from it;
  - anything non-inductive (cyclical or coinductive) is banished from discourse.
  - The whole mathematics/logic enterprise obsessed with only having inductive structures
  - Thus, any theory involving predicates must have an inductive semantics (single model)
  - However, circularity arises everywhere in human experience along with theories that have multiple models (possible worlds semantics).

jack_eats_food :- jill_eats_food.   jill_eats_food :- jack_eats_food.

Two possible worlds: both eat or none eat;   inductive semantics: none eat

Failure of Classical Reasoning Methods for CSR - II

- Non-inductive semantics also has to deal with incomplete information

  John will teach databases if no evidence Mary will teach databases
  Mary will teach databases if no evidence John will teach databases

- What is the meaning of these sentences?
- Humans can imagine two possibilities:
  - Possible world #1: John will teach databases and Mary will not
  - Possible world #2: Mary will teach databases and John will not
- Circular reasoning is assumption-based reasoning
  - If we assume that Mary will not teach databases, John will (and vice versa)
- Multiple possible worlds can be found if we stay within propositional logic
- Problem arises when we allow predicates and reason at the level of predicates
  - X will teach databases if no evidence Y will (where $X \neq Y$)
  - Y will teach databases if no evidence X will (where $X \neq Y$)

Answer Set Programming (ASP)

- Prolog extended with NAF; Rules of the form:

  p :- a_1,…, a_m, not b_1 ,…, not b_n.   m, n ≥0 (rule)

  p.   (fact)

- ASP is a popular formalism for non monotonic reasoning
- **Another reading:** add p to the answer set (model of the program) if a_1 ,…, a_m are in the answer set and b_1 ,…, b_n are not
- The rule could take more general form

  p :- a_1,…, a_m, not b_1 ,…, not b_n, -c_1, ..., -c_k, not -d_1 ,…, not -d_i m, n, k, i ≥0 (rule)

- Logic programs with NAF goals in the body are called normal logic programs
- Applications of ASP to KR&R, planning, constrained optimization, etc.
- Semantics: LFP of a residual program obtained after “Gelfond-Lifschitz” transform
- Popular implementations: Smodels, DLV, CLASP, etc.
- More than 30 years of great research by the ASP community including to CSR

Answer Set Programming

- Answer set programming (ASP)
  - Based on Possible Worlds and Stable Model Semantics;
  - Given an answer set program, find its models
  - Model: assignment of true/false value to propositions to make all formulas true. Models are called answer sets
  - Captures default reasoning, exceptions, preferences, constrained optimization, abductive reasoning, etc., in a natural way; does not cover cyclical reasoning with positive loops
  - Better way to build automated reasoning systems & expert systems (achieve AGI)
- Caveats
  - $p \Leftarrow a, b.$ really is taken to be $p \Leftrightarrow a, b.$ (rules are causal)
  - We are only interested in supported models: if p is in the answer set, it must be in the LHS of a 'successful' rule
    - The rule p :- q. ($q \Rightarrow p$) has a model in which q is false and p is true; such models are not interesting
    - When we write p :- q. we are stating that p is true if q is true (q being true supports p being true)

Slide from C. Baral (ASU)

ASP Example

- Consider the college admission process, modeled in ASP
  (1) eligible(X) :- highGPA(X), not ab_eligible(X).
  (2) eligible(X) :- special(X), fairGPA(X), not ab_eligible(X).
  (3) –eligible(X) :- –special(X), –highGPA(X), not ab_ineligible(X).
  (4) interview(X) :- not eligible(X), not –eligible(X).
  (5) fairGPA(john).
  (6) –highGPA(john)
- Since we have no information about John being special or ¬ special, both eligible(john) and ¬ eligible(john) fail.
- So John will have to be interviewed
- ASP gives us a hierarchy of (un)certainty that we humans employ, given some proposition p (e.g., p = it is raining now):
  - p is definitely true: p
  - p maybe true: not –p (possible p)
  - p unknown: not –p & not p
  - p maybe false: not p (no evidence of p)
  - p definitely false: -p

Generally, we humans do not employ probabilities in our day to day common sense reasoning

Current ASP Systems

- Very sophisticated and efficient ASP systems have been developed based on SAT solvers; :
  - CLASP/CLINGO, DLV, CModels
- These systems work as follows:
  - Ground the programs w.r.t. the constants present in the program
    - Grounding may be incremental
  - Transform the propositional answer set programs into propositional formulas and then find their models using a SAT solver
  - The models found are the stable models of the original program
- Because SAT solvers require formulas to be propositional, programs with only constants and variables are feasible (datalog)
- Thus, only propositional answer set programs can be executed

Current ASP Systems: Issues

- **Finite Groundability:** Program has to be finitely groundable
  - Not possible to have lists, structures, and complex data structures
  - Not possible to have arithmetic over reals
- **Exponential Blowup:** Grounding can result in exponential blowup
  - Given the clause: p(X, a) :- q(Y, b, c).
  - It turns into 3 x 3, i.e., 9 clauses

    p(a, a) :- q(a, b, c). p(a, a) :- q(b, b, c). p(a, a) :- q(c, b, c).
    p(b, a) :- q(a, b, c). p(b, a) :- q(b, b, c). p(b, a) :- q(c, b, c).
    p(c, a) :- q(a, b, c). p(c, a) :- q(b, b, c). p(c, a) :- q(c, b, c).
  - Imagine a large knowledgebase with1000 clauses with 100 variables and 100 constants;
  - Use of ASP severely limited to only solving combinatorial problems (not to KR problems)
  - Programmers have to contort themselves while writing ASP code
- **Only Datalog + NAF:** Programs cannot contain lists structures and complex data structures: result in infinite-sized grounded program

Current ASP Systems: Issues

- **Entire Model:** SAT solvers find the entire model of the program
  - Entire model may contain lot of unnecessary information
  - I want to know the path from Boston to NYC, the model will contain all possible paths from every city to every other city (overkill)
- **Answer Unidentifiable:** Sometimes it may not even be possible to find the answer sought, as they are hidden in the answer set
  - Answer set of the reachability problem
  - The edges that constitute the actual path cannot be identified
  - No query, so no justification
- **KB Inconsistency:** Minor inconsistency in the knowledge base will result in the system declaring that there are is no answer set
  - We want to compute an answer if it doesn't involve the inconsistent part of the KB
  - Impossible to have a large knowledge base that is 100% consistent

Solution: Goal-directed Execution

- Develop goal-directed answer set programming systems that support *predicates*
- Goal-directed means that a query is given, and a proof for the query found by exploring the program search space
- Essentially, we need Prolog style execution that supports stable model semantics-based negation
- Thus, part of the knowledge base that is only relevant to the query is explored
- Predicate answer set programs are directly executed without any grounding: lists and structures also supported
- Realized in s(ASP) system: ASP with predicates and first order terms
- The s(CASP) system: s(ASP) extended with CLP(Q) and many more things:
  - Sophisticated justification tree for the query
  - Abductive/cyclical reasoning even for positive loops
  - On demand execution of global constraints (denials)

Goal-directed execution of ASP

- Key concept for realizing goal-directed execution of ASP:
  - coinductive execution
- Coinduction: dual of induction
  - computes elements of the GFP
  - In the process of proving p, if p appears again, then p succeeds
- Given:

```
eats_food(jack) :- eats_food(jill).
eats_food(jill) :- eats_food(jack).
```

there are two possibilities:

- Both jack and jill eat (GFP)
- Neither one eats (LFP)

?- not eats_food(jack)

not eats_food(jill).

?- not eats_food(jack)

coinductive success

coinductive hypothesis set = {not eats_food(jack), not eats_food(jill)}

Goal-directed execution of ASP

- To incorporate negation in coinductive reasoning, we need a negative coinductive hypothesis rule:
  - In the process of establishing not(p), if not(p) is seen again in the resolvent, then not(p) succeeds **[co-SLDNF Resolution]**
- Also, **not not p** reduces to p.
- Even-loops succeed by coinduction

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

?- p → not q → not not p → p (coinductive success)

The coinductive hypothesis set is the answer set: {p, not q}

- For each constraint rule, we need to extend the query
  - Given a query Q for a program that contains a constraint rule

p :- q, not p. (q ought to be false)

extend the query to: ?- Q, (p ∨ not q)

Commonsense Reasoning

- Human thought process can be largely modeled with:
  - Default rules with exceptions and preferences (manage incomplete information)
  - Multiple possible worlds (cyclical reasoning or assumption-based reasoning)
  - Global constraints (impossibilities or invariants that must always hold)

flies(X) :- bird(X), not abnormal_bird(X).

abnormal_bird(X) :- penguin(X).

bird(tweety).   bird(sam).

Query: ?- flies(tweety) & ?- flies(sam) will succeed

Now add the fact: penguin(tweety).

Query: ?- flies(tweety) fails while ?- flies(sam) succeeds

(Default reasoning with exceptions)

teach_db(john) :- not teach_db(mary).
teach_db(mary) :- not teach_db(john).

(multiple possible worlds)

false :- teach_db(mary).

(constraints)

Defaults and Exceptions

Aggressive

```
flies(X) :- bird(X), not ab(X).
ab(X) :- penguin(X).
penguin(tweety).
bird(tweety).
bird(sam).

?- flies(tweety).   Ans: no
?- flies(sam).      Ans: yes
```

Conservative

```
flies(X) :- bird(X), not ab(X).
ab(X) :- not -penguin(X).
penguin(tweety).
bird(tweety).
bird(sam).

?- flies(tweety).   Ans: no
?- flies(sam).      Ans: no
```

Strong Exception

```
-flies(X) :- ostrich(X).

flies(X) :- bird(X), not ab(X),
            not –flies(X).
```

Nested Exceptions

```
–flies(X) :- animal(X), not ab_animal(X).
ab_animal(X) :- bird(X), not ab_bird(X).
ab_bird(X) :- penguin(X), not ab_penguin(X).
ab_penguin(X) :- superpenguin(X).
```

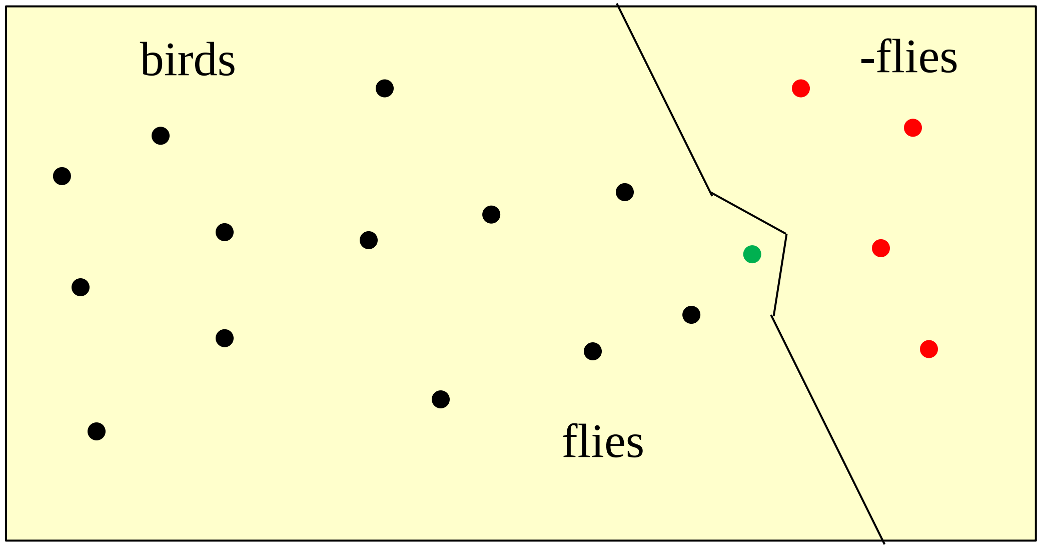
Why Default Theory?

- Defaults & exceptions are excellent for representing inductive generalizations
  - As we learn more by seeing more examples, we continually adjust the decision boundary in an elaboration tolerant manner
- We observe that many birds fly

  flies(X) :- bird(X).
- Aha, but penguins don’t

  flies(X) :- bird(X), not ab_bird(X).

  ab_bird(X) :- penguin(X).



- Aha, but super-penguins do fly

  flies(X) :- bird(X), not ab_bird(X).

  ab_bird(X) :- pengjuin(X), not ab_peng(X).

  ab_peng(X) :- superpenguin(X).
- Can serve as the basis for Machine Learning and Explainable AI

FOLD Algorithm

Learning goal: `flies(X) :- ?`

B:
```
bird(X) :- penguin(X).
bird(tweety).          bird(woody).
cat(kitty).            penguin(polly).
```

E^+: `flies(tweety).` `flies(woody).`

E^-: `flies(polly).` `flies(kitty).`

List of candidate literals

`bird(X), cat(X), penguin(X)`

Initially…

`flies(X) :- true.` E^+ = [tweety, woody] E^- = [polly, kitty]

FOLD Algorithm

Learning goal: `flies(X) :- ?`

B:
```
bird(X) :- penguin(X).
bird(tweety).          bird(woody).
cat(kitty).            penguin(polly).
```

E^+: `flies(tweety).` `flies(woody).`

E^-: `flies(polly).` `flies(kitty).`

List of candidate literals

`bird(X), cat(X), penguin(X)`

After first iteration…

`flies(X) :- cat(X).` E^+ = [] E^- = [kitty]

FOLD Algorithm

Learning goal: `flies(X) :- ?`

B:
```
bird(X) :- penguin(X).
bird(tweety).          bird(woody).
cat(kitty).            penguin(polly).
```

E^+: `flies(tweety).` `flies(woody).`

E^-: `flies(polly).` `flies(kitty).`

List of candidate literals

```
bird(X), cat(X), penguin(X)
```

After first iteration...

`flies(X) :- penguin(X).` E^+ = [] E^- = [polly]

FOLD Algorithm

Learning goal: `flies(X) :- ?`

B:
```
bird(X) :- penguin(X).
bird(tweety).            bird(woody).
cat(kitty).              penguin(polly).
```

E^+: `flies(tweety).` `flies(woody).`

E^-: `flies(polly).` `flies(kitty).`

List of candidate literals

```
bird(X), cat(X), penguin(X)
```

Information gain becomes 0, it's time to swap examples…

`flies(X) :- bird(X).` E^+ = [tweety,woody] E^- = [polly]

FOLD Algorithm

Learning goal: `flies(X) :- ?`

B:
```
bird(X) :- penguin(X).
bird(tweety).          bird(woody).
cat(kitty).            penguin(polly).
```

E^+: `flies(tweety).` `flies(woody).`

E^-: `flies(polly).` `flies(kitty).`

List of candidate literals

`bird(X), cat(X), penguin(X)`

Initially…

`ab(X) :- true.` E^+ = [polly] E^- = [tweety,woody]

FOLD Algorithm

Learning goal: `flies(X) :- ?`

B: `bird(X) :- penguin(X).`

```
bird(tweety).          bird(woody).
cat(kitty).            penguin(polly).
```

E^+: `flies(tweety).` `flies(woody).`

E^-: `flies(polly).` `flies(kitty).`

List of candidate literals

```
bird(X), cat(X), penguin(X)
```

After one iteration… return ab

`ab(X) :- penguin(X).` E^+ = [polly] E^- = []

FOLD Algorithm

Learning goal: `flies(X) :- ?`

B:
```
bird(X) :- penguin(X).
bird(tweety).        bird(woody).
cat(kitty).          penguin(polly).
```

E^+: `flies(tweety).` `flies(woody).`

E^-: `flies(polly).` `flies(kitty).`

Final hypothesis (rules)

```
flies(X) :- bird(X), not ab(X).
ab(X)    :- penguin(X).
```

FOLD Family of Algorithms

- Basis for the FOLD family of algorithms:
  - SHAP-FOLD & LIME-FOLD: Based on Shapley values/HUIM and LIME heuristics, resp.
  - FOLD-R++: Based on IG; Binary classification (available on Github); paper in FLOPS'22
  - FOLD-RM: Based on IG; Multi-category classification (available on GitHub); paper in ICLP'22
  - FOLD-SE: Based on GI; Classification with scalable explainability (not realeased yet)
- Salient features of FOLD-SE:
  - Based on Gini Impurity heuristics
  - Accepts mixed data, both numerical and categorical
  - No major data prep work needed (no one-hot encoding, etc)
  - Missing values allowed, no special action needed
  - Scalable Explainability: regardless of dataset size, small number of rules and literals in the rule-set
  - Same rule-set generated, regardless of the training-testing split
  - Accuracy comparable to XGBoost and MLPs, but explainable and order of magnitude faster

FOLD Family of Algorithms

- Adult Dataset: 32561 x 15; Accuracy: 0.84;
  - Generated model:

income(X,'<=50K') :- not marital_status(X,'Married-civ-spouse'),
capital_gain(X,N1), N1=<6849.0.

income(X,'<=50K') :- marital_status(X,'Married-civ-spouse'), capital_gain(X,N1), N1=<5013.0,
education_num(X,N2), N2=<12.0.

- Rain in Australia Dataset: 145460 x 24; Accuracy: 0.82
  - Generated model:

raintomorrow(X,'No') :- humidity3pm(X,N1), N1=<64.0, rainfall(X,N2), N2=<182.6.

raintomorrow(X,'No') :- rainfall(X,N2), N2=<2.2, humidity3pm(X,N1), not(N1=<64.0),
not(N1>81.0).

FOLD-SE: Scalable Explainability

Data Set			XGBoost					MLP					FOLD-SE				
Name	Rows	Cols	Acc	Prec	Rec	F1	T(ms)	Acc	Prec	Rec	F1	T(ms)	Acc	Prec	Rec	F1	T(ms)
acute	120	7	1.0	1.0	1.0	1.0	122	0.99	1.0	0.99	0.99	22	1.0	1.0	1.0	1.0	1
heart	270	14	0.82	0.83	0.85	0.83	247	0.76	0.79	0.79	0.78	95	0.74	0.77	0.78	0.77	13
ionosphere	351	35	0.88	0.87	0.95	0.91	2,206	0.79	0.91	0.74	0.81	1,771	0.91	0.89	0.98	0.93	119
voting	435	17	0.95	0.93	0.95	0.93	149	0.95	0.92	0.94	0.93	43	0.95	0.92	0.96	0.94	11
credit-a	690	16	0.85	0.86	0.86	0.86	720	0.82	0.84	0.84	0.84	356	0.85	0.92	0.79	0.85	36
breast-w	699	10	0.95	0.96	0.98	0.96	186	0.97	0.98	0.97	0.98	48	0.94	0.88	0.97	0.92	9
autism	704	18	0.97	0.98	0.98	0.98	236	0.96	0.99	0.96	0.97	56	0.91	0.94	0.94	0.94	29
parkinson	765	754	0.76	0.79	0.93	0.85	270,336	0.60	0.77	0.67	0.71	152,056	0.82	0.82	0.96	0.89	9,691
diabetes	768	9	0.66	0.71	0.81	0.76	839	0.66	0.73	0.74	0.73	368	0.75	0.78	0.85	0.81	38
cars	1728	7	1.0	1.0	1.0	1.0	210	0.99	1.0	1.0	1.0	83	0.96	1.0	0.94	0.97	20
kr vs. kp	3196	37	0.99	0.99	1.0	0.99	403	0.99	0.99	1.0	0.99	273	0.97	0.96	0.97	0.97	152
mushroom	8124	23	1.0	1.0	1.0	1.0	697	1.0	1.0	1.0	1.0	394	1.0	1.0	0.99	1.0	254
churn-model	10000	11	0.85	0.87	0.96	0.91	97,727	0.81	0.90	0.86	0.88	18,084	0.85	0.87	0.95	0.91	600
intention	12330	18	0.90	0.93	0.95	0.94	171,480	0.81	0.89	0.88	0.89	41,992	0.90	0.95	0.93	0.94	661
eeg	14980	15	0.64	0.64	0.81	0.71	46,472	0.69	0.72	0.71	0.71	9,001	0.67	0.74	0.63	0.68	1,227
credit card	30000	24	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0.82	0.83	0.96	0.89	3,513
adult	32561	15	0.87	0.89	0.95	0.92	424,686	0.81	0.88	0.87	0.87	300,380	0.84	0.86	0.95	0.90	1,746
rain in aus	145460	24	0.84	0.85	0.96	0.90	385,456	0.81	0.86	0.89	0.88	243,990	0.82	0.85	0.94	0.89	10,243
average[1]	12977	59	0.88	0.89	0.94	0.91	77,914	0.86	0.90	0.88	0.89	42,735	0.88	0.90	0.92	0.90	1,381

Example: Levesque's LP book

1. George is a bachelor.
2. George was born in Boston, collects stamps.
3. A son of someone is a child who is male.
4. George is the only son of Mary and Fred.
5. A man is an adult male person.
6. A bachelor is a man who has never been married.
7. A (traditional) marriage is a contract between a man and a woman, enacted by a wedding and dissolved by a divorce.
8. While the contract is in effect, the man (called the husband) and the woman (called the wife) are said to be married.
9. A wedding is a ceremony where . . . bride . . . groom . . . bouquet . . .

- Conclude that
  - George has never been the groom at a wedding.
  - Mary has an unmarried son born in Boston.
  - No woman is the wife of any of Fred's children.

```
% 1
bachelor(george).
% 2
birth_city(george, boston).
hobby(george, stamp_collecting).
% 3
son(X,Y):- child(X,Y), male(X).
child(X,Y):- son(X,Y).
male(X):- son(X,Y).
child(george,mary).
child(george,fred).
male(george).
% 4.
:- son(X, fred), son(Another_son, fred), Another_son #<> X.
:- son(X, mary), son(Another_son, mary), Another_son #<> X.
% 5.
man(X):- adult(X), male(X).
% 6.
bachelor(X):- man(X), not married(X, Y, T).
```

```
% 7, 8, 9
married(X,Y, T):- groom(X), bride(Y), wedded(X,Y,T1), T #> T1,
                  not divorced(X,Y,T).
-married(X):- bachelor(X).
divorced(X,Y, T):- husband(X), wife(Y), dissolved(X,Y,T1), T #> T1.

groom(X):- wedded(X,Y,T1), male(X).
bride(Y):- wedded(X,Y,T1), female(Y).
husband(X):- groom(X).
wife(X):- bride(X).

% Wedding precedes divorce
:- wedded(X,Y,T1), divorced(X,Y,T2), groom(X), bride(Y), T1 #> T2.
```

- George has never been the groom at a wedding.

```
?- not groom(george).
```

- Mary has an unmarried son born in Boston.

```
umsbb(M,X,C):-son(X,M), birth_city(X,C), not married(X,Y,T).
```

- No woman is the wife of any of Fred's children.

```
fred_child_wife(Y):-child(X,fred), married(X,Y,T), female(Y).
```

DEMO

Possible Worlds

People can talk.

Non-human animals can't talk.

Human-like cartoon characters can talk.

Fish can swim.

A fish is a non-human animal.

Nemo is a human_like cartoon character.

Nemo is a fish.

Can nemo talk?

Can nemo swim?

DEMO

Predicate ASP Systems

- Aside from coinduction many other challenges needed to be addressed to realize the s(ASP) & s(CASP) systems:
  - Dual rules to handle negation: recall rules are assumed as causal
  - Constructive negation support (domains are infinite)
  - Universally Quantified Vars (due to negation & duals)
- s(CASP): Prolog & CLP(Q) + stable model negation
  - available on SWI Prolog SWISH & Ciao Playground; native impl. Downloadable from GitLab
- Has been used for implementing many non-trivial applications:
  - Check if an undergraduate student can graduate at a US university
  - Physician advisory system to manage chronic heart failure
  - Natural Language & Visual Question Answering
  - Modeling Event Calculus (applications to modeling cyber physical systems)
  - Synthesis of concurrent programs (correct by construction program synthesis)
  - Automating Legal Reasoning; Autonomous driving; Software Certification; Logic-based Learning

Predicate ASP Systems: Challenges

- Constructive negation:
  - Consider fact: p(1). and query ?- not p(X). We should be able to produce the answer $X \neq 1$.
  - Need to generalize unification: each variable carries the values it cannot be bound to

    $X \neq a$ unifies with $Y \neq b$: result $X \neq \{a, b\}$

    s(ASP)/s(CASP) are the first systems with constructive negation properly implemented
- Dual rules to handle negation
  - Need to handle existentially quantified variables p(X) :- q(X, Y). $\equiv$ $\forall$X p(X) :- $\exists$Y q(X, Y).
  - Dual of p will have Y universally quantified in the body: need the **forall** mechanism.
  - Dual rule: not_p(X) :- $\forall$Y not_q(X, Y).
- Two key papers:
  - **s(ASP): Computing Stable Models of Normal Logic Programs without Grounding. Marple, Salazar, Gupta (on arXiv)**
  - **s(CASP): Constraint Answer Set Programming without Grounding. Proc. ICLP'18 Arias, Carro, Marple, Salazar, Gupta**

Conclusions

- Automating commonsense reasoning: holy grail of AI
- For automated commonsense reasoning, we need:
  - Default rules with exceptions and preferences
  - Global constraints to model impossibilities and invariants
  - Multiple possible worlds (or assumption-based/circular reasoning)
- Progress stymied by singular focus on inductive structures (single model)
- ASP: allows defaults, constraints and circular reasoning
- ASP: limited scalability due to SAT-solver based implementation
- s(ASP)/s(CASP) systems: Goal-directed predicate ASP; available in Ciao & SWI
- Think of s(CASP) as giving an operational semantics to human thinking
- Related work: several efforts, but they lack in one or more key aspects
- Applications: Many innovative applications pursued (GDE’21/GDE’22 workshop)

THANK YOU

QUESTIONS?

More information: http://utdallas.edu/~gupta