Answer Set Programming and Automating Common Sense Reasoning

Gopal Gupta
Department of Computer Science
The University of Texas at Dallas

Outline

p :- q, r, s.

q :- t, u.

t :- v.

v.

u.

r.

s.

1 step = {r, s, v, u}

2 step = {r, s, v, u, t}

3 step = {r, s, v, u, t, q}

4 step = {r, s, v, u, t, q, p}

5 step = {r, s, v, u, t, q, p} fixpoint

Outline

```
gen(N, L) :- gen(N, [], L).
gen(0, L, L).
gen(N, L, R) :- N > 0, N1 is N-1, gen(N1, [N|L], R).

queen(N,Q) :- gen(N, L), placeQ(L, [], Q).
placeQ([ ], Q, Q).
placeQ(L, PB, R) :- select(Q, L, RQs), safe([Q|PB]),  placeQ(RQs, [Q|PB], R).

safe([]).
safe([X|Y]) :- noattack(X, 1, Y), safe(Y).       [3, 5, 2, 1, 4]

noattack(X,N,[]).
noattack(X,N, [Y|Z]) :- Y =\= X + N, Y =\= X - N,
                        N1 is N+1, noattack(X,N1,Z).

select(X, [X|R], R).
select(X, [Y|T], [Y|R]) :- select(X,T,R).
```

Part 0: AI/KRR/Introduction

Part I: Common Sense Reasoning & Answer Set Programming (ASP)

Part II: Applications

Artificial Intelligence

- Intelligent reasoning by computers has been a goal of computer scientists ever since computers were first invented in the 1950s.
- Intelligence has two components:
  - Acquiring knowledge (learning): automate it ⇒ machine learning
  - Applying knowledge that is known (reasoning) automate it ⇒ automated reasoning
- Our focus: automated reasoning
- Reasoning is essential: machine learning algorithms learn rules that have to be employed for reasoning
- Machine learning oversold?:
  - Used in many places where reasoning would suffice;
  - reasoning ability is critical as humans both learn *and* reason.

Knowledge Representation and Reasoning

- Knowledge representation and reasoning
  - Need a language to represent knowledge [KR]
  - Need a language to process this knowledge and query it [R]
- Ideal if the language for KR and R is the same:
  - KR language is "executable" (i.e., *has a declarative and an operational semantics*)
  - Horn clauses (logic programming) constitute such a language
  - Knowledge is represented as facts and rules:

    p. p is a proposition that is unconditionally true

    q :- $b_1, b_2, b_3, \ldots, b_n$. q is a proposition that is conditionally true

    A rule is an implication $b_1 \wedge b_2 \wedge b_3 \wedge \ldots \wedge b_n \Rightarrow q$
  - Queries are of the form ?- $b_1, b_2, b_3, \ldots, b_n$. answered via SLD resolution

    query answering equals a refutation proof, i.e., show $b_1 \wedge b_2 \wedge b_3 \wedge \ldots \wedge b_n \Rightarrow$ false
- Many other alternatives: first order logic, description logic, semantic net, …

Logic Programming (Prolog)

Facts:

father(jim, john). mother(mary, john).

father(john, rob). mother(sue, rob).

…. ….

Rules:

parent(X, Y) if mother(X, Y).

parent(X, Y) if father(X, Y).

grandparent(X, Y) if parent(X, Z) & parent(Z, Y).

anc(X, Y) if parent(X, Y).

anc(X, Y) if parent(X, Z) & anc(Z, Y).

Queries:

?- anc(jim, rob). ?- anc(A, rob).

Logic Programming (Prolog)

Facts:

father(jim, john). mother(mary, john).

father(john, rob). mother(sue, rob).

…. ….

Rules:

parent(X, Y) ⇐ mother(X, Y).

parent(X, Y) ⇐ father(X, Y).

grandparent(X, Y) ⇐ parent(X, Z) ∧ parent(Z, Y).

anc(X, Y) ⇐ parent(X, Y).

anc(X, Y) ⇐ parent(X, Z) ∧ anc(Z, Y).

Queries:

?- anc(jim, rob). ?- anc(A, rob).

Logic Programming (Prolog)

Facts:

```
father(jim, john).    mother(mary, john).
father(john, rob).    mother(sue, rob).
....                  ....
```

Rules:

```
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
anc(X, Y) :- parent(X, Y).
anc(X, Y) :- parent(X, Z), anc(Z, Y).
```

Queries:

```
?- anc(jim, rob).          ?- anc(A, rob).
```

Part I: Common Sense Reasoning & Answer Set Programming

Common Sense Reasoning

- Standard Logic Prog. fails at performing human-style commonsense reasoning
- In fact, most formalisms have failed; problem: monotonicity
- 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)
     - 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; humans use this [default reasoning] pattern all the time:
     - Can't tell if it is raining outside. If I see no one holding an umbrella, so it must not be raining
     - You text your friend in the morning; he does not respond; normally he responds right away. You may conclude: he must be taking a shower.
  3. Possible worlds semantics: be able to work with multiple worlds (non-inductive semantics)
- Commonsense reasoning requires that we are able to handle *negation as failure & support possible world semantics*

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 Methods for Common Sense Reasoning

- 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 ¬T should hold.
- Most of common sense 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)

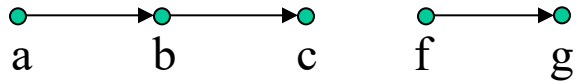
Failure of Classical Reasoning Methods for CSR

- 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) ↔ (edge(X, Y) ∨ (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.

Negation as Failure (NAF)

- Humans use NAF all the time; however, using NAF does make life difficult
- Hard (for humans) to deal with nested negation:

  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 all will go to Mexico?
- Code this as:

```
p :- not s.
s :- not r.
r  :- not p.
r  :- not s.
```

- What is the semantics of this program?
- Individual rules easy to understand; *extremely* hard to understand what the program means as a whole

Answer Set Programming (ASP)

- Prolog extended with NAF; Rules of the form:

  $p :- a_1,\ldots, a_m, not\ b_1 ,\ldots, not\ b_n.$ $m, n \geq 0$ (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 $a_1 ,\ldots, a_m$ are in the answer set and $b_1 ,\ldots, b_n$ are not
- The rule could take more general form

  $p :- a_1,\ldots, a_m, not\ b_1 ,\ldots, not\ b_n, -c_1, ..., -c_k, not\ -d_1 ,\ldots, not\ -d_i$ $m, n, k, i \geq 0$ (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 research invested

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, non-monotonic reasoning, constrained optimization, exceptions, weak exceptions, preferences, etc., in a natural way
  - Better way to build automated reasoning systems & expert systems
- Caveats
  - $p \Leftarrow a, b.$ really is taken to be $p \Leftrightarrow a, b.$
  - 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)

Human-style Common Sense Reasoning

- Why have we failed to model human-style common sense reasoning?
  - NAF + another reason: multiple possible worlds
- Frege & Russell: Frege discovers (naïve) set theory that Russell finds a problem with (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 (coinductive) is banished from the language.
  - 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

Automating Common Sense Reasoning

- If we automate CSR, we can automate the human thought process
- Model any intelligent task that humans can accomplish
- Achieve artificial general intelligence (AGI), at least
- Our goal is to simulate an unerring human
- So far:
  - We need negation-as-failure to model the fact that humans operate in a world where lot of stuff is unknown
  - We need possible world semantics, as humans do not always reason inductively, i.e., humans can reason with multiple worlds simultaneously; each of these multiple worlds has to be consistent though (coinductive)
- Answer Set Programming provides both these features, hence is an excellent candidate for modeling common sense reasoning

ASP

- Given an answer set program, we want to find out the knowledge (propositions) that it entails (answer sets)
- For example, given the program:

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

  the possible answer sets are:

  1. {p} i.e., p = true, q = false
  2. {q} i.e, q = true, p = false

p = Tom teaches DB
q = Mary teaches DB

Two worlds:
Tom teaches DB, Mary does not
Mary teaches DB, Tom does not

- Computed via Gelfond-Lifschitz Method (Guess & Check)
  - Given an answer set S, for each p $\in$ S, delete all rules whose body contains “not p”;
  - delete all goals of the form “not q” in remaining rules
  - Compute the least fix point, L, of the residual program
  - If S = L, then S is an answer set

Finding the Answer Set

- Consider the program:

p	t.	r :- t, s.
	s.	h :- p

- Is {p, r, t, s} the answer set?
- Apply the GL Method
  - -- If x in answer set, delete all rules with not x in body
  - -- Next, remove all negated goals from the remaining program
  - -- Find the LFP of the program: {p, r, t, s, h}
  - -- Initial guess {p, r, t, s} ≠ {p, r, t, s, h} so {p, r, t, s} is not a stable model.

Finding the Answer Set

- Consider the program:

```
                                t.      r :- t, s.
    q :- r.                     s.      h :- p
```

- Is {q, r, t, s} the answer set?
- Apply the GL Method
  - -- If x in answer set, delete all rules with not x in body
  - -- Next, remove all negated goals from the remaining program
  - -- Find the LFP of the program: {q, r, t, s}
  - -- Initial guess {q, r, t, s} = LFP
    so {q, r, t, s} is a stable model.

ASP

- Π_1:
  $p \leftarrow a.$
  $q \leftarrow b.$
  $a \leftarrow .$
  {p,a}

- Π_2:
  $p \leftarrow p.$
  {}

- Π_3:
  $p \leftarrow p.$
  $q \leftarrow .$
  {q}

- Π_4:
  $a \leftarrow$ ***not b.***
  {a}

All examples from Baral'04 (slides)

- Π_5:
  $a \leftarrow$ ***not b.***
  $b \leftarrow$ ***not a.***
  {a}
  {b}

- Π_6:
  $p \leftarrow a.$
  $a \leftarrow$ ***not b.***
  $b \leftarrow$ ***not a.***
  {p, a}
  {b}

- Π_7:
  $a \leftarrow$ ***not b.***
  $b \leftarrow$ ***not c.***
  $d \leftarrow .$
  {b,d}

- Π_8:
  $p \leftarrow$ ***not p.***
  none

- Π_9:
  $p \leftarrow$ ***not p, d.***
  $r \leftarrow .$
  $d \leftarrow .$
  none

- Π_{10}:
  $p \leftarrow$ ***not p, d.***
  $r \leftarrow$ ***not d.***
  $d \leftarrow$ ***not r.***
  {r}

- Π_{11}:
  $p \leftarrow$ ***not p.***
  $p \leftarrow$ ***not d.***
  $r \leftarrow$ ***not d.***
  $d \leftarrow$ ***not r.***
  {p,r}

ASP: Examples

- Consider the following program, $\mathbb{A}$:

  p :- not q. t. r :- t, s.
  q :- not p. s. h :- p, not h.
  $\mathbb{A}$ has 2 answer sets: {p, r, t, s} & {q, r, t, s}.

Even loops create multiple worlds
Odd loops kill worlds

- Now suppose we add the following rule to $\mathbb{A}$:

  h :- p, not h. (falsify p)
  Only one answer set remains: {q, r, t, s}

- Consider another program:

  p :- not s.
  s :- not r.
  r :- not p.
  r :- not s.

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

What are the answer sets? {p, r}

Constraints

- The rules that cause problem are of the form:

  h :- p, q, r, not h. that implicitly declares p ∧ q ∧ r to be false
- ASP also allows rules of the form (headless rule or constraint):

  :- p, q, r.

  which asserts that the conjunction of p, q, and r should be false.
- The two are equivalent, except that in the first case, not h may be called indirectly:

  h :- p, q, s.

  s :- r, not h.
- Constraints are responsible for nonmonotonic behavior
- A rule of the form

  p :- not p

  wrecks the whole knowledge base as p ⇔ not p has no model.

Answer Set Programming

- Answer set programming subsumes Horn clause logic programming
- Generate & Test paradigm realized via even loops and odd loops/constraints
- Even loops: p :- not q. q :- not p.
- Odd loops/constraints: h :- r, not h. OR :- r.
- Even loops create (multiple) worlds, odd loops kill worlds
- Consider

  p :- not q.
  q :- not p.
  :- p.
- Even loops creates two worlds: {p, not q}, {q , not p}
- Constraint kills one of them, so only one world remains: {q, not p}

Slide from Son Tran of NMSU

Commonsense Reasoning

Our knowledge

- is often *incomplete* (it does not contain complete information about the world), and
- contains *defaults* (rules which have exceptions, also called normative sentences).
- contains *preferences* between defaults (prefer a conclusion/default).

For this reasons, we often jump to conclusions (ignore what we do not now), and know to deal with exceptions and preferences.

Example 7 *We know*

Normally, birds fly.
Normally, computer science students can program.
Normally, students work hard.
Normally, things do not change.
Normally, students do not watch TV.
Normally, the speed limit on highways is 70 mph.
Normally, it is cold in December.

From this, we make conclusions such as Tweety *flies if we know that* Tweey is a bird*;* Monica *can program if* she is a computer science student*; etc.*

Slide from Son Tran of NMSU

Representing Defaults

Normally, a's are b's. $b(X) \leftarrow a(X), \textbf{not } ab(r, X)$[1]

Normally, birds fly. $flies(X) \leftarrow bird(X), \textbf{not } ab(bird\_fly, X)$

Normally, animals have four legs.

$numberoflegs(X, 4) \leftarrow animal(X), \textbf{not } ab(animal, X)$

Normally, fishs swim. $swim(X) \leftarrow fish(X), \textbf{not } ab(fish, X)$

Normally, computer science students can program.

$can\_program(X) \leftarrow student(X), is\_in(X, cs), \textbf{not } ab_p(X)$

Normally, students work hard.

$hard\_working(X) \leftarrow student(X), \textbf{not } ab(X)$

Typically, classes start at 8am.

$start\_time(X, 8am) \leftarrow class(X), \textbf{not } ab(X)$

[1] r is the 'name' of the statement; $ab(X)$ or $ab_r(X)$ can also be used.

Slide from Chita Baral (ASU)

Defaults and Exceptions

- "Normally birds Fly. Tweety and Sam are birds."

  $flies(X) \leftarrow bird(X), \textbf{not}\ ab(X).$
  $bird(tweety) \leftarrow.$
  $bird(sam) \leftarrow.$

  AS = {flies(tweety), flies(sam)}

- Adding "Sam is a penguin, and penguins are exceptional birds that do not fly," in an elaboration tolerant manner:

  $bird(X) \leftarrow penguin(X).$
  $ab(X) \leftarrow penguin(X).$
  $penguin(sam) \leftarrow.$

  AS = {flies(tweety))}
  flies(sam) does not hold any more

- Adding "Ostriches are exceptional birds that do not fly," in an elaboration tolerant manner:

  $bird(X) \leftarrow ostrich(X).$
  $ab(X) \leftarrow ostrich(X).$

Our reasoning is aggressive: if we don't know anything about a bird, it can fly

Slide from Chita Baral (ASU)

Defaults and Exceptions

- Our reasoning could be conservative: X flies only if we conclusively prove it is not a penguin

Aggressive

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

-penguin(X) :- not penguin(X).
```

Conservative

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

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

Slide from C. Baral (ASU)

University Admission

To put all this in perspective, consider the college admission process:

$(1)\ eligible(X) \leftarrow highGPA(X).$
$(2)\ eligible(X) \leftarrow special(X), fairGPA(X).$
$(3)\ \neg eligible(X) \leftarrow \neg special(X), \neg highGPA(X).$
$(4)\ interview(X) \leftarrow \mathbf{not}\ eligible(X), \mathbf{not}\ \neg eligible(X).$
$(5)\ fairGPA(john) \leftarrow.$
$(6)\ \neg highGPA(john) \leftarrow.$

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 for Common Sense Reasoning

- ASP allows human-style common sense reasoning to be modeled (due to inclusion of negation as failure)
- Negation as failure allows us to ignore unnecessary information that is not relevant at the moment (defaults & exceptions).
- ASP allows us to take actions predicated on lack of information:

  use_gps_system(X) :- not directions_known(X).

- ASP gives us a hierarchy of (un)certainty that we humans employ, given some proposition p (e.g., p = it is raining now):
  - p 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

- ASP has possible world semantics where multiple worlds can exist
- As a result: ASP allows human knowledge and reasoning to be modeled faithfully.

Incomplete Information

- Consider the course database:

Professor	Course
mike	ai
sam	db
staff	pl

Represented as:

teach(mike,ai).
teach(sam,db).
teach(staff,pl).

- By default professor P does not teach course C, if teach(P,C) is absent.
- The exceptions are courses labeled “staff”. Thus:

  ¬teach(P,C) :- not teach(P,C), not ab(P,C).
  ab(P,C) :- teach(staff, C).
- Queries

  ?- teach(mike, pl) and

  ?- ¬teach(mike,pl)

  will both fail: we really don’t know if mike teaches pl or not (unknown)

Combinatorial Problems: Coloring

Graph Coloring

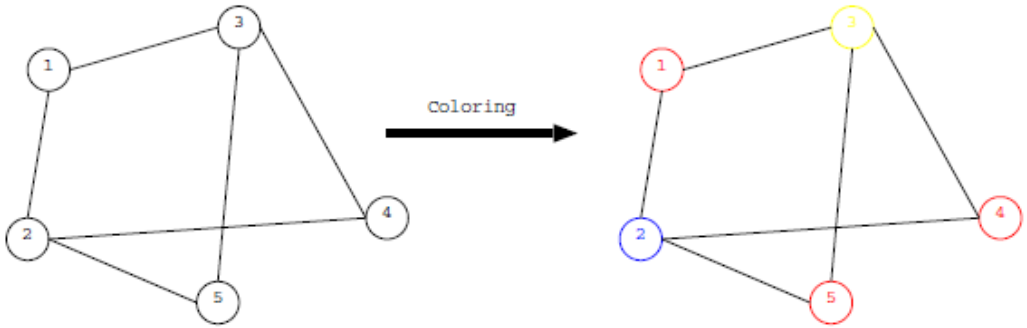


Figure 1: Graph Coloring Problem

Problem: Given a (bi-directed) graph and three colors *red*, *green*, and *yellow*. Find a color assignment for the nodes of the graph such that no edge of the graph connects two nodes of the same color.

Slide from S. Tran (NMSU)

Combinatorial Problems: Coloring

```
%% prog1
%% representing the graph
node(1).   node(2).   node(3).   node(4).   node(5).

edge(1,2). edge(1,3). edge(2,4). edge(2,5). edge(3,4). edge(3,5).

%% each node is assigned a color
color(X,red):- not color(X,green), not color(X, yellow).
color(X,green):- not color(X,red), not color(X, yellow).
color(X,yellow):- not color(X,green), not color(X, red).

%% constraint checking
:- edge(X,Y), color(X,C), color(Y,C).
```

Slide from S. Tran (NMSU)

Combinatorial Problems: N-Queens

1. Declarations:

$queen(1) \leftarrow .$ ... $queen(n) \leftarrow .$
$row(1) \leftarrow .$ ... $row(n) \leftarrow .$
$col(1) \leftarrow .$ ... $col(n) \leftarrow .$

2. Enumeration:

2.1. For each locations (X, Y) and each queen I, either I is in location (X, Y) or not.

$at(I, X, Y) \leftarrow queen(I), row(X), col(Y), \textbf{not}\ not\_at(I, X, Y).$
$not\_at(I, X, Y) \leftarrow queen(I), row(X), col(Y), \textbf{not}\ at(I, X, Y).$

2.2. For each queen I it is placed in at most one location.

$\leftarrow queen(I), row(X), col(Y), row(U), col(Z), at(I, X, Y), at(I, U, Z), Y \neq Z.$
$\leftarrow queen(I), row(X), col(Y), row(Z), col(V), at(I, X, Y), at(I, Z, V), X \neq Z.$

2.3. For each queen I it is placed in at least one location.

$placed(I) \leftarrow queen(I), row(X), col(Y), at(I, X, Y).$
$\leftarrow queen(I), \textbf{not}\ placed(I).$

Slide from S. Tran (NMSU)

Combinatorial Problems: N-Queens

2.4. No two queens are placed in the same location.

$\leftarrow queen(I), row(X), col(Y), queen(J), at(I, X, Y), at(J, X, Y), I \neq J.$

3. Elimination:

3.1. No two distinct queens in the same row.

$\leftarrow queen(I), row(X), col(Y), col(V), queen(J), at(I, X, Y), at(J, X, V), I \neq J.$

3.2. No two distinct queens in the same column.

$\leftarrow queen(I), row(X), col(Y), row(U), queen(J), at(I, X, Y), at(J, U, Y), I \neq J.$

3.3. No two distinct queens attack each other diagonally.

$\leftarrow row(X), col(Y), row(U), col(V), queen(I), queen(J), at(I, X, Y),$
$at(J, U, V), I \neq J, abs(X - U) = abs(Y - V).$

Slide from S. Tran (NMSU)

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?

Representing Knowledge with ASP

- Suppose Mary believes that John was in Holland Park some morning and that Holland Park is in London. Then Mary can deductively reason from these beliefs, to conclude that John was in London that morning. So the reasoning cannot be attacked. However, perfection remains unattainable since the argument is still fallible: its grounds may turn out to be wrong. For instance, Jan may tell us that he met John in Amsterdam that morning around the same time. We now have a reason against Mary's belief that John was in Holland Park that morning, since witnesses usually speak the truth. Can we retain our belief or must we give it up? The answer to this question determines whether we can accept that John was in London that morning.

- Maybe Mary originally believed that John was in Holland Park for a reason. Maybe Mary went jogging in Holland Park and she saw John. We then have a reason supporting Mary's belief that John was in Holland Park that morning, since we know that a person's senses are usually accurate. But we cannot be sure, since Jan told us that he met John in Amsterdam that morning around the same time. Perhaps Mary's senses betrayed her that morning? But then we hear that Jan has a reason to lie, since John is a suspect in a robbery in Holland Park that morning and Jan and John are friends. We then conclude that the basis for questioning Mary's belief that John was in Holland Park that morning (namely, that witnesses usually speak the truth and Jan witnessed John in Amsterdam) does not apply to witnesses who have a reason to lie. So our reason in support of Mary's belief is undefeated and we accept it.

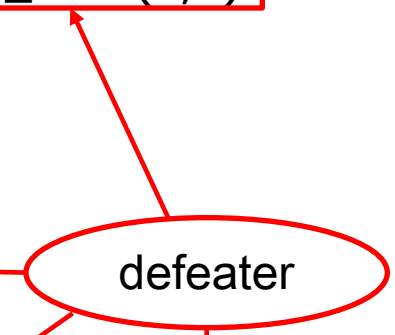
Representing Knowledge with ASP

- We map various OaP concepts into ASP, e.g., the argument (rule) for speaking truth:

speaks_truth(X, Y) :- person(X), person(Y), observer(E,X), theme(E,Y), not ab_speaks_truth(X,Y).
ab_speaks_truth(X, Y) :- may_lie(X,Y).

may_lie(X,Y) :- person(X), person(Y), not -lie(X,Y).
-lie(X,Y) :- person(X), person(Y), not conflict_interest(X,Y).

conflict_interest(X,Y) :- friends(X,Y), crime_suspect(Y), not ab_conflict_interest(X).
crime_suspect(X) :- person(X), robbery_suspect(X), not ab_crime_suspect(X).



- Likewise for Mary's infirmity

accurate_senses(X) :- person(X), not ab_accurate_senses(X).
ab_accurate_senses(X,sense) :- person(X), age(X, A), A = old.

Representing Knowledge with ASP

- Based on the assumptions we make, we can prove different claims:
  - Mary is infirm and has poor eyesight #abducible age(mary, old).
  - Jan and John are friends #abducible friends(john, jan).
  - John is a robbery suspect #abducible robbery_suspect(john).
- Assumptions are modeled as abducibles (abductive reasoning)
  - Given p $\Rightarrow$ q and an observation q, we *abduce* (*assume*) that p is true
- Now claims can be established automatically with requisite assumptions
- ?- claim(john_location_unknown).

  Assumptions: age(mary,old) $\wedge$ friends(jan,john) $\wedge$ robbery_suspect(john)

- ?- claim(john_in_london).

  Assumptions: age(mary,old) $\wedge$ not friends(jan, john).

  Assumptions: age(mary,old) $\wedge$ friends(jan,john) $\wedge$ not robbery_suspect(john)

- ?- claim(john_not_in_london).

  Assumptions: friends(jan,john) $\wedge$ robbery_suspect(john)

  Assumptions: age(mary, B | B $\neq$ old) $\wedge$ friends(jan,john) $\wedge$ robbery_suspect(john)

Justification tree in English can be produced

- s(CASP) can produce justification tree (proof tree) for a given claim:

John is in London at time 8AM, because
Mary saw John at 7:45AM, and
John was at holland_park at 7:45AM, and
holland_park is in london, and
Mary has sharp senses.
Sighting happened at time 7:45AM, because
John was at holland_park at 7:45AM.
Jan saw John at 7:45AM, and ...
John was at Amsterdam at 7:45AM, and
John is no longer in London at 7:45AM, because
Jan saw John at 7:45AM.
Meeting event involves John, because
Jan saw John at 7:45AM.
Meeting event involves John, justified above, and
Jan may lie about John, because
Jan has conflict of interest with John, because
we assume that Jan and John are friends, and
John is a crime suspect.

Yale Shooting Problem

```
#include 'bec-kinjal.lp’

%Initially gun is not loaded.                    %Initially the turkey is alive,
%Load event causes gun to be loaded.             %Shoot event causes turkey to be not alive if gun is Loaded,
%Shoot event causes gun to be not loaded.

initiallyN(loaded).
initiallyP(alive).
initiates(load, loaded, T) :- happens(load, T).        %load causes loaded
terminates(shoot, loaded, T) :- happens(shoot, T).     %shoot causes not loaded
terminates(shoot, alive, T) :- happens(shoot, T).      %shoot causes not alive

happens(load, 1).
happens(shoot, 2).

:- happens(shoot,T1), holds(loaded,T2), T1 .<. T2. %can shoot only if loaded

%?- holds(alive,0.5).          % True          %?- holds(loaded,0.5).          % False
%?- holds(alive,1.5).          % True          %?- holds(loaded,1.5).          % True
%?- -holds(alive,2.5).         % True          %?- -holds(loaded,2.5).         % True
```

Current ASP Systems

- Very sophisticated and efficient ASP systems have been developed based on SAT solvers:
  - CLASP, DLV, CModels
- These systems work as follows:
  - Ground the programs w.r.t. the constants present in the program
  - 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: why is that?

Part III: Applications of (goal-directed) ASP

ASP: Applications

- Combination of negation as failure and classical negation provides very sophisticated reasoning abilities.
- Each rule by itself is easy to understand/write; put the rules together and it is hard to understand what the rules compute
- Many many applications developed by us and others
  - Degree Audit System: Given a student's transcript, can a student graduate subject to catalog rules?
  - Coding knowledge of high school biology text
  - Chronic Disease management (simulate a physician's expertise)
  - Concurrent program synthesis: automatically synthesize programs from axioms specifying a pointer data structure, axioms specifying concurrency, shared memory, etc. Just like humans do.
  - Recommendation systems: intelligent recommendation systems can be constructed based on ASP
  - Intelligent IoT: intelligent behavior programmed using ASP across multiple IoT devices
  - Natural Language Question Answering; Chatbots that truly understand the conversation
  - Visual Question Answering: Answer natural language questions about a picture

Applications Developed with s(ASP)

- Graduation audit system: can an undergrad student at a university graduate?
  - Complex requirements expressed as ASP rules

```
_takenFor(Student, Course, Req):-
    % Use _takenForElective/3 to avoid unnecessary call to _req/2 in _takenForElective/2
    % since we assume a course can only satisfy one requirement.
    not _takenForElective(Student,Course,Req).

_takenForElective(Student, Course):-
    % 'not _takenFor(Student, Course, none)' should always hold because 'none'
    % is a placeholder requirement for courses that do not satisfy any requirement.
    % So, we can avoid an unnecessary call to the dual of _takenFor/3 and gain some performance.
    _req(Course,none).

_takenForElective(Student, Course, Req):-
    not _takenFor(Student,Course,Req).
```

  - 100s of courses that students could potentially choose from
  - Lists and structures come in handy in keeping the program compact
  - Developed over a few weeks by 2 undergraduate students
  - Catalog changes frequently: ASP rules easy to update

Physician Advisory System: Heart Failure

- Advises doctors for managing chronic heart failure in patients
  - Automates American College of Cardiology guidelines
  - Complex guidelines expressed as rules (80 odd rules in 100 pages)
    - Combination of hydralazine & isosorbide dinitrate should not be used for the treatment of HFrEF in patients who have no prior use of standard neurohormonal antagonist therapy.
    - ARBs are recommended in patients with HFrEF with current or prior symptoms who are ACE inhibitor intolerant, unless contraindicated, to reduce morbidity and mortality
    - In patients with a current or recent history of fluid retention, beta blockers should not be prescribed without diuretics
  - Tested with UT Southwestern medical center;
    - ✓ Found diagnosis that was missed by the doctor
    - ✓ Outperforms cardiologists
  - Available as a product: Try it out: http://52.73.186.16/
  - Enter patient data and obtain a diagnosis including justification
- General paradigm for automating medical diagnosis and treatment

Natural Language/Picture Understanding

- When we read natural language passages (or look at a picture), our understanding is enhanced by common sense knowledge we possess
- Suppose we read the hare and turtle story and are asked the question: does the hare have eyes? Easy for humans to answer, hard for AI systems
  - ML systems will fail, unless trained on the needed knowledge
- To be able to answer questions intelligently, we need to
  1. Code the natural language story in ASP
  2. Code common sense knowledge in ASP
  3. Translate the question into an ASP query, pose it against ASP-coded knowledge of 1 & 2
- Similarly, code the information in a picture into ASP, add common-sense knowledge, and then pose questions
- Various systems developed: CASPR, AQuA, SQuARE

Question Answering Projects at UT Dallas

- **CASPR System**: *Example article from* **SQuAD Dataset**

  The American Broadcasting Company (ABC) (stylized in its logo as ABC since 1957) is an American commercial broadcast television network that is owned by the Disney–ABC Television Group, a subsidiary of Disney Media Networks division of The Walt Disney Company. The network is part of the Big Three television networks. The network is headquartered on Columbus Avenue and West 66th Street in Manhattan, with additional major offices and production facilities in New York City, Los Angeles and Burbank, California.

- **CASPR System** *can answer a question such as:*

  Q1. What company owns the American Broadcasting Company?

  A. [The Walt Disney Company, Disney–ABC Television Group, Disney–ABC Television Group]

- **SQuARE system**: 100% accuracy on 12 **bAbI datasets from Facebook** with some stories ~100 lines long (though sentences are simple); each dataset has 2,000+ stories of varying sizes
- **AQuA system**: ~100% accuracy on CLEVR dataset pictures; errors are mostly in picture recognition part

Intelligent IoT

- Intelligent systems that take as input sensor signals and determine actuator action based on some logic can be developed
- Situation where we take an action in the absence of a signal arises frequently (modeled using negation as failure)
- Example: intelligent lighting system

```
turn_on_light :- motion_detected, door_opened_or_closed.
turn_on_light :- light_is_on, motion_detected.
turn_on_light :- light_is_on, not(motion_detected), not(door_opened_or_closed).
-turn_on_light :- not(motion_detected), door_moved.
```

- IoT systems better modeled using the event calculus

Modeling with the Event Calculus

- House safety example:
  - If fire is detected, turn on sprinkler system
  - If water leak detected and no one present, turn off water supply
- Problem: with these rules, house will burn down if there is fire
- Exception: need to recognize that we should not turn off water if there is fire
- Model this with the event calculus, a formalism for dynamic systems
  - Fluent : A property that changes over time and is affected by events: **sprinkler_on, sprinkler_off**
  - Event: A predicate that represents the occurrences of events: **turn_sprinkler_on, turn_sprinkler_off**
  - Events have effect on fluents: **turn_sprinkler_on effects {sprinkler_on}**
  - States (derived fluents) : Fluents that abstract some properties of other fluents:

    house_burns if fire_detected and sprinkler_off

Basic Event Calculus Axioms

BEC1. $StoppedIn(t_1, f, t_2) \equiv$
$\exists e, t \ (\ Happens(e,t) \wedge t_1 < t < t_2 \wedge (\ Terminates(e,f,t) \vee Releases(e,f,t) \ ) \ )$

BEC2. $StartedIn(t_1, f, t_2) \equiv$
$\exists e, t \ (\ Happens(e,t) \wedge t_1 < t < t_2 \wedge (\ Initiates(e,f,t) \vee Releases(e,f,t) \ ) \ )$

BEC3. $HoldsAt(f_2, t_2) \leftarrow$
$Happens(e, t_1) \wedge Initiates(e, f_1, t_1) \wedge Trajectory(f_1, t_1, f_2, t_2) \wedge \neg StoppedIn(t_1, f_1, t_2)$

BEC4. $HoldsAt(f, t) \leftarrow InitiallyP(f) \wedge \neg StoppedIn(0, f, t)$

BEC5. $\neg HoldsAt(f, t) \leftarrow InitiallyN(f) \wedge \neg StartedIn(0, f, t)$

BEC6. $HoldsAt(f, t_2) \leftarrow$
$Happens(e, t_1) \wedge Initiates(e, f, t_1) \wedge t_1 < t_2 \wedge \neg StoppedIn(t_1, f, t_2)$

BEC7. $\neg HoldsAt(f, t_2) \leftarrow$
$Happens(e, t_1) \wedge Terminates(e, f, t_1) \wedge t_1 < t_2 \wedge \neg StartedIn(t_1, f, t_2)$

Is the House Safe?

```
fluent(sprinkler_on).        fluent(sprinkler_off).   fluent(people_present).
event(turn_sprinkler_on).    event(turn_sprinkler_off).
```

% turn sprinkler on immediately when fire_detected

```
happens(turn_sprinkler_on, T) :- fire_detected(T).
```

% fluents enabled and terminated by the event: turn_sprinkler_on

```
initiates(turn_sprinkler_on, sprinkler_on, T) :- happens(turn_sprinkler_on, T).
terminates(turn_sprinkler_on, sprinkler_off, T):-happens(turn_sprinkler_on, T).
```

% turn sprinkler off immediately when water_leak is detected

```
happens(turn_sprinkler_off, T) :- water_leak(T),-holdsAt(people_present, T).
fire_happened(T) :- fire_detected(T1), T .>=. T1
house_burns :- fire_happened(T), T1 .>. T, holdsAt(sprinkler_off, T1).
```

Query: `?- house_burns.`

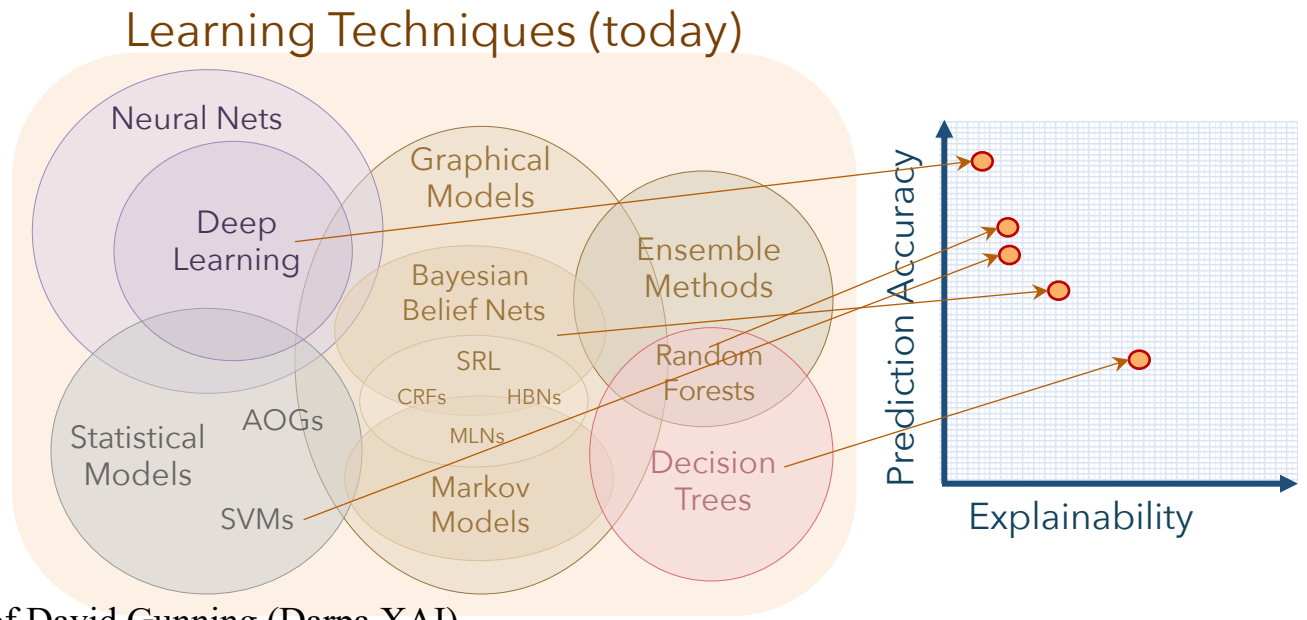
- Event calculus with real-time constraints can be run on the s(CASP) system
- Can be used to model requirements, understanding them, and finding gaps in them using abductive and counterfactual reasoning.
- s(ASP)/s(CASP) support abduction; support for counterfactual reasoning in progress

Recommendation Systems

- Recommendation systems in reality need sophisticated modeling of human behavior
- Current systems based on machine learning look for patterns: underlying knowledge of human behavior will help
- Birthday gift advisor:
  - model the generosity, wealth level of the person
  - model the interests of the person receiving the gift
  - use the information to determine the appropriate gift for one of your friends
  - very complex dynamics go in determining what gift you will get a person
- In general, any type of recommendation system can be similarly built.

ASP for Inducing Programs (ML)

- ASP programs can be learned from examples (induction)
- Aim: make complex machine learning models explainable
  - Important for developing *trust* in the model
  - Important due to regulatory requirements (e.g., GDPR)
- Trade-off between Accuracy vs. Explainability



[graphics courtesy of David Gunning (Darpa XAI)

FOLD System @ UTD (AAAI '19)

```
(1) fiftyK(A) :- maritalstatus(A,married),
                 education(A,bachelors).
(2) fiftyK(A) :- maritalstatus(A,married),
                 age(A,C), 34 < C < 62,
                 occupation(A,professional).
(3) fiftyK(A) :- maritalstatus(A,married),
                 age(A,C), 34 < C < 62,
                 occupation(A,whitecollar).
(4) fiftyK(A) :- maritalstatus(A,married),
                 capitalgain(A,C), 7073 < C < 10585.
(5) fiftyK(A) :- male(A), age(A,C), 34 < C < 62,
                 occupation(A,professional),
                 education(A,prof-school).
(6) fiftyK(A) :- male(A), education(A,bachelors),
                 occupation(A,whitecollar).
```

Marriage and financial stability

Gender Pay Gap

Conclusions

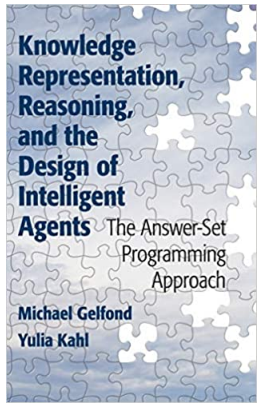
- ASP: A very powerful paradigm
- Can simulate complex reasoning, including common sense reasoning employed by humans
- Rules are easy to write, but semantics hard to compute
- Many interesting applications possible
- Goal-directed implementations are crucial to ASP's success
- Query-driven ASP: a powerful methodology for developing practical common sense reasoning applications in multiple domains
- Automating common sense reasoning: holy grail of AGI

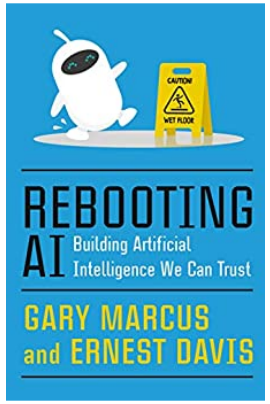
Acknowledgement:

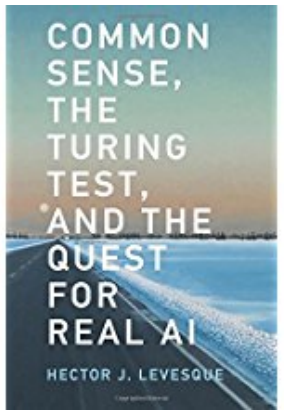
- ✓ Multiple National Science Foundation Projects & DARPA for research support
- ✓ Many students and colleagues: Luke Simon, Ajay Bansal, Ajay Mallya, Kyle Marple, Elmer Salazar, Richard Min, Brian DeVries, Feliks Kluzniak, Neda Saeedloei, Zhuo Chen, Lakshman Tamil, Arman Sobhi, Sai Srirangapalli, Joaquin Arias (IMDEAS), Manuel Carro (IMDEA), Sarat Varanasi, Kinjal Basu, Serdar Erbatur, Farhad Shakerin,

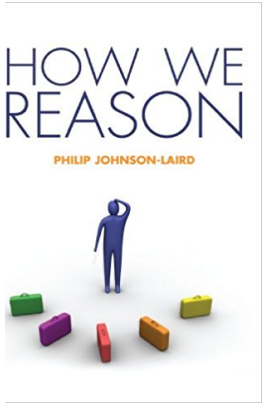
Further Reading

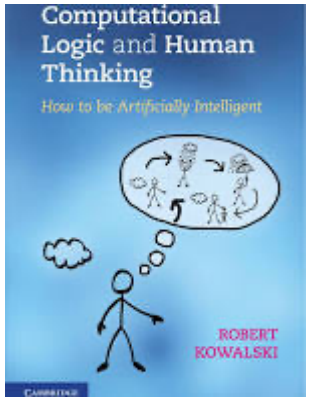
- Kyle Marple, Elmer Salazar, Gopal Gupta: Computing Stable Models of Normal Logic Programs Without Grounding. arXiv.org
- Joaquín Arias, Manuel Carro, Elmer Salazar, Kyle Marple, Gopal Gupta: Constraint Answer Set Programming without Grounding. ICLP 2018
- Z. Chen, K. Marple, E. Salazar, G. Gupta, L. Tamil: A Physician Advisory System for Chronic Heart Failure management based on knowledge patterns. ICLP 2016
- D. Phendarkar, G. Gupta. An ASP-based Approach to Representing and Querying Textual Knowledge. PADL'19; http://utdallas.edu/~gupta/caspr.pdf
- F. Shakerin, G. Gupta: Inducing non-monotonic logic programs for explaining machine learning models. AAAI 2019. https://arxiv.org/abs/1808.00629
- S. Varanasi, N. Mittal, G. Gupta. Pointer Data Structure Synthesis from ASP Specifications, submitted
- K. Basu, et al. SQuARE: Semantics-based Question Answering; submitted

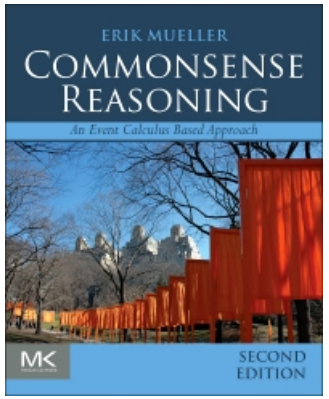


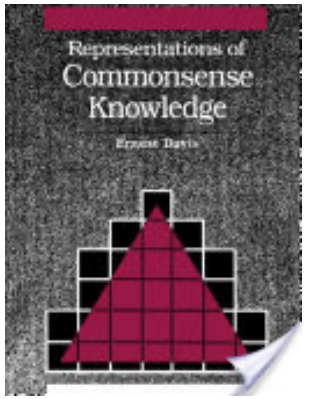












THANK YOU

QUESTIONS?

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

Part II: Goal-directed Execution of ASP

Why only propositional ASP?

- Consider the even loop:

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

- The completed program is:

  teach_db(john) $\Leftrightarrow$ not teach_db(mary). teach_db(mary) $\Leftrightarrow$ not teach_db(john).

- These are propositional formulas; models quickly found using a SAT solver:
  - {teach_db(john) = true, teach_db(mary) = false}
  - {teach_db(mary) = true, teach_db(john) = false}
- What if we have: $\forall X \forall Y$ if X teaches databases, Y does not, and vice versa

  teach_db(X) :- not teach_db(Y), $X \neq Y$. teach_db(Y) :- not teach_db(X).

- One solution is to simply "ground" the program with all possible values of X and Y (that's the approach adopted by most)
- What if domains of X and Y are not known: we would need to have query-driven execution like Prolog

Why only propositional ASP?

- Problem: Possible world semantics in predicate logic has not been explored
  - blame Russell & Tarski's push to keep everything inductive
- All semantics considered for computational logics are least fixed point based, i.e., inductive
- We need greatest fixed point based semantics as well (coinductive semantics)
- Breakthrough: coinductive logic programming (ICLP 2006) [Luke Simon thesis]
- Coinductive LP: operational semantics for computing elements of the greatest fix point of a program:

  Program $\prod$: jack_eats_food :- jill_eats_food.

  jill_eats_food :- jack_eats_food

  gfp($\prod$) = {jack_eats_food, jill_eats_food} lfp($\prod$) = { }
- Possible to now have predicate answer set programming (with full predicates)

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:** Some times 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 Inconsitency:** 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

- 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 the s(ASP) system

Developed at UT Dallas

s(ASP) System

- s(ASP)
  - Prolog extended with stable-model semantics;
  - general predicates allowed;
  - goal-directed execution
- With s(ASP):
  - Grounding and Explosion: are irrelevant as there is no grounding
  - Entire Model: only partial model relevant for answering the query is computed
  - Answer Unidentifiable: answer is easily identifiable due to query driven nature
  - KB inconsistency: consistency checks can be performed incrementally so that if the knowledge base is inconsistent, consistent part of the knowledge base is still useful
- Available on sourceforge; includes justification & abduction
- s(CASP): s(ASP) extended with constraints over real numbers
- s(CASP): impl. by Joaquin Arias, available on github, recommended for use

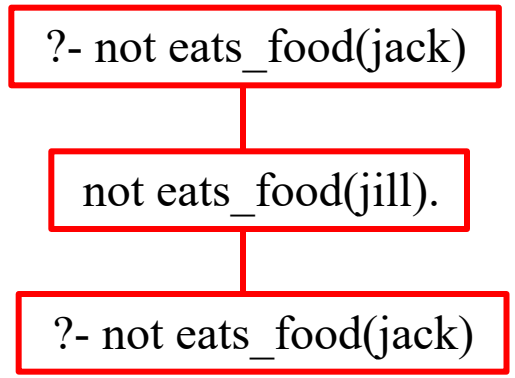
Goal-directed execution of ASP

- 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)



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 v not q)

Goal-directed ASP

- Consider the following program, A':

  p :- not q. t. r :- t, s.

  q :- not p, r. s. h :- p, not h.

- Separate into odd loop and even loop rules: only 1 odd loop rule in this case.
- Execute the query under co-LP, candidate answer sets will be generated.
- Keep ones not rejected by the constraints arising out of odd loop rules.
- Suppose the query is ?- q. Execution:
  - q ⇒ not p, r ⇒ not not q, r ⇒ q, r ⇒ r ⇒ t, s ⇒ s ⇒ success Ans = {q, r, t, s}
- Next, we need to check that constraint rules will not reject the generated answer set.
  - The query is extended with (h ∨ not p)
  - The computed answer set satisfies (h ∨ not p)

Goal-directed ASP

- In general, for the constraint rules of p as head,
  p_1:- B_1. p_2:- B_2. ... p_n :- B_n., generate rule(s) of the form:
  chk_p_1 :- not(p_1), B_1.
  chk_p_2 :- not(p_2), B_2.
  ...
  chk_p_n :- not(p), B_n.
- Generate: nmr_chk :- not(chk_p_1), ... , not(chk_p_n).
- For each pred. definition, generate its negative version:
  not_p :- not(B_1), not(B_2), ... , not(B_n).
- If you want to ask query Q, then ask ?- Q, nmr_chk.
- Execution keeps track of atoms in the answer set (PCHS) and atoms not in the answer set (NCHS).

Goal-directed ASP

- Consider the following program, P1:

  (i) p :- not q. (ii) q:- not r. (iii) r :- not p. (iv) q :- not p.

  P1 has 1 answer set: {q, r}.

- Separate into: 3 constraint rules (i, ii, iii)

  2 non-constraint rules (i, iv).

  p :- not(q). q :- not(r). r :- not(p). q :- not(p).

  chk_p :- not(p), not(q). chk_q :- not(q), not(r). chk_r :- not(r), not(p).

  nmr_chk :- not(chk_p), not(chk_q), not(chk_r).

  not_p :- q. not_q :- r, p. not_r :- p.

Suppose the query is ?- r.

Expand as in co-LP: r → not p → not not q → q (→ not r → fail, backtrack) → not p → success. Ans={r, q} which satisfies the constraint rules of nmr_chk.

Predicate ASP Systems: Challenges

- Aside from coinduction many other innovations needed to realize the s(ASP) system:
  - 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(ASP): Essentially, Prolog with stable model-based negation
- Has been used for implementing non-trivial applications:
  - Check if an undergraduate student can graduate at a US university
  - Physician advisory system to manage chronic heart failure
  - Representing cell biology knowledge as an ASP program
  - Natural Language Question Answering
  - Modeling Event Calculus (applications to IoT and Software Assurance)
  - Synthesis of concurrent programs

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) is the first system 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).
- Programs such as:

  p(X) :- not q(X). q(X) :- not p(X).

  produce uncountable number of models

s(ASP) available on sourceforge
s(CASP) available on github

Computing Stable Models of Normal Logic Programs without Grounding. Marple, Salazar, Gupta, on arXiv.