Introduction to Logic, Logic Programming Concepts and Languages

CSE 505 – Computing with Logic

Stony Brook University

http://www.cs.stonybrook.edu/~cse505

A Puzzle

- Knights and Liars: Knights always tell the truth; Liars always lie.
  - Zoe: "Mel is a liar"
  - Mel: "Neither I nor Zoe are liars“
- Who's lying?

A Puzzle

- Knights and Liars: Knights always tell the truth; Liars always lie.
  - Zoe: "Mel is a liar"
  - Mel: "Neither I nor Zoe are liars"

(1) $z \Leftrightarrow \sim m$

(2) $m \Leftrightarrow m \,\&\, z$

z	m	(1)	(2)	(1) ∧ (2)
T	T	F	T	F
T	F	T	T	T
F	T	T	F	F
F	F	F	T	F

Why Programming?

- Decision making

"It was a dark and stormy night. Three men, Alex, Bob & Carl, taking refuge from the rains, came to a hotel, one after another. One was a "knight": he always spoke the truth; another was a "liar": he always lied; and the third was a "knave": he alternated lying with speaking the truth.

When they arrived at the hotel, the manager was not at the desk. When she came to the front, she said she had only one room available, and will give it to the person who arrived first; the other two have to make do with the chairs in the lobby."

Why Programming?

- The conversation went like this:
  - *Alex: I came first.*
  - *Bob: No, he did not! I came first.*
  - *Carl: No, he did not! I came first.*
  - *Alex: No, he did not! I came first.*
  - *Carl:Well, Bob did not come second.*
  - *Bob:That's true!*
- Who came first? Who came second? Who came third?

Logic and Programming

- Logic forms a formal foundation for describing relationships between entities.
- In many cases, we can infer interesting consequences from these relationships.
- When the inference procedure is simple enough, the descriptions of the relationships can be seen as programs.
- The same set of relationships can be described in many ways: each resulting in a different "program".
- Logic Programming: a framework for describing relationships such that inferences can be done efficiently.

Logic and Programming

- Logic forms a formal foundation for describing relationships between entities.
- In many cases, we can infer interesting consequences from these relationships.
- When the inference procedure is simple enough, the descriptions of the relationships can be seen as programs.
- The same set of relationships can be described in many ways: each resulting in a different "program".
- Logic Programming: a framework for describing relationships such that inferences can be done efficiently.

Languages

- Languages:
  - Imperative = Turing machines
  - Functional Programming = lambda calculus
  - Logical Programming = first-order predicate calculus
- Prolog and its variants make up the most commonly used Logical programming languages.
  - One variant is XSB → developed here at Stony Brook.
  - Prolog systems: SWI Prolog, XSB Prolog, Sicstus, Yap Prolog, Ciao Prolog, GNU Prolog, etc.

Association for Logic Programming

- http://www.cs.nmsu.edu/ALP/
  - the current state of logic programming technology
- Many other groups (news://comp.lang.prolog)
- XSB: http://xsb.sourceforge.net
  - system with SLG-resolution, HiLog syntax, and unification factoring
- Yap: http://www.ncc.up.pt/~vsc/Yap/
- SWI Prolog: http://www.swi-prolog.org
  - Complete, ISO and Edinburgh standard, common optimizations, GC including atoms. Portable graphics, threads, constraints, comprehensive libraries for (semantic) web programming, Unicode, source-level debugger.

Reasoning in Artificial Intelligence

- Flexibility of reasoning is one of the key property of intelligence.
- Commonsense inference is defeasible in its nature; we are all capable of drawing conclusions, acting on them, and then retracting them if necessary in the face of new evidence.
- If Computer programs are to act intelligently, they will need to be similarly flexible.
- Goal: development of formal systems that describe commonsense flexibility

Flexible Reasoning Examples

- **Reiter, 1987:** Consider a statement *Birds fly. Tweety*, we are told, *is a bird*. From this, and the fact that birds fly, we conclude that: *Tweety can fly.*
- **This is defeasible:** Tweety may be an ostrich, a penguin, a bird with a broken wing, or a bird whose feet have been set in concrete.
- **Non-monotonic Inference:** on learning a new fact (that *Tweety has a broken wing*), we are forced to retract our conclusion (that he could fly).

Non-monotonic Logics

- **Non-monotonic Logic** is a logic in which the introduction of a new information (axioms) can invalidate old theorems.
- **Default reasoning** (logics) means drawing of plausible inferences from less-then- conclusive evidence in the absence of information to the contrary.
- Non-monotonic reasoning is an example of the default reasoning.

Auto-epistemic reasoning

- **Moore, 1983:** Consider my reason for believing that I do not have an older brother. It is surely not that one of my parents once casually remarked, You know, you don't have any older brothers, nor have I pieced it together by carefully sifting other evidence.
- I simply believe that if I did have an older brother I would know about it; therefore, since I don't know of any older brothers of mine, I must not have any.

Auto-epistemic reasoning

- "The brother" reasoning is not a form of default reasoning nor non-monotonic. It is reasoning about one's own knowledge or belief. Hence it is called an **auto-epistemic reasoning**.
- Auto-epistemic reasoning models the reasoning of an ideally rational agent reflecting upon his beliefs or knowledge.
  - Auto- Logics are logics which describe the reasoning of an ideally rational agent reflecting upon his beliefs.

Missionaries and Cannibals

- **McCarthy, 1985** revisits the problem: Three missionaries and three cannibals come to a river. A rowboat that seats two is available. If the cannibals ever outnumber the missionaries on either bank of the river, the missionaries will be eaten. How shall they cross the river?
- Traditionally the puzzler is expected to devise a strategy of rowing the boat back and forth that gets them all across and avoids the disaster.

Missionaries and Cannibals

- Traditional Solution: A state is a triple comprising the number of missionaries, cannibals and boats on the starting bank of the river:
  - The initial state is 331, the desired state is 000,
  - A solution is given by the sequence: 331, 220,321, 300,311, 110, 221, 020, 031, 010, 021, 000.

Missionaries and Cannibals

- Imagine now giving someone a problem, and after he puzzles for a while, he suggests going upstream half a mile and crossing on a bridge.
  - What a bridge? you say. No bridge is mentioned in the statement of the problem.
  - He replies: Well, they don't say the isn't a bridge. Open world assumption!

Missionaries and Cannibals

- So you modify the problem to exclude the bridges and pose it again.
- He proposes a helicopter, and after you exclude that, he proposes a winged horse or that the others hang onto the outside of the boat while two row.
- He also attacks your solution on the grounds that the boat might have a leak or lack oars.

Missionaries and Cannibals

- Finally, you must look for a mode of reasoning that will settle his hash once and for all (Closed world assumption!)
- McCarthy proposed circumscription
  - He argues that it is a part of common knowledge that a boat can be used to cross the river unless there is something with it or something else prevents using it.
  - If our facts do not require that there be something that prevents crossing the river, circumscription will generate the conjecture that there isn't.
- Lifschits has shown in 1987 that in some special cases the circumscription is equivalent to a first order sentence

Logic Programming

- Logic Programming encompasses many types of logic:
  - Horn clauses,
  - Non-monotonic,
  - Constraint solving,
  - Satisfiability checking,
  - Knowledge Representation-Object-oriented
  - Inductive logic programming,
  - Transaction Logic, etc.

https://en.wikipedia.org/wiki/Logic_programming

Applications

- Many: Deductive databases, Model checking, Declarative networking, Configuration systems, etc.
- Where? International Space Station, IBM Watson, US Border Control, Windows user access, etc.
- Conferences: International Conference on Logic Programming (ICLP), International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR), International Web Rule Symposium (RuleML) 2016 in Stony Brook, International Conference on Web Reasoning and Rule Systems (RR), etc.

Knowledge Systems Lab, Stony Brook Univ.

Paul Fodor, Michael Kifer, IV Ramakrishnan, CR Ramakrishnan, David S. Warren, Annie Liu

Stony Brook University

- Logic Programming and Deductive databases
- XSB Prolog (25+ years of research at Stony Brook)
  - http://xsb.sourceforge.net and Flora-2, LMC, ETALIS, Ergo, …
- Knowledge Representation & Processing (decision support)
- Research Interests and Projects:
  - **Logic programming: Transaction Logic, F-logic, HiLog, LPDA**
  - Knowledge representation
  - NLP, NLU : IBM Watson Question Analysis with Prolog, Project Halo (Vulcan Inc.) SILK
  - Rule systems benchmarking: OpenRuleBench
  - Stream processing: ETALIS/EP-SPARQL
  - Access control policies and trust management languages
  - Semantic Web
  - Virtual expert systems , …

Introduction: Logic Programming, Prolog, Datalog: Facts, Rules and Queries

	FOL:	Prolog
Socrates is a man.		*man(socrates).*
All men are mortal.	$\forall x$, man(x)→ mortal(x).	*mortal(X) :- man(X).*
Is Socrates mortal?		*?- mortal(X).*

Yes: X=socrates

- Prolog has goal directed top-down resolution
- The *not* (\+) operator is a *closed-world* ***negation as failure:*** if no proof can be found for the fact, then the negative goal succeeds.
  - Example: *illegal(X) :- \+ legal(X).*
  - Adding a fact that something is legal destroys an argument that it is illegal.

Prolog's "Yes" means "I can prove it", while Prolog's "No" means "I can't prove it"

Logic Programming Extensions at Stony Brook

- Prolog pitfalls:
  - redundant computations
  - non-termination of otherwise correct programs

```
path (A , B): - edge (A , C), path (C , B).
path (A , B): - edge (A , B).
```

  - not OO, not defeasible, closed world assumption, …
- Goal: Realize the vision of logic-based knowledge representation with frames, defeasibility, meta, and side-effects, event streams, …
  - Tabling (efficiency, termination, Datalog and well-founded semantics),
  - F-logic (frames, path expressions and reification),
  - Logic programming with defaults and argumentation theories,
  - HiLog,
  - Transaction Logic (and tabling for WFS),
  - Event Condition Action rules and Complex Event Processing (ETALIS) (complex events, aggregates, consumption policies, time and count windows)

Logic Programming Extensions at Stony Brook

Suspend computation when same goal is called again and Consume answers of producers. XSB is sound and complete for LP well-founded semantics.

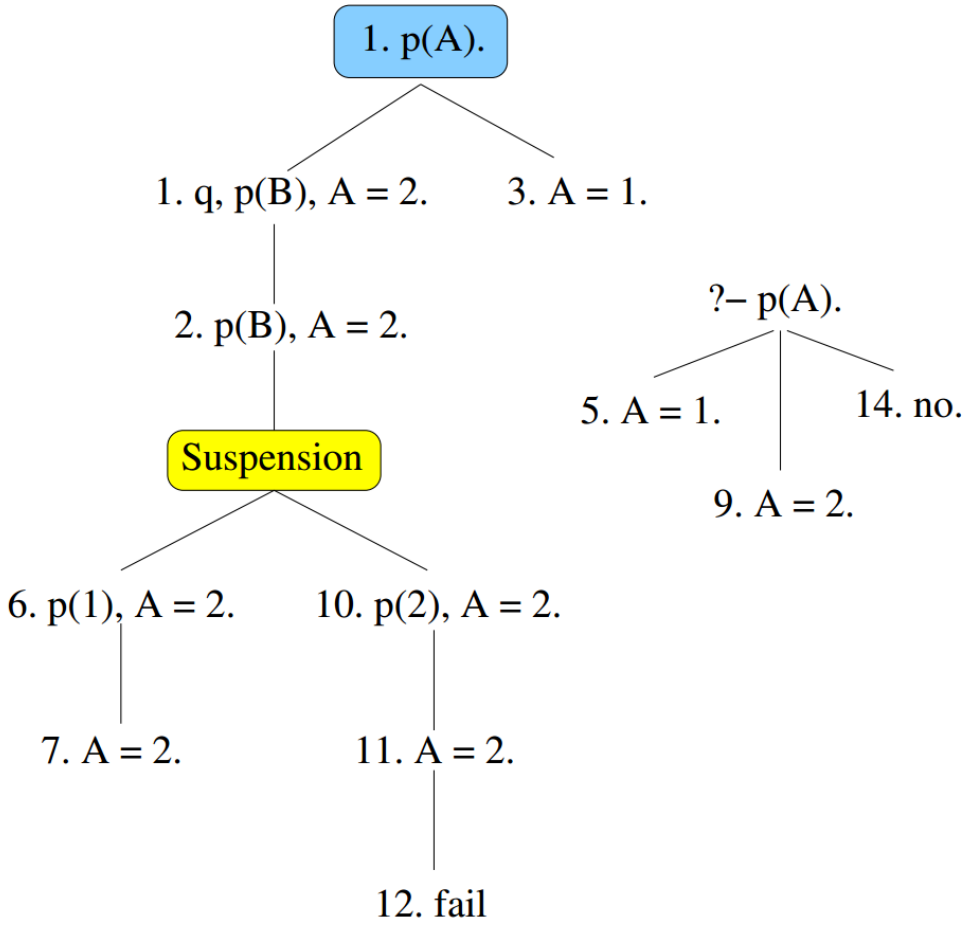
Example

```
:- table p/1.

p(A) :-
    q,
    p(B),
    A = 2.

p(A) :-
    A = 1.
```

Subgoal	Answers
1. p(A)	4. A = 1 8. A = 2 13. Complete



Logic Programming Extensions at Stony Brook: F-Logic (Flora2)

Object Id

Attribute

Object description:

John[*name* –> 'John Doe', *phones* -> {6313214567, 6313214566},
children -> {Bob, Mary}]

Mary[*name* –> 'Mary Doe', *phones* -> {2121234567, 2121237645},
children -> {Anne, Alice}]

Structure can be nested:

Attribute

Sally[*spouse* -> John[*address* -> '123 Main St.']]

Methods: ?P[*ageAsOf*(?Year) –> ?Age] :-
?P:Person, ?P[*born* –> ?B], ?Age is ?Year–?B.

Type signatures: Person[*born* => integer,
ageAsOf(integer) => integer].

Logic Programming Extensions at Stony Brook: Transaction Logic

stack(*0*,*?X*).

stack(*?N*,*?X*) :- *?N*>0 ⊗ *move*(*?Y*,*?X*) ⊗ *stack*(*?N-1*,*?Y*).

move(*?X*,*?Y*) :- *pickup*(*?X*) ⊗ *putdown*(*?X*,*?Y*).

pickup(*?X*) :- *clear*(*?X*) ⊗ *on*(*?X*,*?Y*) ⊗ t_delete{*on*(*?X*,*?Y*)} ⊗ t_insert{*clear*(*?Y*)}.

putdown(*?X*,*?Y*) :- *wider*(*?Y*,*?X*) ⊗ *clear*(*?Y*) ⊗ t_insert{*on*(*?X*,*?Y*)} ⊗ t_delete{*clear*(*?Y*)}.

- Can express not only execution, but all kinds of sophisticated constraints:

?- *stack*(*10*, block43)

/\ ∀*?X*,*?Y* (*move*(*?X*,*?Y*) ⊗ *color*(*?X*,*red*)) => (∃ *?Z* *color*(*?Z*,*blue*) ⊗ *move*(*?Z*,*?X*))

Whenever a red block is stacked, the next block to be stacked must be blue

- Planning with Heuristics: Specifying STRIPS in Transaction Logic

achieve_unstack(*?X*,*?Y*) :-

(*achieve_clear*(*?X*) * *achieve_on*(*?X*,*?Y*) * *achieve_handempty*)

⊗ *unstack*(*?X*,*?Y*).

- Tabling to stop infinite computation paths and defeasibility (ICLP2009)

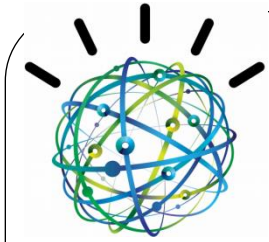
Logic Programming Extensions at Stony Brook: Defeasibility

- Common sense reasoning: rules can be true by default but may be defeated (policies, regulations, law, inductive/scientific learning, natural language understanding): Logic Programming with Defaults and Argumentation theories LPDA (ICLP2009) and Transaction Logic LPDA (ICLP2011)

$$
\begin{array}{l}
buy : -pay \otimes delivery.\\
@b1 delivery : -gold\_member \otimes express\_mail.\\
@b2 delivery : -ground\_mail.\\
@b3 pay : -pay\_credit\_card.\\
@b4 pay : -pay\_cheque.\\
!\mathbf{opposes}(b1, b2).!\mathbf{overrides}(b1, b2).!\mathbf{opposes}(b4, b3).!\mathbf{overrides}(b4, b3).\\
express\_mail : -insert(delivered\_express\_mail).\\
ground\_mail : -insert(delivered\_ground\_mail).\\
pay\_credit\_card : -credit\_card\_credentials \otimes insert(credit\_card\_payment).\\
pay\_cheque : -bank\_account \otimes insert(bank\_payment).\\
credit\_card\_credentials.bank\_account.gold\_member.
\end{array}
$$

Argumentation theory:

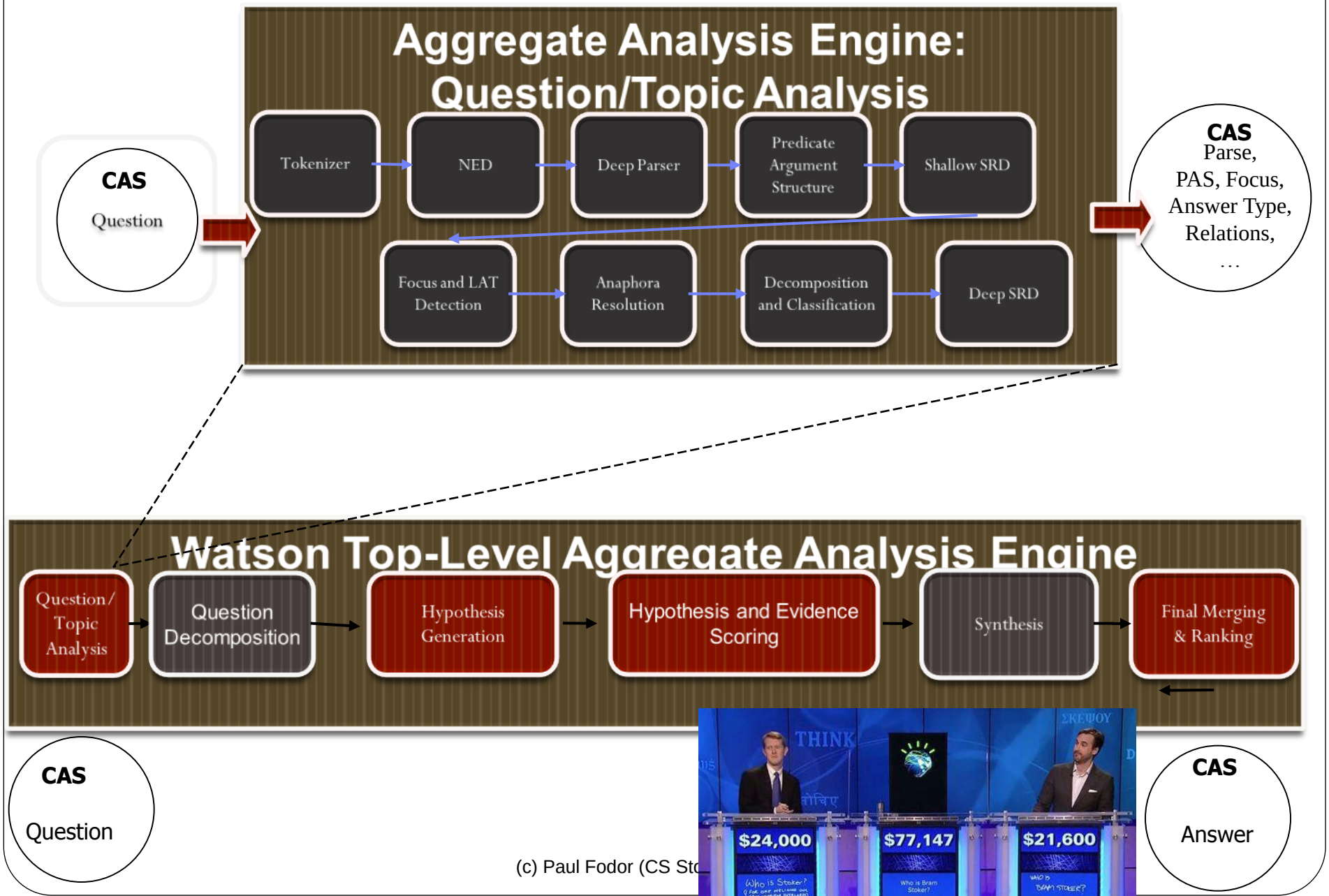
$$
\begin{array}{lll}
\$\mathbf{defeated}(R) & :- & \$refutes(S,R) \wedge \text{not } \$compromised(S).\\
\$\mathbf{defeated}(R) & :- & \$rebuts(S,R) \wedge \text{not } \$compromised(S).\\
\$\mathbf{defeated}(R) & :- & \$disqualified(R).\\
\$refutes(R,S) & :- & \$conflict(R,S) \wedge !\mathbf{overrides}(R,S).\\
\$rebuts(R,S) & :- & \$candidate(R) \wedge \$candidate(S) \wedge\\
 & & !\mathbf{opposes}(R,S) \wedge \text{not } \$compromised(R) \wedge\\
 & & \text{not } \$refutes(\_,R) \wedge \text{not } \$refutes(\_,S).\\
\$compromised(R) & :- & \$refuted(R) \wedge \$\mathbf{defeated}(R).\\
\$disqualified(X) & :- & \$defeats_{tc}(X,X).\\
\$defeats_{tc}(X,Y) & :- & \$defeats(X,Y).\\
\$defeats_{tc}(X,Y) & :- & \$defeats_{tc}(X,Z) \wedge \$defeats(Z,Y).\\
!\mathbf{opposes}(handle(\_,H), handle(\_, neg\ H)). & &
\end{array}
$$



Natural Language Processing with Prolog in the IBM Watson System

- Pattern Matching: question to candidate passages
  - Coding Pattern Matching Rules Directly in a Procedural Language Like Java is Not Convenient
  - Prolog: well-established standard; straightforward syntax; very expressive; development, debugging, and profiling tools exist; efficient, well-understood implementations, proven to be effective for pattern-matching tasks; natural fit for integration with UIMA (IBM R&D Journal 2012)
- We implemented Prolog rule sets for:
  - Focus Detection
  - Lexical Answer Type Detection
  - Shallow and Deep Relation Extraction
  - Question Classification
- Execution is Efficient to Compete At Jeopardy!
  - A Question is analyzed in a fraction of a second
- Open NLP tooling at Stony Brook University
  http://ewl.cewit.stonybrook.edu/sbnlp
  - + **Education:** Stony Brook University courses:
  Computers playing Jeopardy! (2011 - 2016)

Watson Question Analysis



Focus Detection Rules

- The focus is the "node" that refers to the unspecified answer.
- Pattern: WHAT IS X …?

  "What is the democratic party symbol?"

  "What is the longest river in the world?"

```
focus(QuestionRoot, [Pred]):-
    getDescendantNodes(QuestionRoot,Verb),
    lemmaForm(Verb,"be"),
    subj(Verb,Subj),
    lemmaForm(Subj,SubjString),
    whatWord(SubjString), % "what","which" ("this","these")
    pred(Verb,Pred),!.
```

- Pattern: "How much/many":

  "How many hexagons are on a soccer ball?"

  "How much does the capitol dome weigh?"

  "How much folic acid should an expectant mother get daily?"

```
focus(QuestionRoot, [Determiner]):-
    getDescendantNodes(QuestionRoot,Determiner),
    lemmaForm(Determiner,DeterminerString),
    howMuchMany(DeterminerString),!. % "how much/many", "this much"
```

Answer-type Computation Rules

- Time rule (e.g. when): Pattern: When VERB OBJ; OBJ VERB then

Example: When was the US capitol built? answerType => ["com.ibm.hutt.Year"]

```
answerType(_QuestionRoot,FocusList,timeAnswerType,ATList):-
        member(Mod,FocusList),
        lemmaForm(Mod,ModString),
        wh_time(ModString), % "when", "then"
        whadv(Verb,Mod),
        lemmaForm(Verb,VerbString),
        timeTableLookup(VerbString,ATList),!.
```

- "How ... VERB" rule: Pattern: How ... VERB?

Example: "How did Virginia Woolf die?" answerType => ["com.ibm.hutt.Disease", "com.ibm.hutt.MannerOfKilling", "com.ibm.hutt.TypeOfInjury"]

```
answerType(_QuestionRoot,FocusList,howVerb1,ATList):-
        member(Mod,FocusList),
        lemmaForm(Mod,"how"),
        whadv(Verb,Mod),
        lemmaForm(Verb,VerbString),
        howVerbTableLookup(VerbString,ATList), !.
```

Answer-type Computation Rules

Focus lexicalization (lexical chains using Prolog WordNet followed by a mapping to our taxonomy)

Question	QParse 2 AnswerType
What American revolutionary general turned over West Point to the British?	[com.ibm.hutt.MilitaryLeader]

Table lookup for the verb:

Question	QParse 2 AnswerType
How did Jimi Hendrix die?	[com.ibm.hutt.Disease com.ibm.hutt.MannerOfKilling com.ibm.hutt.TypeOfInjury]

Table lookup for the focus:

Question	QParse 2 AnswerType
How far is it from the pitcher's mound to home plate?	[com.ibm.hutt.Length]
When was Lyndon B Johnson president?	[com.ibm.hutt.Year]

Table lookup for the focus (noun) + the verb:

Question	QParse 2 AnswerType
What instrument measures radioactivity?	[com.ibm.hutt.Tool]
What instrument did Louis Armstrong play?	[com.ibm.hutt.MusicalInstrument]

Answer-type Computation Rules

- Cascading rules in order of generality
  - first rule that fires returns the most specific answer-type for the question

Look at the focus + verb:

Question	QParse 2 AnswerType
How much did Marilyn Monroe weigh?	[com.ibm.hutt.Weight]
How much did the first Barbie cost?	[com.ibm.hutt.Money]

Look at the focus + noun:

Question	QParse 2 AnswerType
How many Earth days does it take for Mars to orbit the sun?	[com.ibm.hutt.Duration]
How many people visited Disneyland in 1999?	[com.ibm.hutt.Population]

Look only at the focus:

Question	QParse 2 AnswerType
How many moons does Venus have?	[com.ibm.hutt.WholeNumber]
How much calcium is in broccoli?	[com.ibm.hutt.Number]

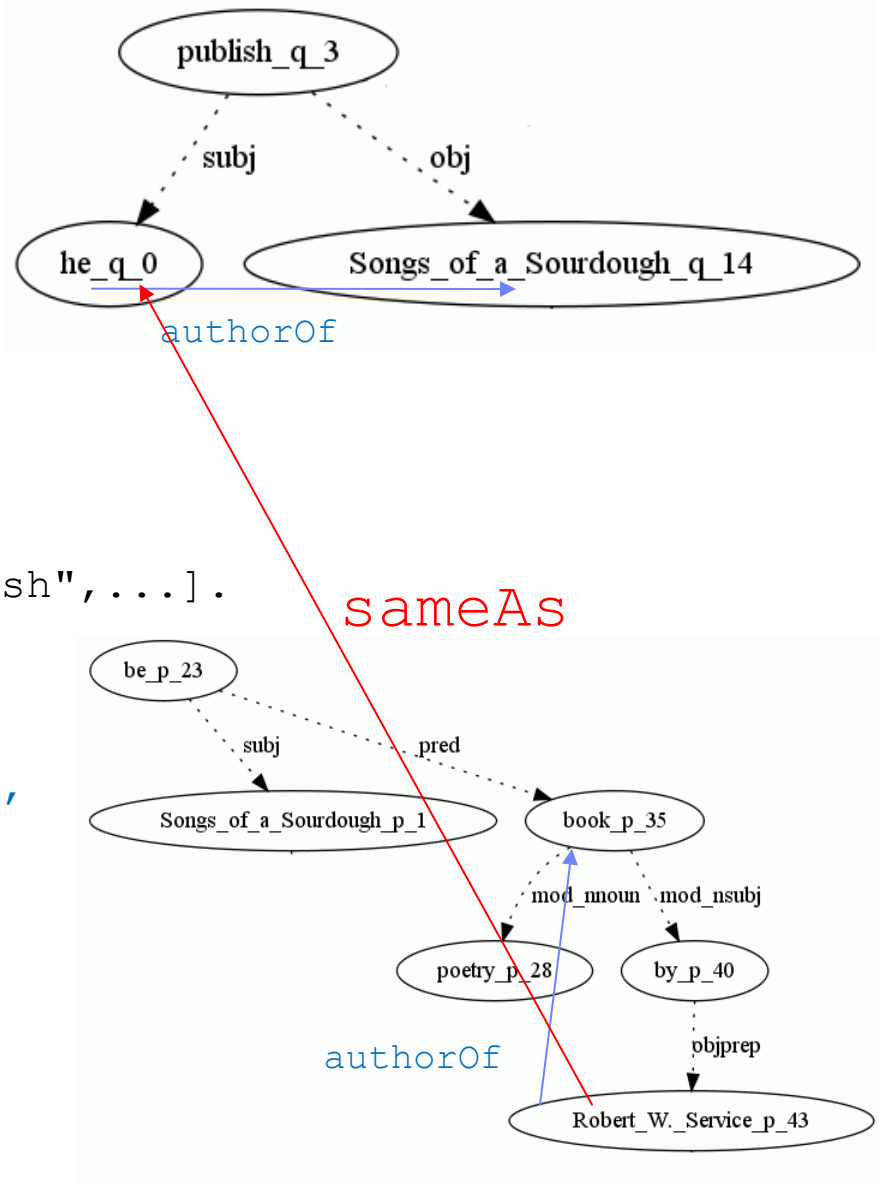
Priority decreases down the chain

Relation Detection Rules

```
authorOf(Author,Composition) :-
  authorVerb(Verb),
  subj(Verb,Author),
  validAuthor(Author),
  obj(Verb,Composition),
  validComposition(Composition).
authorVerb(Verb) :-
  partOfSpeech(Verb,verb),
  lemma(Verb,VerbLemma),
  member(VerbLemma, ["write","publish",...].

authorOf(Author,Composition) :-
  validComposition(Composition),
  argument(Composition,Preposition),
  lemma(Preposition, "by"),
  objprep(Preposition,Author),
  validAuthor(Author).


sameAs(X,Z) :-
  authorOf(X,Y),
  authorOf(Z,Y).
```



ETALIS/ EP-SPARQL Complex Event and Stream Processing

- Data-driven continuous complex event processing:
  - Event filtering, enrichment, projection, translation, and multiplication
  - Declarative semantics
  - Combines detection of complex events and reasoning over states
  - Sliding windows (time and count-based)
  - Aggregation over events (count, avg, sum, min, max, user-defined aggregates)
  - Processing of out-of-order events
- Visual development for sequential and aggregative patterns
- Open source: http://code.google.com/p/etalis
- Uses: stock market, health applications, transit applications, NLP streaming applications (Twitter posts analysis)
  - The Ford OpenXC Challenge: map as weighted graph and update road weights from traffic events

"The Fast Flower Delivery Use Case", *accompanying the book "Event Processing In Action", by Opher Etzion and Peter Niblett, Manning Publications*

```
% Phase 1: Bid Phase
% Multiplier: multiply the event "delivery_request_enriched" for each driver
delivery_request_enriched_multiplied(DeliveryRequestId,DriverId,StoreId,ToCoordinates,DeliveryTime,
                MinRank)<-
      delivery_request_enriched(DeliveryRequestId,StoreId,ToCoordinates,
            DeliveryTime,MinRank) event_multiply driver_record(DriverId,_Ranking).
% gps_location_translated/3
gps_location_translated(DriverId,Rank,Region)<-
      gps_location(DriverId,coordinates(SNHemisphere,Latitude,EWHemisphere,Longitude)) where
            ( driver_record(DriverId,Rank),
                gps_to_region(coordinates(SNHemisphere,Latitude, EWHemisphere,Longitude),Region) ).
% bid_request/5
bid_request(DeliveryRequestId,DriverId,StoreId,ToCoordinates,DeliveryTime)<-
      delivery_request_enriched_multiplied(DeliveryRequestId,DriverId, StoreId,ToCoordinates,
                DeliveryTime, MinRank) and
      gps_location_translated(DriverId,Rank,Region)
      where MinRank <= Rank, gps_to_region(ToCoordinates,Region).
% Phase 2: Assignment Phase
startAssignment(DeliveryRequestId,StoreId,ToCoordinates, DeliveryTime) <-
      delivery_request_enriched(DeliveryRequestId,StoreId,ToCoordinates, DeliveryTime,_MinRank)
      where trigger(start_assignment_time(Time)).
assignment(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime,DriverId,ScheduledPickupTime)<-
      startAssignment(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime) and
      min(ScheduledPickupTime,
                delivery_bid(DeliveryRequestId,DriverId,CurrentCoordinates, ScheduledPickupTime) ).
```

OpenRuleBench: Analysis of the Performance of Rule Engines

Stony Brook University

- Performance tests: database tests (joins, indexing, inference), updates vs. querying, database recursion, default negation in the body, real-data tests (Mondial, DBLP, Wordnet, ontologies), AI puzzles.
- E.g., recursive stratified negation tests:

size	6000	6000	24000	24000
cyclic data	no	yes	no	yes
xsb	2.359	3.408	42.824	44.487
yap	1.875	3.148	43.510	43.452
dlv	20.274	31.346	365.136	438.008
drools	104.884	error	error	error
jess	64.000	error	1517.000	error
jena	21.007	37.692	387.268	415.376
owlim	8.666	13.314	174.968	195.825

- Systems tested: highly optimized Prolog-based systems (XSB, Yap, SWI), deductive databases (DLV, Iris, Ontobroker), rule engines for triples (Jena, BigOWLIM), production and reactive rule systems (Drools, Jess, Prova), knowledge base systems (CYC).

http://rulebench.semwebcentral.org