Interpreting and Reasoning with Simple Natural Language Sentences

Working Note 44
Peter Clark, Niranjan Balasubramanian
Allen Institute for Artificial Intelligence
March 2014

1. Introduction

1.1 Goals

Our goal with project Aristo is to have the computer automatically acquire knowledge for passing 4th grade science exams (primarily using NLP of relevant texts, such as the Barron's study guide), and perform appropriate reasoning to pass those exams. Earlier in February, we illustrated how some questions could be answered by applying a set of rewrite rules to sentences in the book until the rewritten sentences closely matched the question. (The rewrite rules actually operated on "extractions" - parse-like structures derived from book sentences). However, the extractions and rewrite rules had no semantics, used an idiosyncratic notation, and only supported a narrow notion of inference. Our goal here is to place that earlier work on a more formal (and more general) footing, converting the syntactic extractions to use a clear notation (probabilistic first-order logic) and applying inference to them. This document discusses some of the issues that arise in this, how this can be done, and some of the design decisions that appear. In particular, it addresses two questions:

- Part I: how (pragmatically) can we represent the meaning of NL sentences in logic
- Part II: how (pragmatically) can we reason with those expressions computationally

1.2 Syntactic Transformations

As background, we briefly describe the earlier February work. We:

1. generated a simple representation of the book's content by (automatically) converting sentences into parse-like extractions, e.g.,

   Sentence: "In the hot weather our bodies sweat."
   Extraction: ("our bodies" "sweat" "" "in hot weather")

   Sentence: "Animals must eat to get nutrients"
   Extraction: (("animal" "eat") "to"/ENABLES ("animal" "get" "nutrients"))

2. Doing the same for true/false questions, e.g.,

   Question: "Is it true that a human's body produces sweat is a way that an organism may adjust to hot temperatures?"
   Question Extraction: (("human body" "produce" "sweat") "is an example of" ("organism" "adjust to" "hot temperature"))

3. Manually writing down rewrite rules to allow extractions to be transformed. Rewrite rules are a mixture of syntactic manipulation, paraphrases/synonymy, and domain knowledge. Examples are:

"IF X causes Y THEN Y is a response to X"
((?x "causes" ?y) → (?y "is a response to" ?x))

"IF X sweats THEN X produces sweat"
((?x "sweat") → (?x "produces" "sweat"))

4. Finding the minimum cost sequence of transformations to convert a sentence extraction to a true/false question extraction. An example of a transformation sequence looks:

KB "In the hot weather our bodies sweat."
("our bodies" "sweat" "" "in hot weather")
→ ("hot weather" "causes" ("our bodies" "sweat"))
→ (("our bodies" "sweat") "is a response to" "hot weather")
→ (("our bodies" "produces" "sweat") "is a response to" "hot temperature")
→ (("our bodies" "produces" "sweat") "is an example of"
("our bodies" "responding to" "hot temperature"))
→ (("our bodies" "produces" "sweat") "is an example of"
("our bodies" "adjust to" "hot temperature"))
matches
(("human body" "produce" "sweat") "is an example of" ("organism" "adjust to" "hot temperature"))
QN "Is it true that a human's body produces sweat is a way that an organism may adjust to hot temperatures?"

Our goal now is to place this in a more formal and general framework. We start with discussing the general challenge of interpreting natural language, and then move on to specific representational and inference schemes.

Part I: how can we represent NL sentences in logic?

2. The Problem of Generics

Consider the statement

"Dogs chase cats"

Does this mean that:

- every dog is chasing some cat?
- a random dog will (likely) be chasing a random cat?
- if a dog is chasing something, that something will (likely) be a cat?
- if a cat is being chased, then the chaser is (likely) a dog?
- a random dog will (likely) spend at least part of its life chasing a cat?
- etc.

This is the problem of interpreting generics, i.e., identifying what a statement made with generalities (e.g., dogs, cats) means in terms of individuals (some specific dog, some specific cat). We might say the task is to determine the "implicative force" of the English statement (i.e., what it implies). It is a hard problem, with several theoretical proposals in the literature (e.g., [1-4]).

We won't attempt to solve this problem here, but we will adopt a position that can accommodate different solutions, as follows: We say that "Dogs chase cats" is associated with a scenario (actually, a collection of scenarios) in which some dog is chasing some cat. Let's call this scenario the "cat-chasing scenario". If there is a cat-chasing scenario, then a dog is chasing a cat. The cat-chasing scenario consists of a complex of individuals (a dog, a cat, a chasing event) and a set of relationships between them (e.g., the dog is the "agent" of the chasing; the cat is the "object" of the chasing). We don't know when and if that scenario may occur, but if it does, we say that some dog is chasing some cat. Conversely, if we see part of this complex in the world (e.g., a dog chasing something), we might conclude, with some probability, that what we're seeing is actually a cat-chasing scenario, and thus conclude (with some probability) that it is a cat that is being chased.

The idea of representing a scenario (a complex of facts that go together) is by no means a new idea in AI. It is essentially the same as Schank's "scripts" [5] and Minsky's frames [6], and suggests that we can use a collection of stereotypical situations - expectations about likely configurations of the world - to both interpret the incomplete and noisy data that we see, and fill in the gaps.

The scenario/script idea is also closely related to the idea of prototypes in cognitive science (see [7-9] for an interesting AI implementation of this theme). If a prototype is described by a set of assertions, two useful notions from that literature are:

- an assertion's *cue validity*: how strongly an assertion implies the scenario
- an assertion's *salience*: how strongly the scenario implies the assertion

These strengths might, for example, be represented as probabilities in a probabilistic framework.

Given "cats chase dogs", we can write the possible inferences that suggest the cat-chasing scenario as follows (with guestimates on probabilities):

dog → dog chases cat	0.01
cat → dog chases cat	0.01
chase → dog chases cat	0.01
dog chasing → dog chases cat	0.9
chased cat → dog chases cat	0.8

This is one way of expressing what "dogs chase cats" might mean. Really, though, what the sentence means is that.

some-appropriate-context → dog chases cat

i.e., in some suitable context, we would expect (with some probability) the dog to chase the cat. For example, the cat and dog are outside, the dog has seen the cat, the dog isn't on a leash, etc. Identifying this unstated context is very difficult, and not something we address now. Rather, we will just work with the simpler implications above, where some part implies the whole. The probabilities can be viewed as a shallow expression of the likelihood of the (unstated) deeper context in which the implication holds.

3. A Pragmatic Interpretation of Extractions

3.1 From Extractions to Implications

In the enumeration of possible implications above, where do the probabilities come from? We might guess them, but often there are linguistic cues and/or linguistic conventions that suggest which are the

"stronger" implications. For example, our linguistic protocol is to read a sentence like "dogs have tails" as universally quantifying over dogs. Similarly in a sentence like "In hot weather, people sweat", the fronted prepositional phrase suggests there's a stronger implication from "hot weather" to "people sweat", rather than the other way round.

We could enumerate all possible implications from scenario parts-to-wholes such as above and manually assign probabilities, but that is somewhat labor intensive. Rather, we will make some pragmatic assumptions about what the stronger implication(s) are, and only write those down, as follows:

A. A simple triple

Given a simple triple (tuple) extracted from text of the form subject-verb-object:

(S V O)

it will become, informally:

S → (S V O) with probability P

e.g.

"dogs chase cats"

(dogs, chase, cats)

becomes:

% forall dogs, the dog chases a cat (with some probability)

∀D isa(D,dog) → ∃ Ch,C isa(Ch,chase), isa(C,cat), agent(Ch,D), object(Ch,C).

B. A triple-triple relation

Given a larger extraction structure of the form:

(Triple1 Relation Triple2)

this will become, structurally:

∀ Triple1 → ∃ Triple1 Relation Triple2 with probability P1

and

∀ Triple2 → ∃ Triple2 Relation Triple1 with probability P2

e.g.,

"growing fur causes animals to keep warm"

(("animal" "grow" "fur") "causes" ("animal" "keep" "warm"))

becomes:

% if an animal grows fur, this causes that animal to keep warm

∀ A G F isa(A,animal), isa(G,grow), isa(F,fur), agent(G,A), object(G,F)
→ ∃ K isa(K,keep), causes(G,K), agent(K,A), object(K,warm).

and

% if an animal keeps warm, this was caused by the animal growing fur

∀ A K isa(A,animal), isa(K,keep), agent(K,A), object(K,warm[1])
→ ∃ G F isa(G,grow), isa(F,fur), causes(G,K), agent(G,A), object(G,F).

C. A word-triple relation

A third type of extraction structure looks:

(Word Relation Triple)

and becomes

∀ Word → ∃ Word Relation Triple with probability P1

∀ Triple → ∃ Word Relation Triple with probability P2

e.g.,

"A hand lens is used to view small objects"

[1] As a side-note, we are treating adjectives, e..g, "warm", as constants here

("hand lens" PURPOSE ("" "view" "small objects"))
becomes
∀ H isa(H,hand_lens) → ∃ V,S isa(V,view), isa(S,small_object), purpose(H,V), object(V,S).
and
∀ V,S isa(V,view), isa(S,small_object), object(V,S) → ∃ H isa(H,hand_lens), purpose(H,V).

3.2 Simplifying the notation

To simplify the logical notation, we can drop the quantifiers from the expressions (leave them implicit). Note that this implicit quantification is different to that in Prolog:

- **Here:** Variables in antecedent are universally quantified, all other variables are existentially quantified
- **Prolog:** All variables are universally quantified.

3.3 Ontology, Concepts, and Strings

We've so far skirted the issue of mapping from words to concepts (word sense disambiguation). For now, let's make a one-sense-per-word assumption, i.e., that a concept name is simply the word name. We could use either a string or a symbol for the concept name, as we like (it doesn't matter):

isa(D,"dog") → isa(T,"tail"), has(D,T).

To compute subsumption (isa relationships), we can use the entailment engine to perform equivalences and taxonomic inferences, e.g., given

isa(dog01,"the dog").

and the query:

| ?- isa(dog01, "some animal")

we can answer the query using the entailment engine:

| ?- entails("the dog", "some animal").

i.e., applying the general rule:

isa(X,Y) ← isa(X,Z), entails(Z,Y).

A non-zero entailment corresponds to a "yes" answer. In this way, we can side-step word sense disambiguation, and tolerate slight mismatches in the strings used to name concepts.

3.4 Semantic Roles

We are assuming a naive approach to semantic role labeling (SRL). For example, in the extraction:

(X, R, Y, "PP1 Z1", "PP2 Z2", ...)
e.g., (“animals”, “grow”, “fur”, “in the winter”)

we will assume:
X is the agent
Y is the object
Z1 in "PP1 Z1" is the PP1

Z2 in "PP2 Z2" is the PP2

i.e., informally:
agent(grow,animals)
object(grow,fur)
in(grow,the_winter)

Similarly, for ((X Y Z) *relation* (A B C)) extractions, we treat *relation* as the predicate between Y and B, i.e.,

relation(Y,B)

Additional considerations for selecting semantic roles are:

- **Relational verbs:** Some verbs e.g., "contains", might better be treated as a relation rather than a reified event. We need a list of those.
- **Relational nouns:** Some nouns, e.g., "color", might better be treated as a relation rather than a type. We need a list of those also.

3.5 Constants

Let's treat adjectives (e.g., "tall"), and proper nouns (e.g., "Earth") as constants. Let's also treat mass nouns (e.g,. "water", "milk") as constants, in other words we leave them as generics. A few other common nouns that don't have natural instances (e.g., "the ground") should also be treated as constants rather than quantified over. Thus:

("cats", "drink", "milk")
becomes
isa(C,cat) → isa(D,drink), agent(D,C), object(D,milk).

rather than existentially quantifying over milk ("...→ ∃ M isa(M,milk)", meaning that M represents some "piece of milk" that is drunk).

3.6 A Worked Example

For now, we will ignore the issues of uncertainty, although at some point a calculus is needed for manipulating confidence measures. Let's now consider an example question from the 4th grade exam (the below true/false version is derived from the original multiple choice question):

Is a person producing perspiration an example of an organism adjusting to hot weather?

A natural decomposition of this question into SETUP + QUERY is below, where person01 is a Skolemized individual denoting the existentially quantified person (etc.):

SETUP:
% A person is producing perspiration
isa(person01,person).
isa(perspiration01,perspiration).
isa(produce01,produce).
agent(produce01,person01).
object(produce01,perspiration01).

QUERY:

```
| ?- agent(A,person01),
     isa(A,adjust_to),
     object(A,H),
     isa(H,hot_weather).
```

Let's also write down what should be in the KB. An extraction from Barrons states:

> In hot weather, people sweat.
> ("people" "sweat" "" "in hot weather")

Represented as logic, one of the derived implications looks:

> *% IF you sweat THEN you're sweating in hot weather [1]*
> isa(P,person), isa(S,sweat), agent(S,P) → isa(H,hot_weather), in(S,H). *% with some probability*

Let's also write down some of the required rewrite rules as logic statements:

> *% if you produce perspiration, then you're sweating [2]*
> isa(P,produce), isa(R,perspiration), agent(P,X), object(P,R) → isa(S,sweat), agent(S,P).
>
> *% IF A does X in I THEN A adjusts to I. [3]*
> agent(X,A), in(X,I) → isa(R,adjust_to), agent(R,A), object(R,I).

From these rules, we can conclude the answer to the question is "yes" by proving the declarative form of the question:

```
% A person is producing perspiration          % QUESTION SETUP
=> The person is sweating                     % From [2]
   isa(sweat01,sweat), agent(sweat01,person01)
=> The person is sweating in hot weather.     % From [1]
   isa(hot_weather01,hot_weather), in(sweat01,hot_weather01).
=> The person is adjusting to hot weather     % From [3]
   isa(adjust_to01,adjust_to), agent(adjust_to01,person01),object(adjust_to01,hot_weather01).
=> The answer to the question is "true"
```

This general approach captures some of the intuitions described earlier. Although it is a simple example, it illustrates some important general principles:

1. a sentence in the book describes a scenario (configuration of instances) true under certain (unstated) circumstances.
2. we can turn that scenario into implications of the form: *If* part-of-scenario *then* scenario (with some probability).
3. Although we don't know the probabilities, pragmatics of language suggest which implications are strongest and should be made explicit.
4. We can similarly turn a true/false question into a set of ground assertions (setup) and a query.
5. Standard reasoning methods can then be used to prove the query.

4. Logical Inference as Graph Operations

4.1 Graphical Representation of Rules

A helpful way of understanding these rules is to think of them as graph manipulation operations:

> IF some subgraph exists
> THEN some additional nodes and edges are implied (with some confidence)

For example, we can sketch:

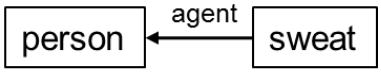
% IF you sweat THEN you're sweating in hot weather
isa(P,person), isa(S,sweat), agent(S,P) → isa(H,hot_weather), in(S,H).

as the graph:

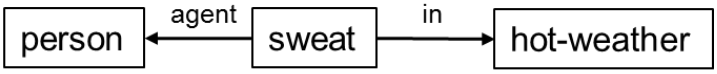
where the black (slightly darker shade, on black-and-white print) denotes what is known, and the red denotes what is inferred. For example, given an instance of person (e.g., person01) who is the agent of a sweat (e.g., sweat01), we can infer (with some probability) hot-weather. The inference step of concluding extra binary predicates can be sketched as adding extra nodes and edges to a graph, as follows. First, let's write down the setup:

> isa(person01,person)
> isa(sweat01,sweat)
> agent(sweat01,person01)

and sketch it as a graph:



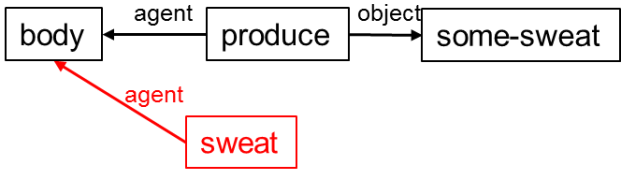
If we now apply the rule above, we will add an edge to this graph, resulting in:



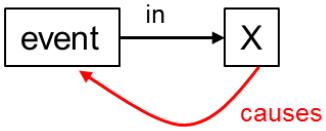
4.2 A Larger Example

Let's consider a larger example, and sketch the following rules, where again the black denotes the universally quantified condition, and the red denotes the existentially quantified conclusion:

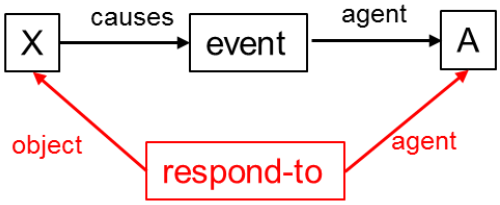
> *If a body produces sweat, then the body sweats.*
> isa(B,body), isa(P,produce), isa(SS,some-sweat), agent(P,B), object(P,SS)
> → isa(S,sweat), agent(S,B)



If an event occurs in X, then X causes the event
isa(E,event), in(E,X) → causes(X,E)



If X causes A to do event, then A is responding to X
isa(E,event), causes(X,E), agent(E,A) → isa(R,respond-to), agent(R,A), object(R,X).



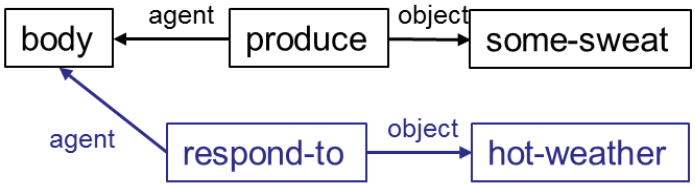
Let's now consider the question, drawing the setup in black and the query to prove in blue:

Is it true that a body producing sweat is an example of an organism responding to hot weather?
Setup:

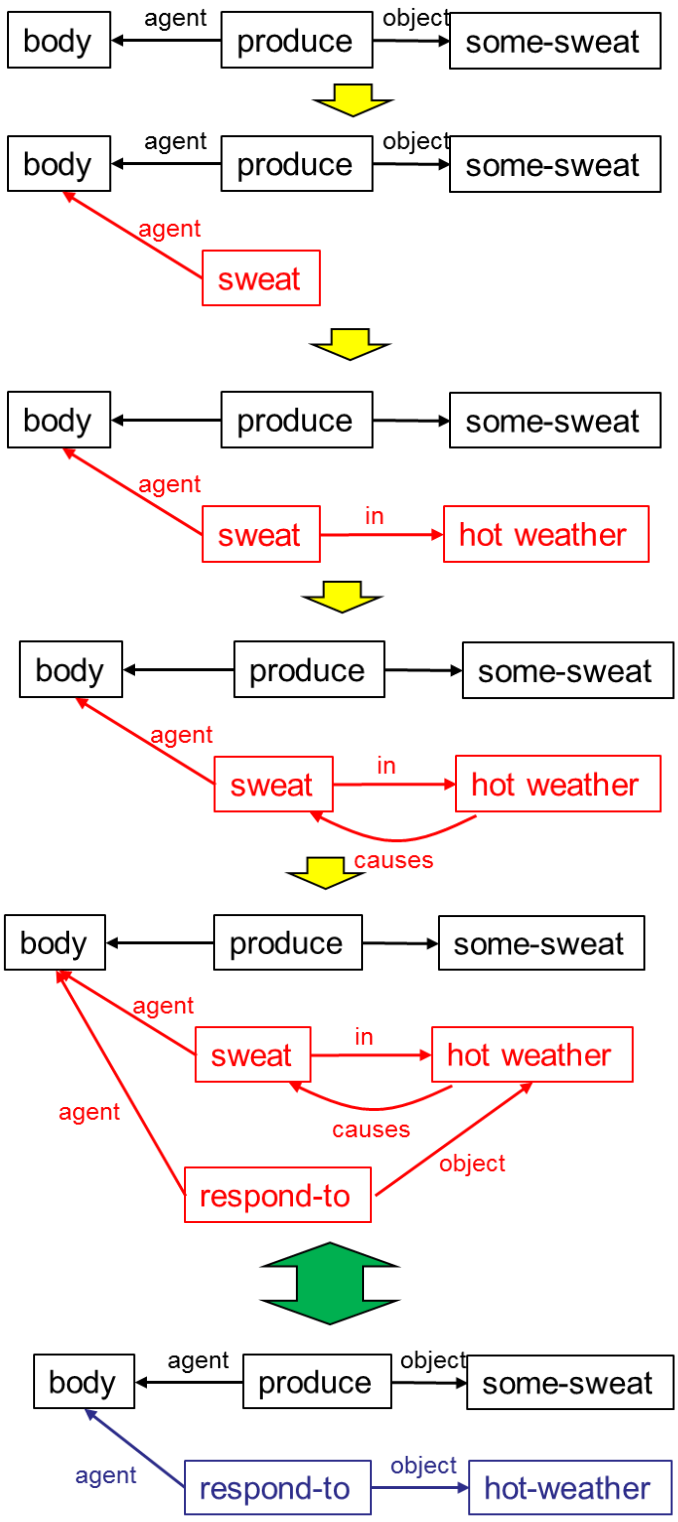
isa(body01,body)
isa(produce01,produce)
isa(sweat01,some-sweat)
agent(produce01,body01)
object(produce01,some-sweat01)

Query

| ?- agent(R,body01), isa(R,respond-to), object(R,H), isa(H,hot-weather)



Now we can sketch the incremental application of rules to the setup as the incremental growth of a graph, starting with the initial setup graph;



If you produce some sweat, then you sweat

Domain knowledge: bodies sweat in hot weather

If X happens in Y, then Y causes X

If X causes Y to do Z, then Y responds to X

This graphical depiction of what is going on is important, as it shows that something quite simple and intuitive is going on underneath the logical formulation. The use of graphical notations to express and

visualize logical inference has become quite important in recent years (e.g., the TextGraph workshops [10], Description Graphs [11,12] from the Description Logic community).

Part II: how can we reason with this computationally?

5. Technical Complications

5.1 Existential Quantification and Skolemization

Although this looks relatively straightforward, it turns out to be quite a nightmare to fit into some real-world inferencing systems (we've looked at ProbLog, but the issues are somewhat general). This Section attempts to explain why, and also how the issues can be addressed.

A key source of challenge is the existential quantification in the rules' conclusions. Let's take another simple example:

% If a person cooks then they produce food
∀ P,S isa(P,person), isa(S,cook), agent(S,P) →
    ∃ R,X isa(R,produce), isa(X,food), agent(R,P), object(R,X).

To put this in a Prolog-like environment, we need to Skolemize the existentially quantified variables in the conclusion. (This is already a potential problem for Markov Logic Networks as they assume a finite domain, and some implementations, e.g., Tuffy, don't allow terms as arguments). Let's name the two Skolem terms for the two existentially quantified variables above prod-sk1 (produce Skolem) and food-sk2 (food Skolem). The Skolemized rule looks:

% If a person cooks then they produce food
∀ P,S isa(P,person), isa(S,cook), agent(S,P) →
    isa(prod-sk1(P,S),produce), isa(food-sk2(P,S),food),
    agent(prod-sk1(P,S),P), object(prod-sk1(P,S),food-sk2(P,S)).

where the term prod-sk1(P,S) denotes a producing event, and food-sk2(P,S) denotes an instance of food. Now simplifying by dropping quantifiers and leaving the "isa" predicates implicit, we get:

% If a person cooks then they produce food
∀ P,S agent(S,P) → agent(prod-sk1(P,S),P), object(prod-sk1(P,S),food-sk2(P,S)).

So good so far. Now, suppose we are told that a person cooks:

agent(cook01,person01)

we can correctly conclude, using the rule, that the person is producing food:

agent(prod-sk1(person01,cook01), person01)
object(prod-sk1(person01,cook01), food-sk2(person01,cook01))

where prod-sk1(..) isa produce, and food-sk2(…) isa food.

Again, so good so far. But now, the practical problems arise if we already know some of the concluded facts to begin with. Suppose we had instead started with knowledge that person01 was already cooking and producing food:

```
agent(cook01,person01)
agent(produce01,person01)          % person01 is producing food01
object(produce01,food01).
```

If we now apply the rule, we get

```
agent(cook01,person01)
agent(produce01,person01)                          % person01 is producing food01[1]
object(produce01,food01).
agent(prod-sk1(person01,cook01), person01)    % person01 is producing food-sk2(…) [2]
object(prod-sk1(person01,cook01), food-sk2(person01,cook01))
```

The problem here is that now the person is apparently producing food twice (lines [1] and [2] above): In contrast, intuitively, we would prefer the inference system to recognize that the rule's conclusion is concluding something that is already known. The reason it does not is that, given the above, it is not strictly provable that the two produced foods are the same. We could add more knowledge to insist that the two foods are the same, e.g., by making produce() a functional predicate:

```
% Each produce event produces just one thing (i.e., if you produce B and C, B will equal C)
produce(A,B), produce(A,C) → B == C.
```

This could help a bit, but unfortunately we are now going beyond what a traditional Prolog environment will support, because Prolog does not allow different instances to be equalized. In other words, the rule above will conclude statements such as:

```
food01 == food-sk2(person01,cook01).
```

which is an illegal statement in Prolog because Prolog makes a Unique Names Assumption (UNA), meaning different names necessarily refer to different individuals. This implementational inconvenience is a major problem for translating such logic into a Prolog environment.

It is possible to work around this in Prolog by introducing a new equality predicate equals(X,Y):

```
produce(A,B), produce(A,C) → equals(B,C).
```

but then for this to have any meaning, every clause in the Prolog program needs to be rewritten to respect this equality, e.g.,

```
          f(X) → g(X)
becomes   f(X), equals(X,X2) → g(X2).
```

throughout the whole program. Such rewrites become much more complex with multiple variables, and the whole program becomes quite a mess.

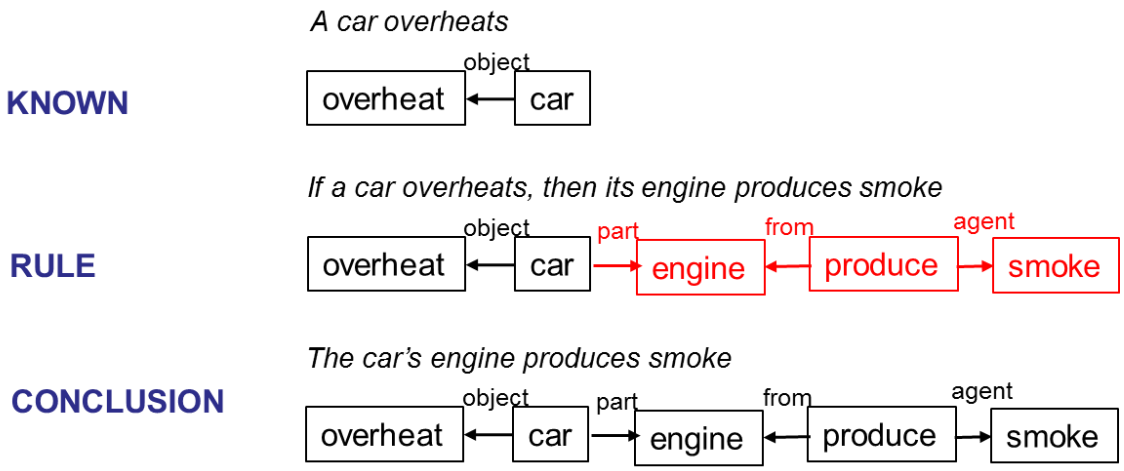
5.2 Matching Conclusions with What is Already Known

More generally, making some predicates functional like this is sometimes unwarranted and overly drastic. Rather, we would like formalize the notion that, informally:

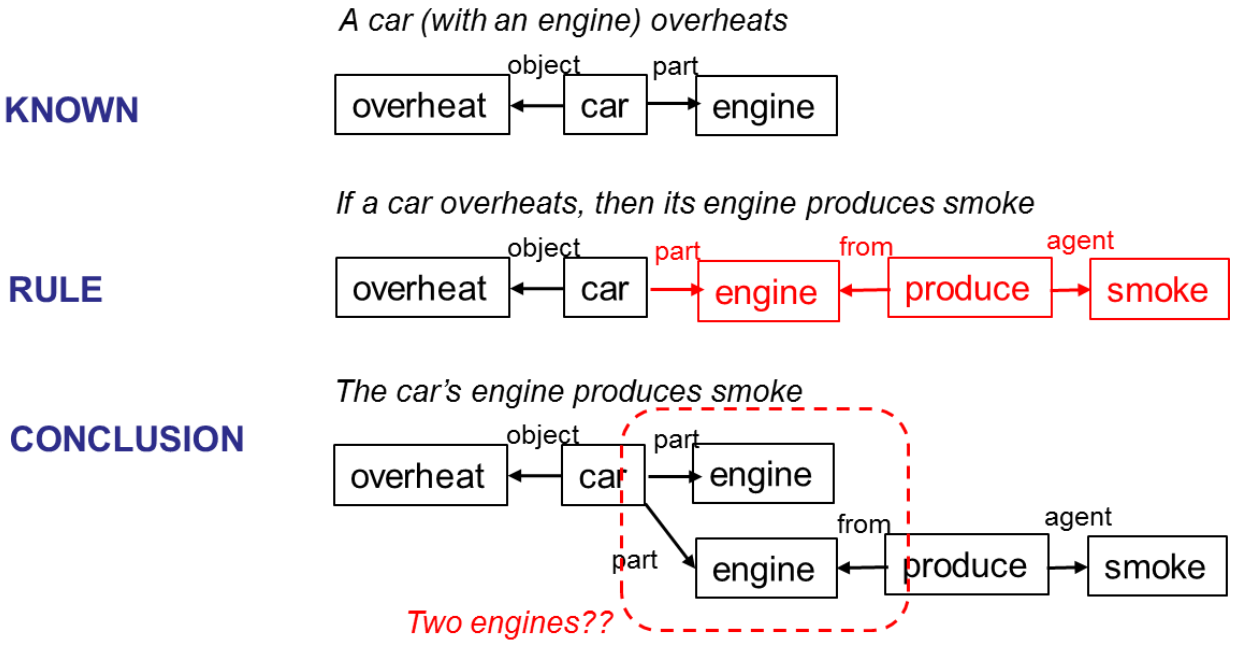
> IF a rule concludes something, and that something matches what you already know
> THEN that something ***is*** (probably) what you already know.

For example, a 4th grade question describes a car skidding and some tires producing smoke, and a definition of skidding talks about a car's tires slipping: although not strictly warranted, we would like to assume that the car tires (implied by the definition) are the same as the tires mentioned in the question.

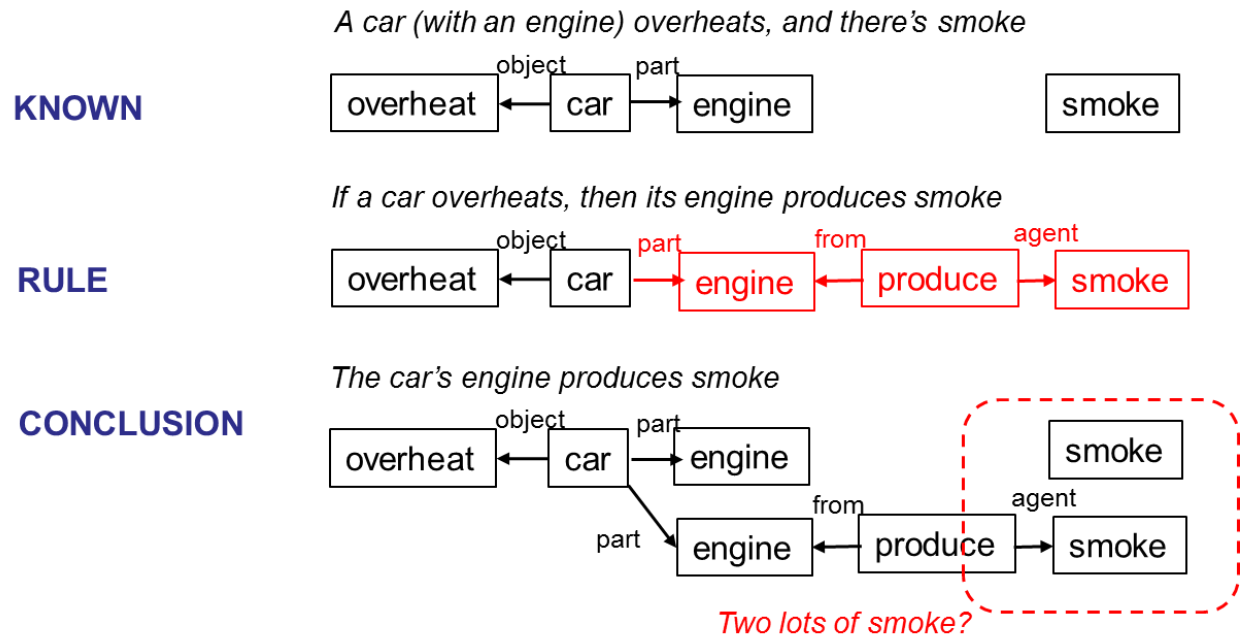
It is helpful to again think of this issue in graphical terms. Let's consider another example: Given a car is overheating, and a rule that the engine of an overheating car produces smoke, we can conclude that the car's engine produces smoke:



This is fine. But suppose we were instead also told at the start that the car also has an engine. The rule will conclude that an engine is smoking, but not necessarily the same engine as the one that was given:



Or suppose we're told there's an overheating car and some smoke; again we can't deductively conclude the smoke we see is the smoke from the overheating engine:



Again, although strictly this behavior is correct, one would like a stronger result, i.e., the evidence we see matches the conclusions we expect to see.

This kind of issue comes up frequently in NLP and commonsense reasoning, as natural language typically only describes fragments of, rather than all of, a scene – it is left to the reader to fill in the gaps.

5.3 Minimization and Coherence

One can think of the challenge of deciding when to "merge" concluded information with what is already known as a kind of minimization problem where, informally, the system should merge as much as possible without losing consistency. We can give a declarative characterization of this task as follows:

> If a concluded fact Fc subsumes a known fact Fk
> and it is consistent to assert Fc=Fk
> then Fc=Fk

Given a rule like this, there may be multiple alternative solutions (alternative sets of consistent conclusions) that can be derived from some initial facts and rules. We can add a second constraint to guide this, namely:

> The best (preferred) solution is the one which minimizes the number of new facts introduced

This is analogous to the declarative specification of circumscription (minimize the number of unexpected conclusions), and also analogous to other treatments of the frame problem. It is also essentially the same idea as Jerry Hobbs' notion of coherence in natural language interpretation [13] and his use of abductive reasoning in NL interpretation [14,15].

This kind of minimization seems important in commonsense reasoning, and it is not difficult to write a simple inference system that implements this, at least for small problems. It does, though, take reasoning outside the boundaries of what Prolog and ProbLog can do, and is a reasoning challenge that we will need to wrestle with going forward.

6. Summary

Our goal is to capture and reason with natural language sentences. In Part I, we described a treatment of generics in which a sentence is treated as describing a scene, and its interpretation consists of a set of implications from part of that scene to the whole with some probability. These can be expressed as simple logical implications with universal quantification in the condition, and existentials in the action. We also made some suggestions for the appropriate ontology to use, and described a graph-based way of thinking about what the inference rules were expressing.

In Part II we described some of the practical complications that have arisen trying to put this mechanism in a computational framework. In a Prolog-style formalism, existentials need to be Skolemized, but Prolog then does not allow equalities between different Skolems to be asserted. If we move to an alternative implementation which does support Skolem equality, a second issue still arises, namely rules produce a proliferation of new individuals because of their existential conclusions. This proliferation arises because, although often a rule's conclusion "obviously" is concluding what is already known, there is nothing in the logic that insists this is the case. We have described how we can handle this by adding an extra constraint that, if a rule's conclusion appears to repeat what is known, then it is repeating what is known, and a minimization rule to guide the search. This mechanism is analogous to solutions to the frame problem in AI (minimize the number of changes that an action may suggest), and may be a suitable approach to employ here also.

References

[1] Schubert, Lenhart K., and Francis Jeffry Pelletier. "Generically speaking, or, using discourse representation theory to interpret generics." *Properties, types and meaning*. Springer Netherlands, 1989. 193-268.

[2] Liebesman, David. "Simple generics." *Noûs* 45.3 (2011): 409-442.

[3] Schubert, Lenhart K., and Francis Jeffry Pelletier. "Problems in the representation of the logical form of generics, plurals, and mass nouns." *New directions in semantics* (1987): 385-451.

[4] Leslie, Sarah-Jane. "Generics and the structure of the mind." *Philosophical Perspectives* 21.1 (2007): 375-403.

[5] Schank, Roger C., and Robert P. Abelson. *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures*. Psychology Press, 2013.

[6] Minsky, Marvin. "A framework for representing knowledge." (1974).

[7] Porter, Bruce W., Ray Bareiss, and Robert C. Holte. "Concept learning and heuristic classification in weak-theory domains." *Artificial Intelligence* 45.1 (1990): 229-263.

[8] Bareiss, E., Bruce W. Porter, and Craig C. Wier. "Protos: An exemplar-based learning apprentice." *International Journal of Man-Machine Studies* 29.5 (1988): 549-561.

[9] Clark, Peter. "PROTOS-A Rational Reconstruction." (1987).

[10] Kozareva, Z., Matveeva, I., Melli, G., Nastase, V. (eds). *TextGraphs-8: Graph-based Methods for Natural Language Processing*. Workshop at the EMNLP 2013 conference. 2013. http://www.textgraphs.org/ws13

[11] Magka, Despoina, Boris Motik, and Ian Horrocks. "Modelling structured domains using description graphs and logic programming." *The Semantic Web: Research and Applications*. Springer Berlin. http://www.cs.man.ac.uk/~sattler/publications/structured-kr08.pdf

[12] Motik, Boris, et al. "Representing Structured Objects using Description Graphs." *KR*. 2008. Heidelberg, 2012. 330-344.http://www.cs.ox.ac.uk/boris.motik/pubs/mmh12dgs-and-lp.pdf

[13] Hobbs, Jerry R. "Coherence and coreference*." *Cognitive science* 3.1 (1979): 67-90.

[14] Hobbs, Jerry R., et al. "Interpretation as abduction." *Proceedings of the 26th annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 1988.

[15] Ovchinnikova, E., A. Gordon, J. R. Hobbs (2013). Abduction for Discourse Interpretation: A Probabilistic Framework. In Proceedings of the Joint Symposium on Semantic Processing, 42--50.