Parsers

-NLP parsers with prolog

Outline

- Natural Language Understanding in Prolog
- Prolog-Based Semantic Representations
- A Context-Free Parser in Prolog
- Probabilistic Parsers in Prolog
- A Context-Sensitive Parser in Prolog
- NLP With Prolog in IBM Watson System

Natural Language Understanding in Prolog

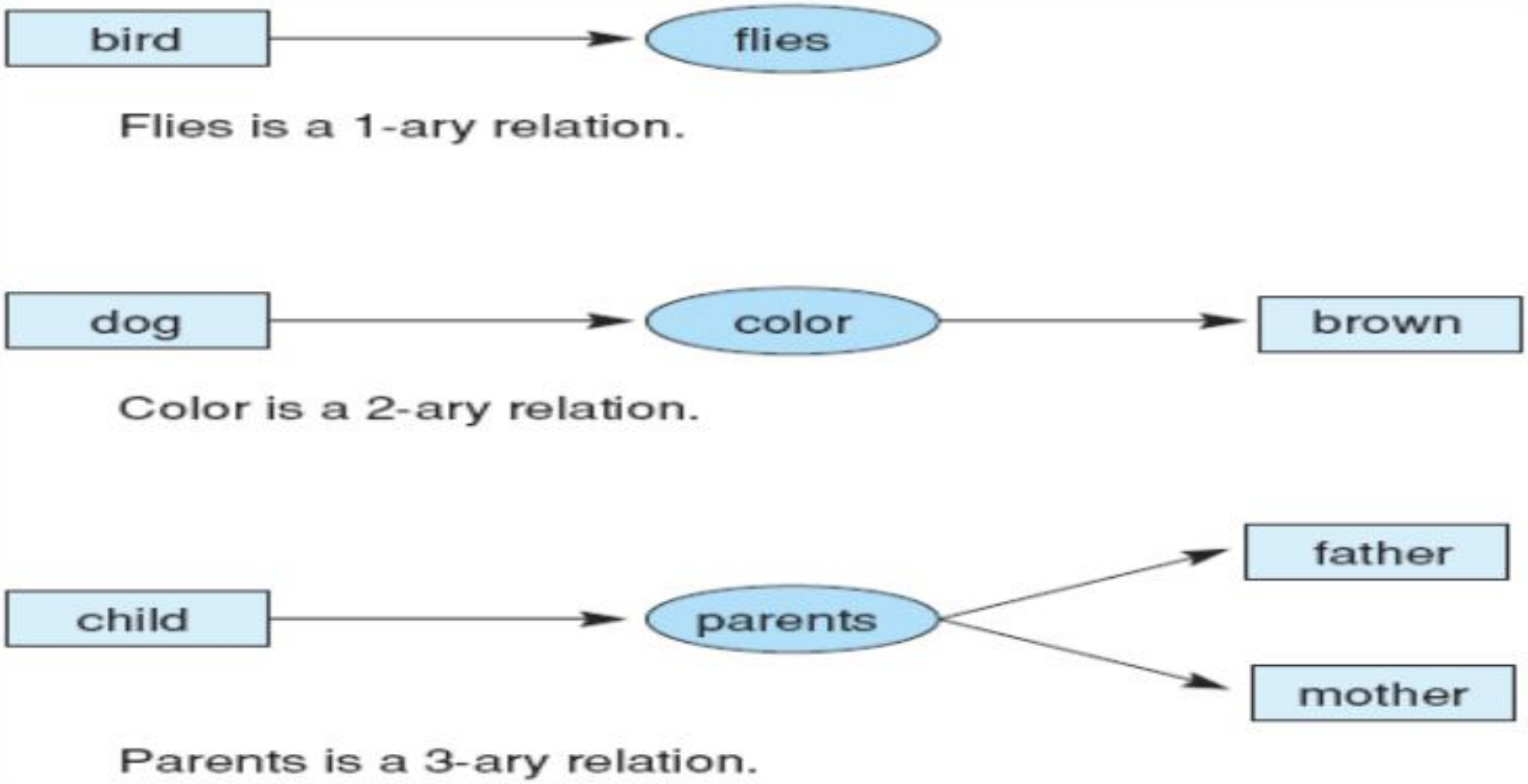
Because of its declarative semantics, built-in search, and pattern matching, Prolog provides an important tool for programs that process natural language.

Indeed, natural language understanding was one of Prolog's earliest applications. We can write natural language grammars directly in Prolog, for example, context-free, context-sensitive, recursive descent semantic network, as well as stochastic parsers.

Prolog-Based Semantic Representations

A **conceptual graph** is a finite, connected, bipartite graph. The nodes of the graph are either concepts or conceptual relations. Conceptual graphs do not use labeled arcs; instead the conceptual relation nodes represent relations between concepts. Because conceptual graphs are bipartite, concepts only have arcs to relations, and vice versa.

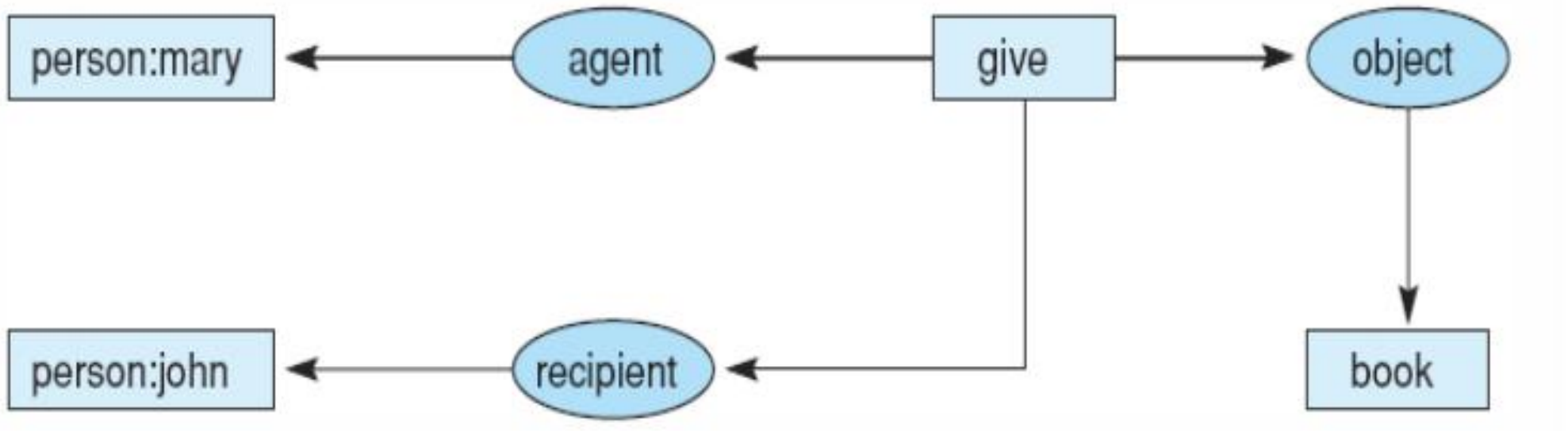
Prolog-Based Semantic Representations



Prolog-Based Semantic Representations

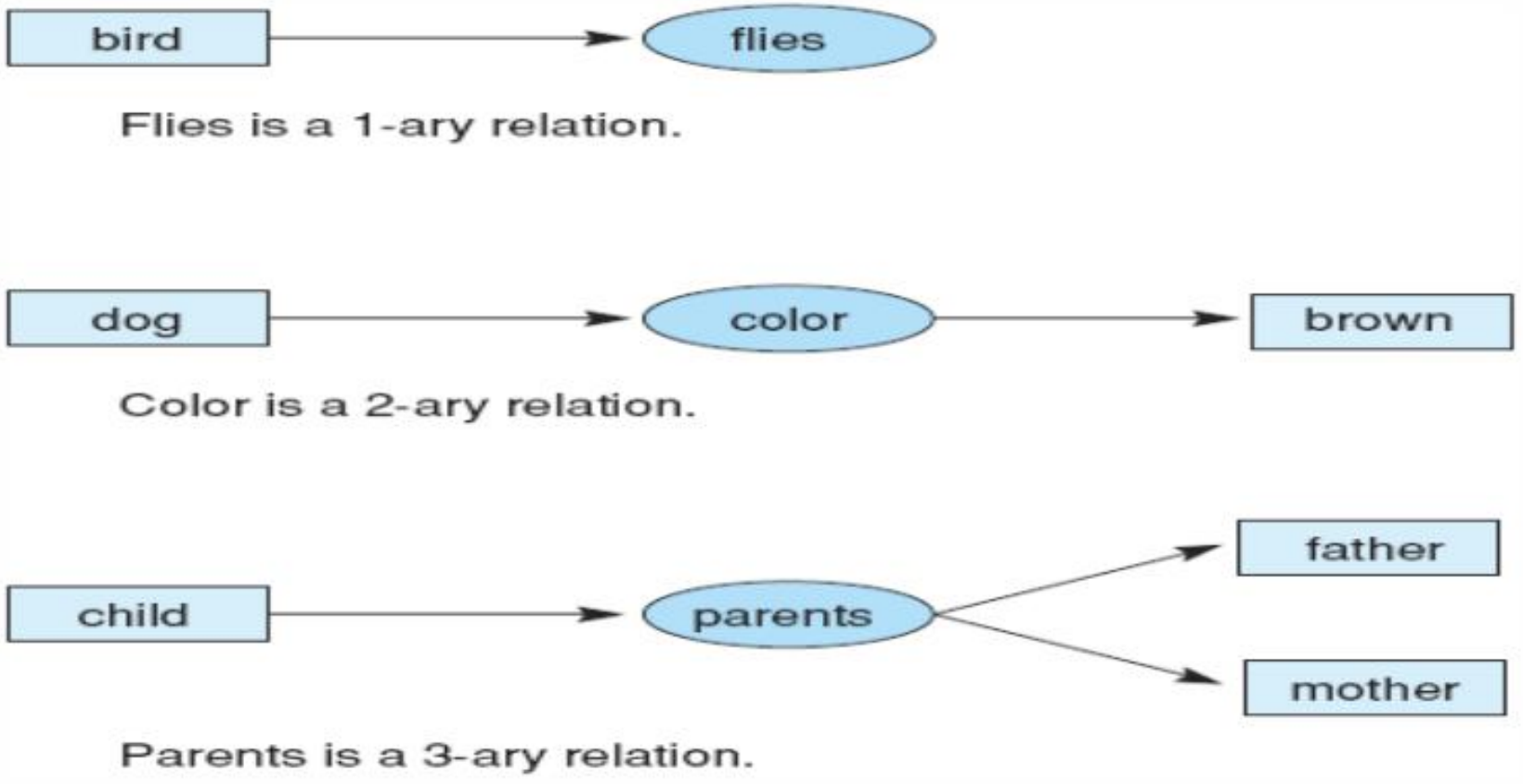
Conceptual graphs are used to model the semantics of natural language.

“Mary gave John the book.”



Prolog-Based Semantic Representations

Conceptual graphs can be translated directly into predicate calculus and hence into Prolog. The conceptual relation nodes become the predicate name, and the arity of the relation indicates the number of arguments of the predicate. Each Prolog predicate, as with each conceptual graph, represents a single proposition.



bird(X), flies(X).
dog(X), color (X, Y), brown(Y).
child(X), parents(X, Y, Z), father(Y), mother(Z).

Prolog-Based Semantic Representations

Logic programming also offers a powerful medium for building grammars as well as representations for semantic meanings. We next build recursive descent parsers in Prolog, and then add syntactic and semantic constraints to these parsers.

A Context-Free Parser in Prolog

Consider the subset of English grammar rules below. These rules are “declarative” in the sense that they simply define relationships among parts of speech.

Sentence <-> NounPhrase VerbPhrase

NounPhrase <-> Noun

NounPhrase <-> Article Noun

VerbPhrase <-> Verb

VerbPhrase <-> Verb NounPhrase

A Context-Free Parser in Prolog

Adding some vocabulary to the grammar rules:

Article(a)

Article(the)

Noun(man)

Noun(dog)

Verb(likes)

Verb(bites)

A Context-Free Parser in Prolog

These grammar rules have a natural fit to Prolog, for example, a sentence is a nounphrase followed by a verbphrase:

```
sentence(Start, End) :-
 nounphrase(Start, Rest), verbphrase(Rest, End).
```

A Context-Free Parser in Prolog

```
sentence(Start, End) :-
 nounphrase(Start, Rest), verbphrase(Rest, End).
```

This **sentence** Prolog rule takes two parameters, each a list. The rule attempts to determine whether some initial part of **Start** list is a NounPhrase. Any remaining tail of the NounPhrase list will match the **Rest** parameter and be passed to the first parameter of the verbphrase predicate. Any symbols that remain after the verbphrase check are passed back as the **End** argument of sentence.

A Context-Free Parser in Prolog

If the original list is a sentence, the second argument of sentence must be empty, [].

```
utterance(X) :- sentence(X, [ ]).
sentence(Start, End) :-
    nounphrase(Start, Rest), verbphrase(Rest, End).
 nounphrase([Noun | End], End) :-
    noun(Noun).
 nounphrase([Article, Noun | End], End) :-
    article(Article), noun(Noun).
```

A Context-Free Parser in Prolog

```
verbphrase([Verb | End], End) :-
    verb(Verb).
verbphrase([Verb | Rest], End) :-
    verb(Verb), nounphrase(Rest, End).
article(a).
article(the).
noun(man).
noun(dog).
verb(likes).
verb(bites).
```

A Context-Free Parser in Prolog

Example sentences may be tested for correctness:

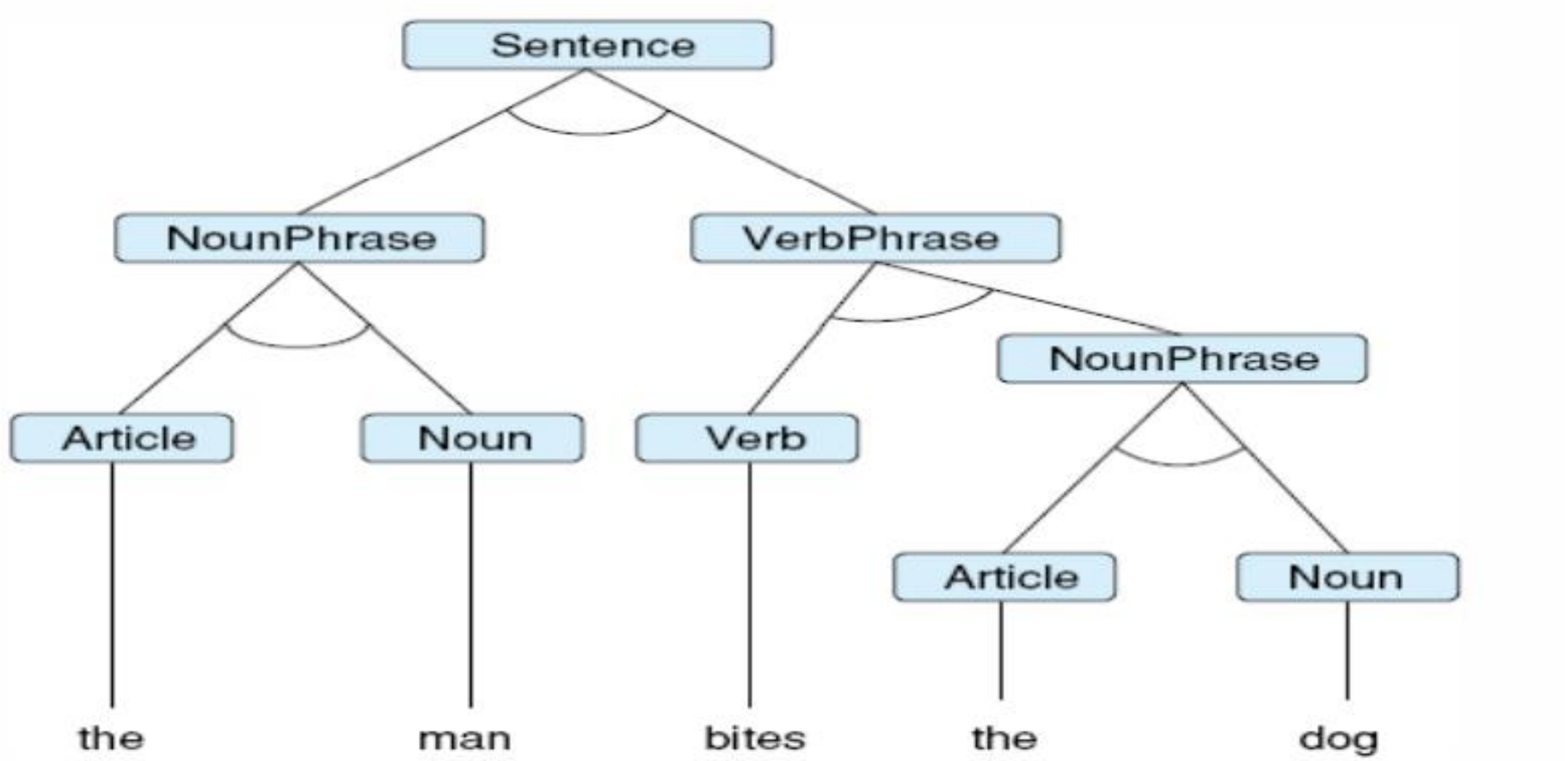
?- utterance([the, man, bites, the, dog]).

Yes

?- utterance([the, man, bites, the]).

no

A Context-Free Parser in Prolog



A Context-Free Parser in Prolog

The interpreter can also fill in possible legitimate words to incomplete sentences:

```
?- utterance([the, man, likes, X]).
X = man
;
X = dog
;
no
```

A Context-Free Parser in Prolog

Finally, the same code may be used to generate the set of all well-formed sentences using this limited dictionary and set of grammar rules:

?- utterance(X).

[man, likes]

;

[man, bites]

;

[man, likes, man]

;

etc.

A Context-Free Parser in Prolog

The grammar rules specify a subset of legitimate sentences of English. The Prolog grammar code represents these specifications. The interpreter is asked questions about them and the answer is a function of the specifications and the question asked. Since there are no constraints enforced across the subtrees that make up the full parse of a sentence, the parser/generator for this grammar is said to be **context free**.

Probabilistic Parsers in Prolog

In this section we extend the context-free grammar to include further syntactic and semantic constraints. For example, we may want some grammatical structures to be less likely than others, such as a noun by itself being less likely than an article followed by a noun.

Probabilistic Parsers in Prolog

Our first extension is to build a probabilistic context-free parser. To do this, we add a probabilistic parameter, **Prob**, to each grammar rule. Note that the probability that a sentence will be a noun phrase followed by a verb phrase is 1.0, while the probability that a noun phrase is simply a noun is less than the probability of it being an article followed by a noun. These probabilities are reflected in **pr** facts that are related to each grammar rule, r1, r2, …, r5.

Probabilistic Parsers in Prolog

The full probability of a particular sentence, **Prob**, however, is calculated by combining a number of probabilities: that of the rule itself together with the probabilities of each of its constituents. Thus, the full probability **Prob** of r1 is a product of the probabilities that a particular noun phrase is combined with a particular verb phrase.

```
utterance(Prob, X) :- sentence(Prob, X, [ ]).
sentence(Prob, Start, End) :-
    nounphrase(P1, Start, Rest),
    verbphrase(P2, Rest, End),
    pr(r1, P), Prob is P*P1*P2.
nounphrase(Prob, [Noun | End], End) :-
    noun(P1, Noun), pr(r2, P), Prob is P*P1.
nounphrase(Prob, [Article, Noun | End], End) :-
    article(P1, Article), noun(P2,Noun), pr(r3, P),
    Prob is P*P1*P2.
```

```
verbphrase(Prob, [Verb | End], End) :-
    verb(P1, Verb), pr(r4, P), Prob is P*P1.
verbphrase(Prob, [Verb | Rest], End) :-
    verb(P1, Verb),
    nounphrase(P2, Rest, End), pr(r5, P),
    Prob is P*P1*P2.
pr(r1, 1.0).
pr(r2, 0.3).
pr(r3, 0.7).
pr(r4, 0.2).
pr(r5, 0.8).
```

Rules	**Prob**
Sentence <-> NounPhrase VerbPhrase	1.0
NounPhrase <-> Noun	0.3
NounPhrase <-> Article Noun	0.7
VerbPhrase <-> Verb	0.2
VerbPhrase <-> Verb NounPhrase	0.8

Probabilistic Parsers in Prolog

```
article(0.25, a).

article(0.75, the).

noun(0.65, man).

noun(0.35, dog).

verb(0.9, likes).

verb(0.1, bites).
```

Probabilistic Parsers in Prolog

We now run several example sentences as well as offer general patterns of sentences：

?- utterance(Prob, [the, man, likes, the, dog]).

Prob = 0.0451474

Yes

?- utterance(Prob, [bites, dog])

No

Probabilistic Parsers in Prolog

?- utterance(Prob, [the, dog, bites, X]).

Prob = 0.0028665

X = man

;

Prob = 0.0015435

X = dog

;

No

Probabilistic Parsers in Prolog

We next enforce many of the same syntax/semantic relationships seen in this section by imposing constraints (**context sensitivity**) across the subtrees of the parse. **Context sensitivity** can be used to constrain subtrees to support relationships within a sentence such as article-noun and noun-verb number agreement.

A Context-Sensitive Parser in Prolog

A context-sensitive parser addresses the issues of the previous section in a different manner. Suppose we desire to have proper noun–verb agreement enforced by the grammar rules themselves. In the dictionary entry for each word its singular or plural form can be noted as such. Then in the grammar specifications for nounphrase and verbphrase a further parameter is used to signify the **Number** of each phrase.

A Context-Sensitive Parser in Prolog

This enforces the constraint that a singular noun has to be associated with a singular verb. Similar constraints for article–noun combinations can also be enforced. The technique we are using is constraining sentence components by enforcing variable bindings across the subtrees of the parse of the sentence.

```
utterance(X) :- sentence(X, [ ]).
sentence(Start, End) :-
    nounphrase(Start, Rest, Number),
    verbphrase(Rest, End, Number).
nounphrase([Noun | End], End, Number) :-
    noun(Noun, Number).
nounphrase([Article, Noun | End], End, Number) :-
    noun(Noun, Number), article(Article, Number).
```

```
verbphrase([Verb | End], End, Number) :-
    verb(Verb, Number).
verbphrase([Verb | Rest], End, Number) :-
    verb(Verb, Number), nounphrase(Rest, End, _).


article(a, singular).
article(these, plural).
article(the, singular).
article(the, plural).
```

```
noun(man, singular).

noun(men, plural).

noun(dog, singular).

noun(dogs, plural).

verb(likes, singular).

verb(like, plural).

verb(bites, singular).

verb(bite, plural).
```

A Context-Sensitive Parser in Prolog

We next test some sentences. The answer to the second query is no, because the subject (men) and the verb (likes) do not agree in number.

?- utterance([the, men, like, the, dog]).

Yes

?- utterance([the, men, likes, the, dog]).

no

A Context-Sensitive Parser in Prolog

If we enter the following goal, X returns all verb phrases that complete the plural “the men …” with all verb phrases with noun–verb number agreement. The final query returns all sentences with article–noun as well as noun–verb agreement.

```
?- utterance([the, men X]).
?- utterance(X).
```

NLP With Prolog in IBM Watson System

On February 14-16, 2011, the IBM Watson question answering system won the Jeopardy! Man vs. Machine Challenge by defeating two former grand champions, Ken Jennings and Brad Rutter. To compete successfully at Jeopardy!, Watson had to answer complex natural language questions over an extremely broad domain of knowledge. Moreover, it had to compute an accurate confidence in its answers and to complete its processing in a very short amount of time.

NLP With Prolog in IBM Watson System

The Question-Answering (QA) problem requires a machine to go beyond just match ing keywords in documents, which is what a web-search engine does, and correctly in terpret the question to figure out what is being asked. The QA system also needs to find the precise answer without requiring the aid of a human to read through the returned documents.

NLP With Prolog in IBM Watson System

Watson utilizes Natural Language Processing (NLP) technology to interpret the question and extract key elements such as the answer type and relationships between entities. Also, NLP was used to analyze (prior to the competition) the vast amounts of unstructured text (encyclopedias, dictionaries, news articles, etc.) that may provide evidence in support of the answers to the questions. Some of Watson's algorithms evaluate whether the relationships between entities in the question match those in the evidence.

NLP With Prolog in IBM Watson System

Watson's NLP begins by applying a parser [5] that converts each text sentence into a more structured form: a tree that shows both surface structure and deep, logical structure.

For example, in the following example Jeopardy! question:

POETS & POETRY: He was a bank clerk in the Yukon before he published "Songs of a Sourdough" in 1907

NLP With Prolog in IBM Watson System

The output of the parser includes, among many other things, that “published” is a verb with base form (or lemma) “publish”, subject “he”, and object “Songs of a Sourdough”.

NLP With Prolog in IBM Watson System

```
lemma(1, "he").
partOfSpeech(1,pronoun).
lemma(2, "publish").
partOfSpeech(2,verb).
lemma(3, "Songs of a Sourdough").
partOfSpeech(3,noun).
subject(2,1).
object(2,3).
```

```
authorOf(Author,Composition) :-
createVerb(Verb),
subject(Verb,Author),
author(Author),
object(Verb,Composition),
composition(Composition).

createVerb(Verb) :-
partOfSpeech(Verb,verb),
lemma(Verb,VerbLemma),
member(VerbLemma, ["write", "publish",...]).
```

NLP With Prolog in IBM Watson System

“The Prolog language is recognized to be an excellent solution for the problem of pattern matching and all problems that involve a depth-first search and backtracking. “

“Using Prolog for this task has significantly improved our productivity in developing new pattern matching rules and has delivered the execution efficiency necessary in order to be competitive in a Jeopardy! game.”