Programming Research Group

LEARNING TO READ BY THEORY REVISION

Stephen Muggleton
David Page
Ashwin Srinivasan

PRG-TR-26-96

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Learning to Read by Theory Revision

Stephen Muggleton
David Page
Ashwin Srinivasan

Oxford University Computing Laboratory
Wolfson Building
Parks Road
Oxford OX1 3QD
U.K.
Email: {steve,dpage,ashwin}@comlab.oxford.ac.uk

Abstract

This is the second paper reporting the progress of a pilot study using ILP to learn an English grammar directly from the sentences of a series of children's early reader books. In this paper we describe the addition of theory revision and predicate invention to Progol's learning methodology for this problem. These additions appear necessary for advancing beyond the first book in the series. With the addition of predicate invention, it appears necessary at times to invoke an oracle to choose one of several competing grammar rules that can be added in order to extend the grammar to parse a given sentence. Given the addition of theory revision, predicate invention, and access to an oracle, Progol is able to incrementally learn a human-comprehensible grammar from Books 1-3 of the series. The grammar of course parses all sentences in these books, and it is able to parse over two-thirds of the more difficult, previously unseen sentences of Book 4 in the series. These results provide encouragement and useful lessons for continuing the study through all 16 books of the series. Perhaps more importantly, they provide some general principles for efficient theory revision and predicate invention in ILP.

1 Introduction and Summary

Inductive Logic Programming has demonstrated strengths over other machine learning techniques on natural language data. (For a review see [1].) In a recent paper Stephen Muggleton describes the initial results of a pilot study aimed at acquiring an English natural language grammar directly from the text of a children's book [3]. The training sentences in that study were taken from Book 1 of the Ladybird "Read with me" series. In the experiment described in that paper, the legal sentence predicate was learned, given background knowledge of relevant phrase structures. Appendix A of this paper shows the background knowledge. The learned predicate definition was accurate and human-comprehensible. It was also non-trivial, involving multiple clauses, one of which was recursive. This paper describes the next step in the study, in which Book 2 and then Book 3 are used for training, and in which theory revision with predicate invention is employed.

In the initial experiment, the ILP system Progol [2] learned an accurate, human-comprehensible definition of the sentence predicate, resulting in an initial English grammar. Nevertheless, when applied in the same manner to sentences of Book 2, Progol is unable to improve the initial grammar. The reason for this failure is simple. The initial grammar correctly parses most of the sentences in Book 2, but when it fails to parse a sentence, the problem is not with the definition of the sentence predicate but with the definition of the noun phrase predicate. For example, the initial grammar cannot parse the sentence, "Here is Sam the dog", because it cannot recognize that "Sam the dog" is a noun phrase in which "the dog" modifies "Sam". From this observation we conclude that Progol must engage in theory revision, rather than single-predicate learning, if the pilot study is to advance beyond Book 2.

With the addition of a theory revision capability, described in Section 2, Progol successfully upgrades the initial grammar when training on sentences in Book 2. At least equally importantly, the upgrade is simple and comprehensible, involving the addition of a single clause to the definition of noun phrase. Nevertheless, theory revision alone is insufficient for Book 3 because the complexity of the sentences takes a large jump from Book 2 to Book 3, including the addition of dialog ("Tom have a look at this says Kate"), widespread use of infinitives and prepositional phrases ("Sam wants to jump into the tree"), use of "please" ("I want to go please says Tom"), and various idiosyncracies of normal English usage ("We want to go up up they say", "Up up up", and "And here are some for you Sam"). To meet the demands of this large jump in complexity, predicate invention is incorporated into the theory revision process.

Given the ability to perform theory revision with predicate invention, Progol learns a relatively accurate, comprehensible upgraded grammar from Book3. In the course of learning, five new predicates are invented. In learning general definitions for the invented predicates, Progol gives one of the invented predicates a recursive definition. A second invented predicate becomes identified with the first. Two others use the definition of the first in their own definitions. Furthermore, one of the new predicate definitions is in fact a definition of prepositional phrase, a phrase type that was not provided in the original background theory. Thus the new predicates are not merely collections of special cases but are conceptually rich.

The grammar that results from learning using Book 3 sentences of course parses all Book 3 sentences. When moving to the more difficult Book 4 for testing, the grammar parses just over two-thirds of the new sentences (52 of 77, or 67.5 %, of Book 4 sentences). Section 2 provides details about the experimental methodology, Section 3 presents details about the results, and Section 4 discusses general conclusions.

2 Experimental Methodology

Our first goal in the experimental methodology was to keep as true to the text of the books as possible. Re-ordering the sentences according to increasing length would have eased the learning task, but this was not done. Similarly, providing as examples all substrings of sentences that are themselves sentences would have helped the learning algorithm greatly, but this was not done. Another common technique to help with learning is to eliminate unusual or idiosyncratic sentences, of which these books contain many, but this was not done. The only changes made to the sentences were to eliminate commas and full stops, write all letters as lower case, expand contractions, and write each sentence as a list. For example, the sentence, "Here are some for you, Kate" is written as the list, *[here,are,some,for,you,kate]*. Appendix B provides the examples used from Books 1-4.

Our second goal in the experimental methodology was uniformity— to apply the same learning technique to each book, namely Progol with theory revision, predicate invention, and an oracle. Thus, although we knew theory revision was unnecessary for Book 1, and predicate invention was unnecessary for Book 2, the entire procedure was run on all books, to ensure that the results and computational demands did not degrade greatly when using the full procedure.

Our third goal was efficiency. To this end it was necessary after each book to convert the resulting grammar into Greibach Normal Form, to remove front recursion. Because Progol contains a depth-bounded theorem prover, removal of front recursion is theoretically unnecessary for termination. Nevertheless, front recursion remains a practical hindrance to efficiency.

The resulting procedure is as follows.

1. Begin with the current grammar being simply the phrase structure definitions in Appendix A, and the current book being Book 1.

2. Test to see how many sentences in the current book are accepted by the current grammar.

3. Using the first uncovered (unparsable) sentence in the current book as a seed example, and using the current grammar as the background theory, apply Progol with theory revision and predicate invention to learn possible new productions (clauses). Productions for sentence, noun phrase, and verb phrase may be learned, with or without the use of an invented predicate, yielding zero to six potential new productions.

4. Ask the user (the oracle) which one, if any, of the new productions to add to the grammar. If the user selects a production, add it to the grammar; otherwise, add the production yielding the seed sentence itself to the grammar as a special case.

5. If uncovered sentences remain, return to step 3.

6. The current grammar now covers (accepts) all sentences in the current book. Convert the grammar to Greibach Normal Form. This now becomes

the current grammar. If the current book is not Book 3, make the next book the current book, and go to step 2.

7. Test the final grammar on Book 4.

It remains to describe the technique used for theory revision with predicate invention.

The basic theory revision technique (without predicate invention) is as follows. Given a sentence that cannot be parsed, check to see whether it is "almost parsable", that is to say, whether it can be parsed with the exception of a single body literal of a single clause failing once. For example, consider again the sentence "Here is Sam the dog", and consider the clause

```
s(A,B) :- np(A,C), vp(C,D), np(D,B).
```

This clause says that any string consisting of a noun phrase, followed by a verb phrase, followed by a noun phrase is a sentence. The sentence "Here is Sam the dog" is almost parsable, given that "Here" is a noun phrase and "is" is a verb phrase (these facts are provable from the background theory in Appendix A). The failing literal instantiation is then taken to be a positive example for that literal's predicate. In the current example, the literal *np([sam,the,dog],[])*, which says "Sam the dog" is a noun phrase, is taken to be a positive example for the noun phrase predicate. In this fashion examples are generated for every "fixable" predicate. In effect this is an application of the identification operator of inverse resolution [4]. A new definition is learned for each fixable predicate, and one of the new definitions is selected. Selection currently is by an oracle, although replacement of the oracle by some form of quality measure, such as compression, is an interesting area for further research. To minimize computation time, we assume that only sentence, noun phrase, and verb phrase are fixable.

Of course, the preceding method of theory revision has some shortcomings. First, it assumes that for every clause that must be learned, some example is parsable with the exception of a single application of this missing clause. If more than one additional clause is necessary for a parse, or if the missing clause must be applied more than once, this method will not learn the missing clause. The second shortcoming is that this method works only for overly-specific theories; it cannot be used to specialize a theory. The third shortcoming is that all the necessary predicates must be present already in the background theory. If the missing clause needs to use a definition of prepositional phrase, for example, and that predicate is not in the background theory, the missing clause cannot be constructed. For the purposes of the present experiment, we simply accept the first two shortcomings, but we address the third shortcoming by allowing predicate invention as described below.

The technique for predicate invention can be described as follows. During clause construction, a single new (invented) predicate can be used in a clause body. For each sentence, the invented predicate can match any substring of the sentence as needed to allow the clause body to succeed. For example, given the clause

```
s(A,B) :- np(A,C), vp(C,D), np(D,E), newpred(E,B).
```

and the background knowledge in Appendix A, the clause body succeeds on the sentence "Tom threw the ball to Kate" if the new predicate is true of the prepositional phrase "to Kate". The literal assumed to be true—in this case, *newpred([to,kate],[])*—is taken to be a positive example for the new predicate. Once a clause involving a new predicate is learned, the positive examples for the new predicate are then used to learn a compressive definition of the new predicate.

This method of predicate invention also has several shortcomings. First, it assumes that only one new predicate is needed for the clause currently being learned, and that this new predicate is needed only once in the clause. Nevertheless, to keep computation time down we accept this shortcoming. Second, the new predicate's definition is generated only after the clause that uses it is learned. To see the potential difficulty here, imagine that two clauses C_1 and C_2 for the sentence predicate are built during a search and that both require an invented predicate. Assume further that C_1 has slightly better compression than C_2. This compression measure does not take into account the sizes of the definitions of the new predicates used by C_1 and C_2. It may be that choosing C_2 will lead to an invented predicate with a highly compressive definition, while choosing C_1 will lead to an invented predicate whose definition is just a non-compressive extensional definition. While this is a potential problem to keep in mind, again we accept this shortcoming for the present experiment, in exchange for efficient computation.

3 Results

On Book 1 data, Progol learns the following clauses representing sentence productions, which it adds to the background theory in Appendix A, to yield an initial grammar that we call Grammar 1. In these productions *np* is a noun phrase, *s* is a sentence, *conj* is a conjunction, *inter* is an interjection, and *snp* is a simple noun phrase (proper name, noun, or pronoun).

```
s(A,B) :- np(A,C), verb(C,D), np(D,B).
s(A,B) :- s(A,C), conj(C,D), s(D,B).
s(A,B) :- inter(A,C), inter(C,B).
s(A,B) :- inter(A,C), snp(C,D), inter(D,B).
s([tom,is,in,here,and,sam,is,in,here],[]).
```

The first two productions are quite natural and sensible. The third arises from "sentences" of the form "No, no", while the fourth comes from "sentences" of the form "No, Sam, no". Grammar 1 has an accuracy of 94.7% on the previously unseen data of Book 2.

Training on Book 2 results in the addition of the production

```
np(A,B) :- snp(A,C), detnp(C,B).
```

to Grammar 1 to yield Grammar 2. Here *detnp* denotes determinate noun phrase. Addition of this production allows the sentences "Here is Sam the dog" and "Sam the dog is here" to be parsed. Nevertheless, Grammar 2 has an accuracy mark of only 11.4 % on Book 3, because of the large jump in sentence complexity that Book 3 takes.

Training on Book 3 with Grammar 2 as the background theory results in more sweeping changes to the theory than on previous books, because of the complexity of Book 3. The definitions of sentence, noun phrase, and verb phrase are changed, and the definitions of five new predicates are added. The new predicates are named $p1$, $p2$, $p3$, $p4$, and $p5$. The resulting definitions of the above predicates are as follows. Notice that the definition of the invented predicate $p2$ is a reasonable definition of prepositional phrase.

```
p1([i,want,this,hat,please,says,tom],[says,tom]).
p1([i,want,to,go,please,says,tom],[says,tom]).
p1([tom,have,a,look,at,this,says,kate],[says,kate]).
p1([yes,here,we,go,says,tom],[says,tom]).
p1([can,we,go,into,it,says,kate],[says,kate]).
p1([no,sam,no,not,in,the,water],[in,the,water]).
p1([no,sam,you,cannot,jump,into,the,tree],[into,the,tree]).
p1([yes,it,is,fun,in,the,boat],[in,the,boat]).
p1(A,B) :- vp(A,C), np(C,B).
p1(A,B) :- inter(A,B).
p1(A,B) :- pron(A,C), vp(C,B).
p1(A,B) :- verb(A,B).
p1(A,B) :- np(A,C), p1(C,B).

p2(A,B) :- prep(A,C), p1(C,B).
p2(A,B) :- prep(A,C), np(C,B).

p3(A,B) :- p1(A,B).

p4([some,for,you,kate],[kate]).
p4(A,B) :- adj(A,B).
p4(A,B) :- adj(A,C), prep(C,B).

p5([and,sam,wants,to,go],[to,go]).
p5([yes,says,kate,i,want,to,go],[to,go]).
p5(A,B) :- p1(A,B).

vp(A,B) :- avp(A,B).
vp(A,B) :- averb(A,C), p4(C,B).
vp(A,B) :- p1(A,C), vp(C,B).
vp(A,B) :- vp(A,C), prep(C,B).

np(A,B) :- snp(A,B).
np(A,B) :- detnp(A,B).
```

```
np(A,B) :- prep(A,C), iverb(C,D), p2(D,B).
np(A,B) :- np(A,C), np(C,B).
np(A,B) :- np(A,C), conj(C,D), np(D,B).

s(A,B) :- p2(A,C), prep(C,B).
s(A,B) :- p3(A,C), s(C,B).
s(A,B) :- np(A,C), p3(C,B).
s(A,B) :- np(A,C), s(C,B).
s(A,B) :- inter(A,C), inter(C,B).
s(A,B) :- adv(A,C), s(C,B).
s(A,B) :- conj(A,C), s(C,B).
s(A,B) :- p1(A,C), tverb(C,D), snp(D,B).
s(A,B) :- p1(A,C), conj(C,D), vp(D,B).
s(A,B) :- p1(A,C), np(C,D), vp(D,B).
s(A,B) :- p3(A,C), prep(C,D), detnp(D,B).
s(A,B) :- p5(A,C), prep(C,D), vp(D,B).
s(A,B) :- vp(A,C), p2(C,D), np(D,B).
s(A,B) :- adjp(A,C), np(C,D), verb(D,B).
s(A,B) :- np(A,C), vp(C,D), np(D,B).
s(A,B) :- inter(A,C), snp(C,D), inter(D,B).
s(A,B) :- s(A,C), conj(C,D), s(D,B).
s(A,B) :- s(A,C), propname(C,B).
```

The resulting grammar, which we call Grammar 3, parses just over two-thirds (52 of 77, or 67.5%) of the sentences of Book 4 in the series.

It is somewhat difficult to report the computation time, because of the human component in the work. First, the user participates as the oracle when more than one candidate production can be added to parse a given sentence. Second, we have not yet implemented the algorithm for converting to Greibach Normal Form, so this was done by hand. Even so, the entire experiment can be done in under three hours, with most of this time being used by the human rather than the computer. The computer involved was a Sun SPARC 20 workstation.

4 Conclusions

The specific conclusion of the experiment reported in this paper is that the pilot study should be continued through Book 16. Whether the study should be continued was in some doubt until it was seen that theory revision and predicate invention could be used to efficiently upgrade the initial grammar to handle Books 2 and 3.

The more general conclusion is that for specific domains, theory revision with predicate invention can be constrained and efficient enough to be practical. In this natural language domain, the constraints on predicate invention are as follows.

- Invented predicates are binary.

- Given an example of the form *s(L,[])*, where L is a list, the only legal examples for an invented predicate have as the first argument a suffix of L and as the second argument a suffix of the first argument. Thus for an original example of size n, the number of possible examples for an invented predicate is at most n^2.
- It is assumed that only one invented predicate appears in the body of any clause.

The results of these restrictions are that (1) for any given example, the number of possible examples of an invented predicate is small, and (2) for any candidate clause being considered, only one example of an invented predicate is generated. In this way, the restrictions on predicate invention provide efficient computation. For other domains some of the restrictions probably will be different, but if they achieve results (1) and (2) above, then predicate invention should remain practical.

Acknowledgements

The authors wish to acknowledge the support of two EPSRC grants, an EPSRC Advanced Research Fellowship, and the European Community's long-term research project ILP2.

Appendix A

The theory below is a definite clause grammar used as the initial background theory for the experiment described in this paper. For the reader unfamiliar with definite clause grammars, a clause of the form

```
np(A,B) :- det(A,C), noun(C,B).
```

represents a production of the form

```
NP -> Det Noun
```

In this representation, each predicate denotes a nonterminal or a terminal. All predicates are binary. The first argument denotes an input string, while the second denotes a suffix of the input string designated as an output string. A literal of the form $np(A, B)$ asserts that the string A begins with a noun phrase, and when the noun phrase is removed the string B remains. Thus we might assert that $np([the, dog, ran], [ran])$.

The initial background theory for the experiment is the following.

```
np(A,B) :- snp(A,B).
np(A,B) :- detnp(A,B).
np(A,B1) :- snp(A,B), np1(B,B1).
np(A,B1) :- detnp(A,B), np1(B,B1).
```

```
np1(S1,S2) :- conj(S1,S4), np(S4,S2).
np1(S1,S3) :- conj(S1,S4), np(S4,S2), np1(S2,S3).


detnp(S1,S2) :- det(S1,S3), adjp(S3,S4), noun(S4,S2).
detnp(S1,S2) :- det(S1,S3), noun(S3,S2).

snp(S1,S2) :- noun(S1,S2).
snp(S1,S2) :- pron(S1,S2).
snp(S1,S2) :- propname(S1,S2).

det([a|S],S).
det([the|S],S).

vp(S1,S2) :- avp(S1,S2).
vp(S1,S2) :- averb(S1,S3), avp(S3,S2).
vp(S1,S2) :- avp(S1,S3), prep(S3,S2).

avp(S1,S2) :- tverb(S1,S2).
avp(S1,S2) :- tverb(S1,S3), adv(S3,S2).
avp(S1,S2) :- adv(S1,S3), tverb(S3,S2).

adjp(S1,S2) :- adj(S1,S2).
adjp(S1,S2) :- adv(S1,S3), adj(S3,S2).
adjp(S1,S2) :- adj(S1,S3), adj(S3,S2).

propname([tom|S],S).
propname([kate|S],S).
propname([sam|S],S).

noun([dog|S],S).
noun([shop|S],S).
noun([toy|S],S).
noun([ball|S],S).
noun([tree|S],S).
noun([trees|S],S).
noun([here|S],S).
noun([dragon|S],S).
noun([boat|S],S).
noun([hat|S],S).
noun([fun|S],S).
noun([water|S],S).
noun([fish|S],S).
noun([down|S],S).
noun([up|S],S).
```

```
inter([no|S],S).
inter([yes|S],S).
inter([please|S],S).

adj([some|S],S).
adj(S1,S2) :- noun(S1,S2).

pron([i|S],S).
pron([it|S],S).
pron([this|S],S).
pron([we|S],S).
pron([they|S],S).
pron([you|S],S).
pron([he|S],S).

tverb([like|S],S).
tverb([likes|S],S).
tverb([has|S],S).
tverb([have|S],S).
tverb([is|S],S).
tverb([are|S],S).
tverb([says|S],S).
tverb([wants|S],S).

iverb([come|S],S).
iverb([go|S],S).
iverb([look|S],S).
iverb([jump|S],S).

averb([have|S],S).
averb([are|S],S).
averb([is|S],S).
averb([can|S],S).
averb([cannot|S],S).

prep([in|S],S).
prep([to|S],S).
prep([up|S],S).
prep([down|S],S).
prep([at|S],S).
prep([for|S],S).
prep([into|S],S).

conj([and|S],S).

adv([no|S],S).
adv([not|S],S).
```

Appendix B

The following are the examples for the experiment described in this paper.

Book 1 Data

```
s([here,is,kate],[]).
s([here,is,tom],[]).
s([here,is,sam],[]).
s([sam,is,a,dog],[]).
s([here,is,kate,and,here,is,sam],[]).
s([no,no],[]).
s([tom,is,in,here,and,sam,is,in,here],[]).
s([kate,likes,the,dog,and,tom,likes,the,dog],[]).
s([sam,likes,tom,and,sam,likes,kate],[]).
s([no,sam,no],[]).
s([i,like,tom],[]).
s([i,like,kate],[]).
s([i,like,the,dog],[]).
s([here,is,tom,and,here,is,kate],[]).
s([no,sam,no],[]).
s([here,is,a,shop],[]).
s([the,dog,is,in,the,shop],[]).
s([tom,is,in,the,toy,shop],[]).
s([kate,is,in,the,toy,shop],[]).
s([sam,is,in,the,toy,shop],[]).
s([sam,has,a,toy],[]).
s([sam,likes,the,toy],[]).
s([no,sam,no],[]).
s([kate,and,tom,like,the,ball],[]).
s([sam,likes,the,ball],[]).
s([the,dog,has,the,ball],[]).
s([the,dog,likes,the,ball],[]).
s([here,is,a,tree],[]).
s([the,ball,is,in,the,tree,and,tom,is,in,the,tree],[]).
s([tom,has,the,ball],[]).
s([tom,and,kate,like,the,ball],[]).
s([sam,has,the,ball],[]).
s([no,sam,no],[]).
s([sam,has,no,toy],[]).
s([sam,has,the,toy],[]).
s([sam,likes,the,toy],[]).
s([i,like,kate,and,i,like,tom,and,i,like,sam],[]).
```

Book 2 Data

```
s([here,is,a,shop],[]).
s([tom,is,here],[]).
s([kate,is,here],[]).
s([tom,is,in,the,shop],[]).
s([kate,is,in,the,shop],[]).
s([kate,likes,the,trees,and,the,dragon],[]).
s([tom,likes,the,trees,and,the,dragon],[]).
s([here,is,sam,the,dog],[]).
s([sam,is,in,the,shop],[]).
s([sam,the,dog,is,here],[]).
s([sam,has,a,toy],[]).
s([no,sam,no],[]).
s([kate,has,the,toy],[]).
s([here,is,the,toy],[]).
s([sam,has,a,ball],[]).
s([sam,likes,the,ball],[]).
s([i,like,the,ball],[]).
s([kate,has,a,ball,and,tom,has,a,ball],[]).
s([tom,and,kate,like,the,trees,and,the,dragon],[]).
s([tom,and,kate,like,the,dog],[]).
s([tom,is,in,here,and,kate,is,in,here],[]).
s([sam,is,in,here],[]).
s([sam,has,the,toy],[]).
s([sam,likes,the,toy],[]).
s([no,sam,no],[]).
s([tom,has,a,toy,tree],[]).
s([i,like,the,toy,trees],[]).
s([i,like,tom,and,kate],[]).
s([i,like,the,dog],[]).
s([no,sam,no],[]).
s([here,is,tom,and,here,is,sam],[]).
s([here,is,kate],[]).
s([kate,is,here,and,tom,is,here],[]).
s([here,is,the,dog],[]).
s([here,is,sam],[]).
s([here,is,tom],[]).
s([here,is,kate,and,here,is,sam],[]).
s([i,like,kate,and,tom,and,sam],[]).
```

Book 3 Data

```
s([come,here,kate,please],[]).
```

```
s([come,and,look],[]).
s([kate,and,sam,come,and,look],[]).
s([it,is,like,a,boat,says,kate],[]).
s([it,is,a,boat,says,tom],[]).
s([i,have,a,boat,says,tom],[]).
s([have,this,tom,says,kate],[]).
s([yes,says,tom],[]).
s([tom,wants,a,hat],[]).
s([i,want,this,hat,please,says,tom],[]).
s([i,like,this,hat,says,kate],[]).
s([come,in,the,boat,sam,says,kate],[]).
s([it,is,fun],[]).
s([yes,here,we,go,says,tom],[]).
s([we,want,to,go,up,up,they,say],[]).
s([they,like,the,boat],[]).
s([it,is,fun],[]).
s([yes,it,is,fun,in,the,boat],[]).
s([tom,have,a,look,at,this,says,kate],[]).
s([look,at,the,water],[]).
s([they,go,to,look,at,the,water],[]).
s([look,at,the,fish,kate],[]).
s([look,at,the,fish,in,the,water],[]).
s([we,can,go,to,look,at,the,fish],[]).
s([no,sam,no,not,in,the,water],[]).
s([you,cannot,go,in,the,water],[]).
s([you,cannot,have,the,fish],[]).
s([here,they,are,in,the,boat],[]).
s([look,at,this,kate],[]).
s([we,can,go,down],[]).
s([down,down,they,go],[]).
s([here,are,some,for,you,kate],[]).
s([you,have,some,tom],[]).
s([here,are,some,for,you],[]).
s([and,here,are,some,for,you,sam],[]).
s([they,go,up,in,the,boat],[]).
s([sam,wants,to,jump,into,a,tree],[]).
s([no,sam,you,cannot,jump,into,the,tree],[]).
s([sam,wants,to,jump,into,the,tree],[]).
s([he,can,have,fun,in,it],[]).
s([sam,wants,to,jump],[]).
s([no,sam,you,cannot,jump,into,it],[]).
s([look,says,tom],[]).
s([i,want,to,go,into,it,he,says],[]).
s([can,we,go,into,it,says,kate],[]).
s([here,we,go,sam],[]).
s([up,up,up],[]).
s([here,we,are],[]).
```

```
s([we,can,go,and,look],[]).
s([come,here,sam,says,kate],[]).
s([we,can,have,fun],[]).
s([you,cannot,jump,she,says],[]).
s([tom,looks],[]).
s([look,at,this,kate,says,tom],[]).
s([no,no],[]).
s([he,cannot,come,into,the,boat],[]).
s([down,kate,down,tom,says],[]).
s([look,at,sam,says,kate],[]).
s([sam,no,no],[]).
s([down,sam,down],[]).
s([you,cannot,jump,sam],[]).
s([come,here,please],[]).
s([i,want,to,go,please,says,tom],[]).
s([and,sam,wants,to,go],[]).
s([yes,says,kate,i,want,to,go],[]).
s([down,they,go],[]).
s([here,they,are],[]).
s([it,is,fun,in,the,boat,says,kate],[]).
s([yes,says,tom],[]).
s([we,like,this,boat],[]).
```

Book 4 Data

```
s([tom,and,kate,are,here],[]).
s([kate,has,a,toy],[]).
s([tom,has,the,ball],[]).
s([sam,is,not,here],[]).
s([sam,looks,for,the,toy,he,likes],[]).
s([the,toy,is,not,here],[]).
s([he,looks,for,the,toy],[]).
s([here,they,are],[]).
s([here,they,are,in,a,tree],[]).
s([tom,i,can,jump],[]).
s([can,you,jump,like,i,can],[]).
s([no,kate,no],[]).
s([look,here,tom,look,at,this],[]).
s([in,here,tom,look,at,this],[]).
s([he,comes,and,looks],[]).
s([they,look,at,the,fish,in,the,water],[]).
s([sam,looks,at,the,fish,in,the,water],[]).
s([no,sam,no,you,cannot,have,it],[]).
s([it,is,not,for,you],[]).
s([the,fish,can,jump],[]).
```

```
s([it,jumps,up],[]).
s([look,at,it,jump,says,kate,to,tom],[]).
s([they,like,to,look,at,the,fish],[]).
s([like,this,tom,says,to,kate],[]).
s([it,is,fun],[]).
s([yes,like,this,kate,says,to,tom],[]).
s([we,like,it],[]).
s([they,are,in,the,shop],[]).
s([i,want,a,ball,please,says,tom],[]).
s([yes,says,kate],[]).
s([and,i,want,a,boat,please],[]).
s([we,can,have,fun,they,say],[]).
s([here,you,are,says,tom],[]).
s([we,want,to,have,fun],[]).
s([look,down,comes,the,ball],[]).
s([yes,here,it,comes,down,down,he,says],[]).
s([come,here,tom,and,kate],[]).
s([some,for,you,kate,and,some,for,you,tom],[]).
s([you,cannot,have,this,sam],[]).
s([here,is,some,water,for,you],[]).
s([sam,looks,for,the,toy],[]).
s([he,wants,it],[]).
s([he,likes,the,toy],[]).
s([no,sam,you,cannot,have,it],[]).
s([it,is,not,a,toy],[]).
s([here,is,the,boat,tom,and,kate],[]).
s([have,a,go,in,it],[]).
s([sam,cannot,go,in,the,boat],[]).
s([sam,likes,to,go,into,the,water],[]).
s([he,jumps,in],[]).
s([tom,says,i,want,you,to,come,in,kate],[]).
s([no,no,tom],[]).
s([here,is,toms,hat],[]).
s([look,says,kate,a,fish,is,in,the,hat],[]).
s([the,fish,in,the,hat,jumps,up],[]).
s([it,jumps,into,the,water],[]).
s([have,some,kate,and,tom],[]).
s([the,dog,jumps,up],[]).
s([down,sam,down],[]).
s([yes,sam,you,can,have,some],[]).
s([he,can,have,some,says,kate],[]).
s([look,tom,look],[]).
s([look,at,this,says,kate],[]).
s([go,in,the,water,please,tom,says,to,the,dog],[]).
s([the,dog,jumps,into,the,water],[]).
s([go,for,it,sam,they,say],[]).
s([here,comes,sam],[]).
```

```
s([he,has,the,boat],[]).
s([look,at,this,says,tom],[]).
s([the,water,has,come,in,he,says],[]).
s([and,look,at,this,says,kate],[]).
s([sam,you,look,at,this],[]).
s([we,have,to,go,tom,says],[]).
s([yes,we,have,to,go,kate,says],[]).
s([no,no,sam,you,cannot,have,it],[]).
s([here,you,are],[]).
s([here,we,go],[]).
```

References

[1] R.J. Mooney. Inductive Logic Programming for natural language processing. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 205–224, Report No. 96-019, 1996. Stockholm University/Royal Institute of Technology.

[2] S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

[3] S. Muggleton. Experimental acquisition of grammar from early reader books. PRG-TR 18-96, Oxford University Computing Laboratory, Oxford, 1996.

[4] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352. Kaufmann, 1988.