To the International Computing Community: a new East-West Challenge

Donald Michie, Stephen Muggleton
David Page and Ashwin Srinivasan
Oxford University Computing Laboratory, UK.

Figure 1, following the appendix, originated 20 years ago from Ryszard Michalski. As in scientific discovery, it is required to conjecture some plausible Law, in this case governing what kinds of trains are Eastbound and what kinds Westbound. A few years later, Donald Michie published the trains problem in the British popular computer press. The post-bag contained some neat conjectures from readers, such as:

Theory A: If a train has a short closed car, then it is Eastbound and otherwise Westbound.

Theory B: If a train has two cars, or has a car with a jagged roof, then it is Westbound and otherwise Eastbound.

One and the same set of observations can of course support different theories. Pending new observations we generally take the simplest and hope for the best. Theory A is marginally simpler than B and a good deal simpler than C from one reader:

Theory C: If a train has more than two different kinds of load, then it is Eastbound and otherwise Westbound.

No learning system of those days was capable of coming up with a theory like C, still less one like the following from another reader:

Theory D: For each train add up the total number of sides of loads (taking a circle to have one side). If the answer is a divisor of 60 then the train is Westbound and otherwise Eastbound.

Time has moved on and thanks to Stephen Muggleton's Prolog train generator we have observed ten more trains (see Figure 2). We here present three competitions inspired by these. Closing date for all three competitions is eight weeks after the appearance on $4^{th}August$ in the British weekly *Computing* of an article from Donald Michie that poses Competition 1 to their readership. Specifically, the closing date is September 30, 1994.

Competition 1

Merging the new trains with Michalski's original ten we applied a freshly conjectured Law to yield class labels for the resultant set of twenty. Theories A, B, C and D all now fail on the enlarged sample. Can inductive inference recover the new Law, or one as good or better, fitting all 20?

By kind donation of Oxford University Press, the best entry, judged on accuracy and simplicity, wins a free copy of Richard Gregory's handsome book *The Oxford Companion to the Mind* (price £35, US$49.95, in the bookstores). Laws may be of human or of machine authorship. To be in line for the prize entrants should email their entries, along with their names and postal addresses to:

Machine.Intelligence@comlab.ox.ac.uk

Each email entry for Competition 1 should have the subject heading of the form *Trains1 (Entrant's Code Name)*. For example, an acceptable subject heading for an email entry to Competition 1 by M Bain is *Trains1 (bain)*.

The additional ten trains were selected from a randomly generated pool and assigned class labels, all in a way that ensured that the resulting set of 20 was split into East and West subsets by a new Law – call it Theory X. The train generator itself applies attribute constraints suggested by Michalski's original ten as follows:

1. A train has two, three or four cars, each of which can either be long or short.

2. A long car can have either two or three axles.

3. A short car can be rectangular, u-shaped, bucket-shaped, hexagonal, or elliptical, while a long car must be rectangular.

4. A hexagonal or elliptical car is necessarily closed, while any other car can be either open or closed.

5. The roof of a long closed car can be either flat or jagged.

6. The roof of a hexagonal car is necessarily flat, while the roof of an elliptical car is necessarily an arc. Any other short closed car can have either a flat or a peaked roof.
7. If a short car is rectangular then it can also be double-sided.
8. A long car can be empty or it can contain one, two or three replicas of one of the following kinds of load: circle, inverted-triangle, hexagon, rectangle.
9. A short car contains either one or two replicas of the following kinds of load: circle, triangle, rectangle, diamond.
10. No sub-distinctions are drawn among rectangular loads, even though some are drawn square and others more or less oblong. The presumption is that they are drawn just as oblong as they need to be in each case to fill the available container space.
11. In Michalski's original version a possible distinction between hollow and solid wheels was ignored, as also here.

Muggleton's Prolog train-generator embodies the above constraints together with certain distributional assumptions concerning values of descriptors, so as to preserve statistical coherence with Michalski's original ten. The Prolog text of the generator supplied below will resolve any obscurities in the above account. Figure 3 provides a randomly generated set of 100 unclassified trains that may also be useful for this purpose (see Table 2 for the Prolog representation of these trains). The generator assigns no class labels. We mentally performed such assignments according to Theory X and found that 56 of the 100 trains are X-classified as Eastbound. The simplest Law Y to be received that correctly classifies the twenty trains of Figures 1 and 2 (see Table 1 for their Prolog representation) wins Competition 1. Y may or may not be equivalent to X. When all entries have been received from readers of *Computing* and from readers of this communication, Law Y will be divulged, along with results and analysis.

Measurement of complexity for Competition 1

For assessing the complexity of candidate theories (to be a candidate the theory must correctly classify the given training set) we propose as follows:

1. If the candidate classifier is not expressed as a Prolog program, we will encode it in Prolog as tersely as we can, using as reference language Clocksin and Mellish's *Programming in Prolog* (3^{rd} edition, 1987), Springer-Verlag, including their library predicates, together with those provided in the "concept tester" in Muggleton's Prolog code below. If the entry is already in Prolog, we will help it to score well by any obviously advantageous re-phrasings that occur to us, within the bounds of the specified rule language.

2. We will run the candidate theory on the set of 20 to check that it correctly reproduces the classification. For this we will apply Muggleton's concept tester, with the Prolog-encoding of the supplied rule set, to the Prolog fact file corresponding to the trains of Figure 1 and 2. The contents of this file are reproduced below in Table 1, following the text of the train-generator and concept tester.

3. We will then compute a complexity score as the number of clause occurrences plus the number of term occurrences plus the number of atom occurrences. Remember that the score includes the complexity cost of any "background knowledge" in the form of additional procedures that are found necessary to add in order to render the submitted classifier runnable on the target data.

The definitions of *clause occurrence*, *atom occurrence*, and *term occurrence* that are clear for first-order definite clause logic are ambiguous for Prolog in general. To redress this, a substring in a Prolog program is identified as a *clause*, *atom*, or *term* if it is labelled as such during a parse of the program according to an unambiguous context-free grammar for Prolog. Section A.1 of Appendix A describes such a grammar. For complete clarity, this appendix also provides (as Section A.2) Ashwin Srinivasan's Prolog program for measuring complexity scores.

As a preliminary exercise, we applied the above three steps to an initial encoding of Theory X and obtained a complexity score of 19. We first verified that the machine execution of our Prolog version of X assigned precisely the same class labels to the same trains as did the classification by hand, not only over the set of 20, but also on the set of 100. For these machine tests, we used two files of 20 and 100 Prolog facts respectively represented in Tables 1 and 2. These files, `20trains.pl` and `100trains.pl`, and the program for computing the complexity score `complex.pl` are available as ASCII text in the compressed tar file obtainable at

```
URL = ftp://ftp.comlab.ox.ac.uk/pub/Packages/ILP/trains.tar.Z

FTP site = ftp.comlab.ox.ac.uk

FTP file = pub/Packages/ILP/trains.tar.Z
```

Competition 2

The rules of engagement in Competition 1 fail to allow for subsymbolic and semi-symbolic forms of inductive analysis, ranging from multi-variate non-linear statistical approaches through neural nets and genetic algorithms to paranormal and other intuitive human mental skills. A separate competition is accordingly available for entries in the form of an allocation of Eastbound/Westbound labels to the hundred trains of Figure 3 (and Table 2), unaccompanied by any explicit classifying rule or formula. Since Theory X is the simplest known to us we provisionally take it as the oracle for adjucating subsymbolically derived classifications of the test set of 100. But what if a subsymbolic learner uncovers a classification that is closest to an entry Z either taken from Competition 1 or otherwise known to us? Provided that the complexity score of Z lies within the bottom quartile of the scores of all such theories, then Z's class labellings will be used for assessment of that subsymbolic learner. The idea is that discovery of a sufficiently strong regularity in the collection of trains of Figures 1 and 2 should be an optional touchstone in Competition 2, rather than some particular regularity that appeals to the judges.

For this sub-symbolic section the solution that correctly classifies, on the above assessment principle, the highest number of new trains wins a free copy of the "Statlog Book" by kind donation of Paramount Publishing International. This is the 265-page *Machine Learning, Neural and Statistical Classification* (eds. D.Michie, D.J.Spiegelhalter and C.C.Taylor, Ellis Horwood Series in Artificial Intelligence), 1994 (price £39.95, US$67.95, in the bookstores). As before, we do not care whether the induction agent is human, machine, or human-machine. To be in line for the prize, entrants should email their entries, along with their names and postal addresses to:

Machine.Intelligence@comlab.ox.ac.uk

Each email entry for Competition 2 should have the subject heading of the form *Trains2 (Entrant's Code Name)*. For example, an acceptable subject heading for an email entry to Competition 2 by M Bain is *Trains2 (bain)*.

Competition 3

Returning to theory discovery, we propose a further challenge, this time based on induction from trains generated and pre-classified entirely randomly (strictly speaking pseudo-randomly). For this final exercise, the cross-ruling imposed on the first page of Figure 3 partitions 50 trains into five sets of ten. Arbitrarily assigning "Eastbound" to the trains in the left column and "Westbound" to those in the right column, we set up five separate induction tasks analogous to Michalski's original.

Viewing inductive inference as a way of compressing facts into theories, the random origin of these five sets might seem to defy discovery of anything beyond marginal regularities. This is of course true of large samples. It is now open to the computing community, this time including ourselves, to investigate on a competitive basis whether samples as small as these turn out to possess interesting degrees of describability in terms of the descriptive language and vocabulary supplied. Even the simplest and most compelling theory got from one of these five small samples has, of course, zero predictivity – for example, as could be tested against the second 50 trains in Figure 3. Within its own small world such a theory explains everything, but predicts nothing. A neat example of the opposite kind, that predicts everything but explains nothing, is presented by Ian Stewart in the July 1994 *Scientific American* ("The Ultimate in Anty-Particles",—yes the spelling is correct!).

Only entries that submit theories for each of the 5 constituent sub-tasks will be considered. First place will go to the entry with the lowest grand total complexity summed over the 5 sub-tasks. No prizes are offered, but the best entries will be published as part of a follow-up communication in which the challenge as a whole and its outcomes will be discussed.

Entrants to Competition 3 should email their entries, along with their names and postal addresses to:

Machine.Intelligence@comlab.ox.ac.uk

Each email entry for Competition 3 should have the subject heading of the form *Trains3 (Entrant's Code Name)*. For example, an acceptable subject heading for an email entry by M Bain is *Trains3 (bain)*.

Prolog generator and concept-tester plus train files follow.

```
% Stephen Muggleton´s Prolog code for
% randomly generating Michalski trains.
% To run this you need a Prolog interpreter which executes
% a goal of the form:
%        R is random
% Otherwise replace the definition of random/2 appropriately.
% (``R is random´´ binds R to a pseudo-random number--floating
% point--between 0 and 1.)
% The top-level predicates are trains/0 and trains/1.

trains :-
        repeat, train1(X), show(X),
        write(´More (y/n)? ´),
        read(n).

trains([]) :- !.
trains([H|T]) :- train1(H), trains(T), !.

train1(Carriages) :-
        random([0,0.3,0.3,0.4],NCarriages),
        len1(Carriages,NCarriages),
        carriages(Carriages,1), !.

carriages([],_).
carriages([C|Cs],N) :-
        carriage(C,N), N1 is N+1,
        carriages(Cs,N1), !.

carriage(c(N,Shape,Length,Double,Roof,Wheels,Load),N) :-
        randprop(car_length,[0.7,0.3],Length),
        shape(Length,Shape),
        double(Length,Shape,Double),
        roof1(Length,Shape,Roof),
        wheels(Length,Wheels),
        load(Length,Load), !.

shape(long,rectangle).
shape(short,S) :-
        randprop(car_shape,[0.048,0.048,0.524,0.190,0.190],S).

double(short,rectangle,Double) :-
        randprop(car_double,[0.73,0.27],Double), !.
```

```
double(_,_,not_double) :- !.

roof1(short,ellipse,arc) :- !.
roof1(short,hexagon,flat) :- !.
roof1(short,_,R) :- randprop(roof_shape,[0.842,0.105,0,0.053,0],R).
roof1(long,_,R) :- randprop(roof_shape,[0.333,0.444,0.223,0,0],R).

wheels(short,2).
wheels(long,W) :- random([0,0.56,0.44],W).

load(short,l(Shape,N)) :-
        randprop(load_shape,[0.381,0.048,0,0.190,0.381,0],Shape),
        random([0.952,0.048],N).
load(long,l(Shape,N)) :-
        randprop(load_shape,[0.125,0,0.125,0.625,0,0.125],Shape),
        random([0.11,0.55,0.11,0.23],N1), N is N1-1.

random(Dist,N) :-
        R is random,
        random(1,0,R,Dist,N).

random(N,_,_,[_],N).
random(N,P0,R,[P|_],N) :-
        P1 is P+P0, R=<P1, !.
random(N,P0,R,[P|Rest],M) :-
        P1 is P+P0, N1 is N+1,
        random(N1,P1,R,Rest,M), !.

randprop(Prop,Dist,Value) :-
        random(Dist,R),
        Call=..[Prop,R,Value],
        Call, !.

car_shape(1,ellipse).  car_shape(2,hexagon).
car_shape(3,rectangle).  car_shape(4,u_shaped). car_shape(5,bucket).
car_length(1,short).  car_length(2,long).
car_open(1,open).  car_open(2,closed).
car_double(1,not_double).  car_double(2,double).
roof_shape(1,none).  roof_shape(2,flat).  roof_shape(3,jagged).
roof_shape(4,peaked). roof_shape(5,arc).
load_shape(1,circle).  load_shape(2,diamond).  load_shape(3,hexagon).
load_shape(4,rectangle).  load_shape(5,triangle).  load_shape(6,utriangle).

show(Train) :-
```

```
        nl,
        (eastbound(Train) -> (write('Eastbound train:'), nl)
        ;otherwise -> (write('Westbound train:'),nl)),
        show0(Train), nl, !.

show0([]).
show0([C|Cs]) :-
        C=c(N,Shape,Length,Double,Roof,Wheels,l(Lshape,Lno)),
        writes(['Car ',N,': Shape = ',Shape,
         ', Length = ',Length,', Double = ',Double,nl, tab(8),
         'Roof = ',Roof,
         ', Wheels = ',Wheels,
         ', Load = ',Lno,' of ',Lshape,nl]),
        show0(Cs), !.

% writes([]).
% writes([H|T]) :-
% mywrite(H),
% writes(T).

mywrite(nl) :- nl, !.
mywrite(tab(X)) :- tab(X), !.
mywrite(X) :- write(X), !.

% Concept tester below emulates Michalski predicates.

has_car(T,C) :- member(C,T).

infront(T,C1,C2) :- append(_,[C1,C2|_],T).

ellipse(C) :- arg(2,C,ellipse).  hexagon(C) :- arg(2,C,hexagon).
rectangle(C) :- arg(2,C,rectangle).  u_shaped(C) :- arg(2,C,u_shaped).
bucket(C) :- arg(2,C,bucket).

long(C) :- arg(3,C,long).  short(C) :- arg(3,C,short).

double(C) :- arg(4,C,double).

has_roof(C,r(R,N)) :- arg(1,C,N), arg(5,C,R).

open(C) :- arg(5,C,none).  closed(C) :- not open(C).

has_wheel(C,w(NC,W)) :- arg(1,C,NC), arg(6,C,NW), nlist(1,NW,L), member(W,L).
```

```
has_load(C,Load) :- arg(7,C,l(_,NLoad)), nlist(1,NLoad,L), member(Load,L).

has_load0(C,Shape) :- arg(7,C,l(Shape,N)), 1=<N.

has_load1(T,Shape) :- has_car(T,C), has_load0(C,Shape).

none(r(none,_)).  flat(r(flat,_)).
jagged(r(jagged,_)).  peaked(r(peaked,_)).
arc(r(arc,_)).

member(X,[X|_]).
member(X,[_|T]) :- member(X,T).

nlist(N,N,[N]) :- !.
nlist(M,N,[M|T]) :-
        M=<N,
        M1 is M+1, nlist(M1,N,T), !.

len1([],0) :- !.
len1([_|T],N) :- len1(T,N1), N is N1+1, !.

append([],L,L) :- !.
append([H|L1],L2,[H|L3]) :-
        append(L1,L2,L3), !.
```

Table 1. Prolog representation of 20 trains.

```
eastbound([c(1,rectangle,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        long,not_double,none,3,l(hexagon,1)),c(3,rectangle,short,
        not_double,peaked,2,l(triangle,1)),c(4,rectangle,long,
        not_double,none,2,l(rectangle,3))]).

eastbound([c(1,rectangle,short,not_double,flat,2,l(circle,2)),c(2,bucket,
        short,not_double,none,2,l(rectangle,1)),c(3,u_shaped,
        short,not_double,none,2,l(triangle,1))]).

eastbound([c(1,rectangle,long,not_double,flat,3,l(utriangle,1)),c(2,hexagon,
        short,not_double,flat,2,l(triangle,1)),c(3,rectangle,
        short,not_double,none,2,l(circle,1))]).

eastbound([c(1,rectangle,short,not_double,none,2,l(rectangle,1)),c(2,ellipse,
```

```
        short,not_double,arc,2,l(diamond,1)),c(3,rectangle,short,
        double,none,2,l(triangle,1)),c(4,bucket,short,not_double,
        none,2,l(triangle,1))]).

eastbound([c(1,rectangle,short,not_double,flat,2,l(circle,1)),c(2,rectangle,
        long,not_double,flat,3,l(rectangle,1)),c(3,rectangle,
        short,double,none,2,l(triangle,1))]).

eastbound([c(1,rectangle,long,not_double,jagged,3,l(rectangle,1)),c(2,hexagon,
        short,not_double,flat,2,l(circle,1)),c(3,rectangle,short,
        not_double,none,2,l(triangle,1)),c(4,rectangle,long,not_double,
        jagged,2,l(rectangle,0))]).

eastbound([c(1,rectangle,long,not_double,none,2,l(hexagon,1)),c(2,rectangle,
        short,not_double,none,2,l(rectangle,1)),c(3,rectangle,
        short,not_double,flat,2,l(triangle,1))]).

eastbound([c(1,rectangle,short,not_double,peaked,2,l(rectangle,1)),c(2,
        bucket,short,not_double,none,2,l(rectangle,1)),c(3,rectangle,
        long,not_double,flat,2,l(circle,1)),c(4,rectangle,short,
        not_double,none,2,l(rectangle,1))]).

eastbound([c(1,rectangle,long,not_double,none,2,l(rectangle,3)),c(2,rectangle,
        short,not_double,none,2,l(circle,1)),c(3,rectangle,long,
        not_double,jagged,3,l(hexagon,1)),c(4,u_shaped,short,
        not_double,none,2,l(triangle,1))]).

eastbound([c(1,bucket,short,not_double,none,2,l(triangle,1)),c(2,u_shaped,
        short,not_double,none,2,l(circle,1)),c(3,rectangle,short,
        not_double,none,2,l(triangle,1)),c(4,rectangle,short,
        not_double,none,2,l(triangle,1))]).

:- eastbound([c(1,rectangle,short,not_double,none,2,l(triangle,1)),
        c(2,rectangle, long,not_double,flat,2,l(circle,3))]).

:- eastbound([c(1,rectangle,long,not_double,jagged,2,l(circle,0)),c(2,u_shaped,
        short,not_double,none,2,l(triangle,1)),c(3,rectangle,short,
        double,none,2,l(circle,1))]).

:- eastbound([c(1,u_shaped,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        long,not_double,flat,3,l(rectangle,1))]).

:- eastbound([c(1,bucket,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        short,not_double,none,2,l(rectangle,1)),c(3,rectangle,
```

```
        long,not_double,jagged,3,l(rectangle,1)),c(4,bucket,short,
        not_double,none,2,l(circle,1))]).

:- eastbound([c(1,rectangle,long,not_double,none,2,l(rectangle,2)),c(2,u_shaped,
        short,not_double,none,2,l(rectangle,1))]).

:- eastbound([c(1,bucket,short,not_double,none,2,l(rectangle,1)),c(2,rectangle,
        long,not_double,flat,2,l(utriangle,3))]).

:- eastbound([c(1,rectangle,long,not_double,none,2,l(hexagon,1)),c(2,rectangle,
        short,not_double,none,2,l(circle,1)),c(3,rectangle,short,
        double,none,2,l(circle,1)),c(4,rectangle,long,not_double,
        none,2,l(rectangle,3))]).

:- eastbound([c(1,u_shaped,short,not_double,none,2,l(triangle,1)),c(2,rectangle,
        long,not_double,none,3,l(rectangle,3))]).

:- eastbound([c(1,rectangle,long,not_double,flat,3,l(rectangle,3)),
        c(2,rectangle, long,not_double,flat,2,l(rectangle,3)),c(3,rectangle,
        long,not_double,none,2,l(rectangle,0)),c(4,u_shaped,short,
        not_double,none,2,l(triangle,1))]).

:- eastbound([c(1,rectangle,long,not_double,flat,3,l(hexagon,1)),c(2,u_shaped,
        short,not_double,none,2,l(triangle,1))]).
```

Table 2. Prolog representation of 100 trains, unclassified.

```
train([c(1,rectangle,long,not_double,flat,2,l(rectangle,3)),c(2,rectangle,
        short,not_double,none,2,l(triangle,1)),c(3,rectangle,
        long,not_double,none,2,l(rectangle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(circle,1)),c(2,bucket,
        short,not_double,flat,2,l(triangle,1)),c(3,rectangle,
        short,not_double,none,2,l(circle,2))]).

train([c(1,rectangle,short,not_double,flat,2,l(circle,1)),c(2,rectangle,
        short,not_double,none,2,l(rectangle,1)),c(3,u_shaped,
        short,not_double,none,2,l(triangle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(rectangle,1)),c(2,rectangle,
        long,not_double,none,2,l(rectangle,3)),c(3,u_shaped,short,
        not_double,none,2,l(triangle,1)),c(4,bucket,short,not_double,
```

```
        none,2,l(circle,1))]).

train([c(1,u_shaped,short,not_double,none,2,l(rectangle,1)),c(2,bucket,
        short,not_double,flat,2,l(circle,1)),c(3,rectangle,short,
        not_double,flat,2,l(triangle,1))]).

train([c(1,u_shaped,short,not_double,none,2,l(circle,1)),c(2,bucket,
        short,not_double,none,2,l(circle,1)),c(3,bucket,short,
        not_double,none,2,l(circle,2))]).

train([c(1,rectangle,long,not_double,flat,2,l(rectangle,1)),c(2,rectangle,
        short,not_double,none,2,l(circle,1)),c(3,u_shaped,short,
        not_double,none,2,l(rectangle,1)),c(4,rectangle,short,
        not_double,none,2,l(triangle,1))]).

train([c(1,u_shaped,short,not_double,none,2,l(triangle,1)),c(2,rectangle,
        short,double,flat,2,l(rectangle,1))]).

train([c(1,rectangle,long,not_double,flat,2,l(utriangle,2)),c(2,rectangle,
        long,not_double,flat,3,l(rectangle,1)),c(3,bucket,short,
        not_double,none,2,l(circle,1)),c(4,rectangle,short,not_double,
        none,2,l(circle,1))]).

train([c(1,rectangle,long,not_double,none,3,l(rectangle,1)),c(2,u_shaped,
        short,not_double,none,2,l(circle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(triangle,1)),c(2,bucket,
        short,not_double,flat,2,l(triangle,1)),c(3,rectangle,
        long,not_double,none,3,l(rectangle,3)),c(4,u_shaped,short,
        not_double,none,2,l(triangle,1))]).

train([c(1,u_shaped,short,not_double,none,2,l(triangle,2)),c(2,bucket,
        short,not_double,none,2,l(rectangle,1)),c(3,rectangle,
        short,not_double,none,2,l(rectangle,1)),c(4,ellipse,short,
        not_double,arc,2,l(rectangle,1))]).

train([c(1,ellipse,short,not_double,arc,2,l(triangle,1)),c(2,u_shaped,
        short,not_double,none,2,l(circle,1))]).

train([c(1,rectangle,long,not_double,jagged,2,l(rectangle,1)),c(2,rectangle,
        short,not_double,none,2,l(triangle,1)),c(3,rectangle,
        short,not_double,none,2,l(triangle,1)),c(4,rectangle,
        short,double,peaked,2,l(circle,1))]).
```

```
train([c(1,u_shaped,short,not_double,none,2,l(triangle,1)),c(2,bucket,
        short,not_double,none,2,l(rectangle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        short,not_double,none,2,l(triangle,1))]).

train([c(1,u_shaped,short,not_double,none,2,l(triangle,1)),c(2,rectangle,
        long,not_double,flat,2,l(circle,1)),c(3,rectangle,short,
        not_double,flat,2,l(circle,1)),c(4,rectangle,long,not_double,
        flat,2,l(rectangle,1))]).

train([c(1,bucket,short,not_double,none,2,l(triangle,1)),c(2,rectangle,
        short,double,flat,2,l(diamond,1)),c(3,rectangle,long,
        not_double,none,2,l(rectangle,3)),c(4,u_shaped,short,
        not_double,none,2,l(diamond,1))]).

train([c(1,ellipse,short,not_double,arc,2,l(rectangle,1)),c(2,u_shaped,
        short,not_double,none,2,l(triangle,1)),c(3,bucket,short,
        not_double,none,2,l(circle,2))]).

train([c(1,rectangle,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        short,not_double,none,2,l(triangle,1)),c(3,bucket,short,
        not_double,none,2,l(triangle,1))]).

train([c(1,rectangle,long,not_double,none,2,l(rectangle,1)),c(2,rectangle,
        long,not_double,flat,3,l(utriangle,1)),c(3,ellipse,short,
        not_double,arc,2,l(circle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(triangle,1)),c(2,rectangle,
        short,double,none,2,l(diamond,1)),c(3,u_shaped,short,
        not_double,none,2,l(triangle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(rectangle,1)),c(2,hexagon,
        short,not_double,flat,2,l(circle,1)),c(3,rectangle,long,
        not_double,none,3,l(rectangle,0))]).

train([c(1,rectangle,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        short,not_double,none,2,l(circle,1)),c(3,rectangle,short,
        not_double,none,2,l(triangle,1)),c(4,rectangle,short,
        not_double,none,2,l(triangle,1))]).

train([c(1,rectangle,long,not_double,jagged,2,l(circle,1)),c(2,rectangle,
        long,not_double,none,2,l(circle,2)),c(3,u_shaped,short,
        not_double,none,2,l(circle,1)),c(4,rectangle,short,not_double,
```

```
        peaked,2,l(diamond,1))]).

train([c(1,bucket,short,not_double,flat,2,l(circle,1)),c(2,bucket,short,
        not_double,none,2,l(triangle,1)),c(3,bucket,short,not_double,
        none,2,l(circle,2)),c(4,rectangle,short,not_double,flat,
        2,l(triangle,1))]).

train([c(1,rectangle,long,not_double,none,2,l(rectangle,3)),c(2,bucket,
        short,not_double,none,2,l(triangle,1)),c(3,rectangle,
        short,not_double,none,2,l(circle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        long,not_double,flat,3,l(hexagon,1)),c(3,rectangle,short,
        double,none,2,l(circle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        short,not_double,flat,2,l(rectangle,1))]).

train([c(1,rectangle,short,double,none,2,l(circle,1)),c(2,rectangle,
        short,not_double,peaked,2,l(triangle,1)),c(3,rectangle,
        long,not_double,jagged,3,l(rectangle,1)),c(4,rectangle,
        short,not_double,none,2,l(diamond,1))]).

train([c(1,rectangle,long,not_double,flat,3,l(hexagon,3)),c(2,rectangle,
        long,not_double,none,2,l(rectangle,3)),c(3,rectangle,
        short,not_double,none,2,l(triangle,1)),c(4,rectangle,
        long,not_double,flat,2,l(rectangle,1))]).

train([c(1,u_shaped,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        long,not_double,none,3,l(circle,1)),c(3,ellipse,short,
        not_double,arc,2,l(triangle,1))]).

train([c(1,ellipse,short,not_double,arc,2,l(triangle,1)),c(2,u_shaped,
        short,not_double,none,2,l(circle,1)),c(3,bucket,short,
        not_double,none,2,l(circle,1)),c(4,bucket,short,not_double,
        peaked,2,l(circle,1))]).

train([c(1,bucket,short,not_double,none,2,l(rectangle,1)),c(2,rectangle,
        short,not_double,none,2,l(diamond,1)),c(3,rectangle,short,
        not_double,peaked,2,l(triangle,1)),c(4,rectangle,long,
        not_double,none,3,l(circle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(triangle,1)),c(2,u_shaped,
        short,not_double,flat,2,l(triangle,1))]).
```

```
train([c(1,ellipse,short,not_double,arc,2,l(triangle,1)),c(2,hexagon,
        short,not_double,flat,2,l(rectangle,1)),c(3,rectangle,
        long,not_double,none,2,l(circle,3)),c(4,rectangle,short,
        not_double,none,2,l(triangle,1))]).

train([c(1,hexagon,short,not_double,flat,2,l(rectangle,1)),c(2,u_shaped,
        short,not_double,none,2,l(circle,1)),c(3,rectangle,long,
        not_double,jagged,2,l(rectangle,1))]).

train([c(1,rectangle,long,not_double,flat,2,l(rectangle,3)),c(2,rectangle,
        short,not_double,none,2,l(triangle,1)),c(3,rectangle,
        long,not_double,jagged,2,l(rectangle,1))]).

train([c(1,u_shaped,short,not_double,none,2,l(triangle,1)),c(2,rectangle,
        short,not_double,none,2,l(circle,1)),c(3,u_shaped,short,
        not_double,none,2,l(rectangle,1))]).

train([c(1,rectangle,short,double,none,2,l(circle,1)),c(2,u_shaped,
        short,not_double,none,2,l(rectangle,1))]).

train([c(1,rectangle,long,not_double,jagged,3,l(rectangle,1)),c(2,bucket,
        short,not_double,none,2,l(triangle,2))]).

train([c(1,rectangle,short,double,none,2,l(triangle,1)),c(2,rectangle,
        short,not_double,none,2,l(rectangle,1))]).

train([c(1,rectangle,short,double,none,2,l(circle,1)),c(2,rectangle,
        short,not_double,none,2,l(rectangle,1))]).

train([c(1,rectangle,long,not_double,jagged,2,l(circle,1)),c(2,rectangle,
        long,not_double,jagged,2,l(rectangle,2)),c(3,rectangle,
        long,not_double,none,2,l(rectangle,3)),c(4,u_shaped,short,
        not_double,none,2,l(rectangle,1))]).

train([c(1,u_shaped,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        short,not_double,none,2,l(triangle,1)),c(3,rectangle,
        long,not_double,flat,3,l(hexagon,1))]).

train([c(1,rectangle,long,not_double,jagged,3,l(rectangle,1)),c(2,ellipse,
        short,not_double,arc,2,l(rectangle,1)),c(3,u_shaped,short,
        not_double,none,2,l(rectangle,1)),c(4,rectangle,short,
        not_double,none,2,l(triangle,2))]).
```

```
train([c(1,u_shaped,short,not_double,none,2,l(triangle,1)),c(2,bucket,
        short,not_double,none,2,l(circle,1)),c(3,rectangle,short,
        not_double,none,2,l(triangle,1))]).

train([c(1,rectangle,long,not_double,none,3,l(circle,1)),c(2,rectangle,
        short,not_double,flat,2,l(triangle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(rectangle,1)),c(2,hexagon,
        short,not_double,flat,2,l(circle,1))]).

train([c(1,u_shaped,short,not_double,none,2,l(triangle,1)),c(2,rectangle,
        long,not_double,jagged,3,l(utriangle,1)),c(3,bucket,short,
        not_double,none,2,l(circle,1)),c(4,rectangle,short,double,
        none,2,l(circle,1))]).

train([c(1,rectangle,short,double,none,2,l(rectangle,1)),c(2,rectangle,
        short,not_double,none,2,l(triangle,1)),c(3,rectangle,
        long,not_double,none,3,l(rectangle,3))]).

train([c(1,rectangle,short,not_double,none,2,l(triangle,1)),c(2,rectangle,
        long,not_double,flat,3,l(utriangle,1)),c(3,bucket,short,
        not_double,none,2,l(triangle,1)),c(4,u_shaped,short,not_double,
        none,2,l(rectangle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(rectangle,1)),c(2,rectangle,
        short,double,none,2,l(circle,1))]).

train([c(1,rectangle,short,not_double,peaked,2,l(circle,1)),c(2,rectangle,
        long,not_double,flat,3,l(hexagon,0)),c(3,rectangle,short,
        not_double,flat,2,l(circle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        short,not_double,none,2,l(circle,1)),c(3,bucket,short,
        not_double,none,2,l(circle,1))]).

train([c(1,u_shaped,short,not_double,none,2,l(triangle,1)),c(2,rectangle,
        long,not_double,none,3,l(rectangle,3)),c(3,rectangle,
        short,double,none,2,l(circle,1)),c(4,rectangle,short,
        not_double,none,2,l(circle,1))]).

train([c(1,rectangle,long,not_double,jagged,3,l(rectangle,1)),c(2,rectangle,
        long,not_double,jagged,2,l(hexagon,3)),c(3,rectangle,
        short,not_double,none,2,l(triangle,1))]).
```

```
train([c(1,bucket,short,not_double,peaked,2,l(circle,1)),c(2,hexagon,
        short,not_double,flat,2,l(circle,1)),c(3,rectangle,short,
        not_double,none,2,l(rectangle,1))]).

train([c(1,u_shaped,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        long,not_double,jagged,2,l(rectangle,1)),c(3,bucket,short,
        not_double,none,2,l(triangle,1)),c(4,rectangle,short,
        double,none,2,l(triangle,1))]).

train([c(1,rectangle,short,double,none,2,l(circle,1)),c(2,bucket,short,
        not_double,flat,2,l(circle,1)),c(3,rectangle,short,double,
        none,2,l(circle,1)),c(4,rectangle,short,not_double,peaked,
        2,l(triangle,1))]).

train([c(1,rectangle,long,not_double,none,3,l(rectangle,1)),c(2,rectangle,
        long,not_double,flat,2,l(rectangle,1)),c(3,rectangle,
        short,double,none,2,l(circle,1)),c(4,bucket,short,not_double,
        peaked,2,l(rectangle,1))]).

train([c(1,rectangle,short,double,flat,2,l(triangle,1)),c(2,rectangle,
        long,not_double,none,3,l(rectangle,1)),c(3,ellipse,short,
        not_double,arc,2,l(rectangle,2))]).

train([c(1,hexagon,short,not_double,flat,2,l(triangle,1)),c(2,bucket,
        short,not_double,none,2,l(triangle,1)),c(3,rectangle,
        short,not_double,peaked,2,l(triangle,1)),c(4,rectangle,
        short,not_double,none,2,l(triangle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(triangle,1)),c(2,rectangle,
        short,not_double,none,2,l(circle,1))]).

train([c(1,rectangle,short,not_double,flat,2,l(circle,1)),c(2,rectangle,
        long,not_double,jagged,3,l(rectangle,3)),c(3,u_shaped,
        short,not_double,none,2,l(circle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(triangle,1)),c(2,rectangle,
        long,not_double,jagged,3,l(rectangle,2)),c(3,rectangle,
        short,not_double,none,2,l(triangle,1)),c(4,rectangle,
        short,double,none,2,l(triangle,1))]).

train([c(1,rectangle,long,not_double,flat,3,l(rectangle,2)),c(2,rectangle,
        long,not_double,none,2,l(rectangle,2)),c(3,rectangle,
        short,not_double,flat,2,l(circle,1))]).
```

```
train([c(1,rectangle,long,not_double,flat,3,l(rectangle,3)),c(2,rectangle,
        short,double,none,2,l(diamond,1)),c(3,rectangle,long,
        not_double,none,2,l(utriangle,1)),c(4,rectangle,short,
        not_double,none,2,l(circle,1))]).

train([c(1,rectangle,long,not_double,flat,3,l(rectangle,0)),c(2,rectangle,
        long,not_double,none,3,l(rectangle,0))]).

train([c(1,rectangle,long,not_double,flat,2,l(circle,1)),c(2,rectangle,
        short,not_double,none,2,l(circle,1)),c(3,rectangle,long,
        not_double,none,2,l(rectangle,3))]).

train([c(1,u_shaped,short,not_double,none,2,l(rectangle,1)),c(2,rectangle,
        long,not_double,flat,3,l(rectangle,1)),c(3,rectangle,
        short,double,none,2,l(triangle,1)),c(4,bucket,short,not_double,
        none,2,l(triangle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        short,not_double,none,2,l(diamond,1)),c(3,bucket,short,
        not_double,flat,2,l(circle,1))]).

train([c(1,hexagon,short,not_double,flat,2,l(triangle,1)),c(2,bucket,
        short,not_double,peaked,2,l(circle,1))]).

train([c(1,u_shaped,short,not_double,flat,2,l(circle,1)),c(2,ellipse,
        short,not_double,arc,2,l(circle,1)),c(3,rectangle,long,
        not_double,none,3,l(rectangle,1)),c(4,bucket,short,not_double,
        none,2,l(circle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(triangle,1)),c(2,rectangle,
        short,not_double,none,2,l(rectangle,1))]).

train([c(1,ellipse,short,not_double,arc,2,l(rectangle,1)),c(2,rectangle,
        long,not_double,none,3,l(hexagon,1))]).

train([c(1,rectangle,long,not_double,flat,2,l(rectangle,0)),c(2,rectangle,
        short,not_double,none,2,l(triangle,1)),c(3,bucket,short,
        not_double,none,2,l(triangle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        long,not_double,flat,3,l(rectangle,3)),c(3,rectangle,
        short,not_double,none,2,l(circle,1)),c(4,rectangle,long,
        not_double,flat,2,l(rectangle,1))]).
```

```
train([c(1,rectangle,long,not_double,none,2,l(hexagon,1)),c(2,rectangle,
        short,not_double,none,2,l(triangle,1)),c(3,hexagon,short,
        not_double,flat,2,l(circle,1)),c(4,rectangle,short,not_double,
        none,2,l(circle,1))]).

train([c(1,hexagon,short,not_double,flat,2,l(circle,1)),c(2,rectangle,
        long,not_double,flat,2,l(rectangle,3)),c(3,rectangle,
        short,not_double,flat,2,l(rectangle,1)),c(4,bucket,short,
        not_double,none,2,l(triangle,1))]).

train([c(1,rectangle,long,not_double,flat,3,l(hexagon,1)),c(2,bucket,
        short,not_double,none,2,l(rectangle,1)),c(3,rectangle,
        short,not_double,none,2,l(circle,1))]).

train([c(1,rectangle,long,not_double,flat,3,l(rectangle,2)),c(2,rectangle,
        long,not_double,flat,3,l(rectangle,3)),c(3,rectangle,
        long,not_double,flat,2,l(rectangle,3)),c(4,bucket,short,
        not_double,none,2,l(triangle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(triangle,1)),c(2,rectangle,
        long,not_double,flat,2,l(hexagon,3))]).

train([c(1,rectangle,long,not_double,flat,2,l(hexagon,1)),c(2,rectangle,
        long,not_double,jagged,2,l(rectangle,1)),c(3,rectangle,
        long,not_double,jagged,2,l(hexagon,0))]).

train([c(1,u_shaped,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        long,not_double,flat,2,l(rectangle,3)),c(3,rectangle,
        short,not_double,none,2,l(circle,1)),c(4,rectangle,short,
        not_double,none,2,l(rectangle,1))]).

train([c(1,rectangle,short,double,flat,2,l(rectangle,1)),c(2,bucket,
        short,not_double,none,2,l(triangle,1)),c(3,bucket,short,
        not_double,none,2,l(diamond,1))]).

train([c(1,rectangle,long,not_double,flat,3,l(rectangle,3)),c(2,bucket,
        short,not_double,none,2,l(triangle,2)),c(3,bucket,short,
        not_double,none,2,l(triangle,1)),c(4,hexagon,short,not_double,
        flat,2,l(circle,1))]).

train([c(1,rectangle,long,not_double,jagged,3,l(rectangle,2)),c(2,rectangle,
        short,not_double,none,2,l(circle,1)),c(3,rectangle,long,
        not_double,flat,2,l(rectangle,1))]).
```

```
train([c(1,rectangle,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        short,not_double,none,2,l(rectangle,1)),c(3,bucket,short,
        not_double,none,2,l(circle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(circle,1)),c(2,rectangle,
        long,not_double,flat,2,l(rectangle,3)),c(3,rectangle,
        short,not_double,none,2,l(rectangle,1))]).

train([c(1,rectangle,long,not_double,flat,3,l(utriangle,1)),c(2,u_shaped,
        short,not_double,flat,2,l(triangle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(triangle,1)),c(2,rectangle,
        short,not_double,none,2,l(circle,1))]).

train([c(1,u_shaped,short,not_double,flat,2,l(circle,1)),c(2,rectangle,
        long,not_double,flat,3,l(utriangle,0)),c(3,rectangle,
        long,not_double,flat,3,l(circle,1)),c(4,rectangle,short,
        double,none,2,l(circle,1))]).

train([c(1,u_shaped,short,not_double,peaked,2,l(triangle,1)),c(2,ellipse,
        short,not_double,arc,2,l(diamond,1)),c(3,rectangle,long,
        not_double,none,2,l(rectangle,1)),c(4,rectangle,long,
        not_double,none,2,l(rectangle,1))]).

train([c(1,rectangle,short,double,none,2,l(triangle,1)),c(2,rectangle,
        long,not_double,flat,2,l(circle,1))]).

train([c(1,rectangle,short,double,none,2,l(triangle,1)),c(2,bucket,
        short,not_double,none,2,l(triangle,1)),c(3,hexagon,short,
        not_double,flat,2,l(circle,1)),c(4,rectangle,long,not_double,
        none,2,l(rectangle,3))]).

train([c(1,rectangle,long,not_double,flat,2,l(rectangle,3)),c(2,hexagon,
        short,not_double,flat,2,l(circle,2))]).

train([c(1,rectangle,long,not_double,flat,3,l(rectangle,1)),c(2,rectangle,
        short,not_double,peaked,2,l(circle,1))]).

train([c(1,rectangle,short,not_double,none,2,l(triangle,1)),c(2,rectangle,
        short,not_double,peaked,2,l(triangle,1))]).

train([c(1,u_shaped,short,not_double,none,2,l(triangle,1)),c(2,bucket,
        short,not_double,none,2,l(circle,1)),c(3,bucket,short,
        not_double,none,2,l(rectangle,1))]).
```

Appendix A Details of Complexity Measurement

The definitions of *clause occurrence*, *atom occurrence*, and *term occurrence* that are clear for first-order definite clause logic are ambiguous for Prolog in general. This ambiguity arises because some of the predicates described in Clocksin and Mellish's *Programming in Prolog* (3^{rd} edition, 1987) are actually *meta-logical*. Specifically, some predicates (more precisely, predicate symbols) may be used to build atoms with arguments that are actually goals (atoms or combinations of atoms) rather than terms. These predicates include *not*, *findall*, *call*, and the semicolon operator (for disjunction). In the presence of such meta-logical predicates, the definitions of *clause occurrence*, *atom occurrence*, and *term occurrence* become unclear. For example, consider the following theory.

Example 1

```
eastbound(T) :- findall(C,(has_car(T,C),has_load0(C,triangle)), L),
                length(L) > 2.
```

Should the occurrence of *has_car*(T,C) in this theory be counted as an atom occurrence (since it is built from a predicate symbol), a term occurrence (since it is an argument to a predicate symbol), or both? Even more fundamentally, this theory does not actually contain *any* clause occurrences under the standard definition, since

findall$(C,($*has_car*$(T,C),$ *has_load0*$(C,$*triangle*$)),L)$

is not a literal under the standard definition. As another example, consider the theory

Example 2

```
eastbound(T) :- has_car(T,C), (has_roof(C,jagged) ;
                               has_roof(C,peaked) ;
                               has_load(T,utriangle)).
```

This theory does not contain a well-defined clause either, since the semicolon represents disjunction. Nevertheless, it is a correct Prolog theory. We now extend the definitions of *term occurrence*, *atom occurrence*, and *clause occurrence*, in what seems to be the most natural way possible, to apply to all Prolog theories rather than only to pure first-order definite clause theories.

Our extension of the definitions simply identifies a substring of a Prolog program as a *term occurrence*, *atom occurrence*, or *clause occurrence* if it is labeled as a *term*, *atom*, or *clause*, respectively, during a parse of the Prolog program according to an unambiguous context-free grammar for Prolog. In addition to presenting such a grammar, below, we provide a high-level description in the following paragraph. For complete clarity, we also provide Ashwin Srinivasan's Prolog program for measuring the complexity of any theory (this code also appears in the file complex.pl).

A *term occurrence* (as standard) has the form either X (a variable) or $f(t_1, ..., t_n)$, where f is an n-ary function symbol and $t_1, ..., t_n$ are terms. Shorthand expressions for terms, most notably the representation of lists in the form [1,2,3], are expanded before complexity is measured. Thus, for example, the complexity of [1,2,3] is 7, since the expanded representation is

$$cons(1, cons(2, cons(3, nil))).$$

An *atom occurence* has the form $p(t_1, ..., t_n)$, where either (1) p is an *ordinary* predicate symbol and $t_1, ..., t_n$ are terms (in which case we call the atom *ordinary*), or (2) p is a *meta-logical* predicate symbol, and each t_i is a term or a goal as specified for that predicate in Clocksin and Mellish (in which case we call the atom *meta-logical*). For example, in a meta-logical atom $not(t)$, t must itself be a goal, while in the meta-logical atom $findall(t_1, t_2, t_3)$, t_1 must be a variable, t_2 must be a goal, and t_3 must be a term denoting a list. It is worth special observation that a *cut* is treated as an atom (built from the nullary predicate symbol '!'). A *clause occurrence* has either form (1) 'A.' or (2) '$A : -B_1, ..., B_n$.' where A is an ordinary atom and $B_1, ..., B_n$ are atoms.

Returning to the earlier examples, the theory in Example 1 consists of one clause, containing one occurrence of each of the following 5 atoms:

```
eastbound(T)
findall(C,(has_car(T,C),has_load0(C,triangle)),L),
has_car(T,C)
has_load0(C,triangle)
length(L) > 2.
```

The theory also contains 9 term occurrences, for a total complexity score of 15. What about the theory in Example 2? It consists of one clause, containing one occurrence of each of the following 7 atoms:

```
eastbound(T)
has_car(T,C)
```

```
;(has_roof(C,jagged),;(has_roof(C,peaked),has_load(T,utriangle)))
;(has_roof(C,peaked),has_load(T,utriangle))
has_roof(C,jagged)
has_roof(C,peaked)
has_load(T,utriangle)
```

This theory has 9 term occurrences, for a total complexity score of 17.

A.1 An LL(1) grammar for Prolog

The following grammar is an LL(1)—and therefore unambiguous context-free—grammar for Prolog, together with our symbols and any new symbols that may be introduced to define a theory, given that each new symbol is appropriately inserted into the grammar. (The grammar is not complete, but we believe it makes obvious the complete LL(1) grammar.) Note that meta-logical predicates have been accorded special treatment, since some of their arguments may be goals rather than terms. It is also worth noting that, to keep matters simple, we view the list syntax as syntactic sugar, giving shorthand for terms built from the cons function (e.g., $[X,Y]$ is $cons(X,cons(Y,nil))$, and $[X|Y]$ is $cons(X,Y)$), and we view infix representation of operators such as '=' and ';' as syntactic sugar for a prefix representation. Nonterminals are alphanumeric strings beginning with a capital letter; terminals are strings enclosed in single quotes.

LL(1) Grammar for Prolog

```
Theory    ::= Clause Theory | `end-of-file´
Clause    ::= OrdAtom Remainder
Remainder ::= `.´ | `:-´ Body
Body      ::= Atom Body1
Body1     ::= `.´ | `,´ Body
Goal      ::= Atom | `(´ Comp_goal
Comp_goal ::= Atom Goal1
Goal1     ::= `)´ | `,´ Comp_goal
OrdAtom   ::= PredSym_0 |
              PredSym_1 `(´ Term `)´ |
              PredSym_2 `(´ Term `,´ Term `)´ |
              .
              .
Atom      ::= OrdAtom |
              `findall´ `(´ Term `,´ Goal `,´ LTerm `)´ |
              `;´ `(´ Goal `,´ Goal `)´
```

```
              `not´ `(´ Comp_goal |
              `call´ `(´ Comp_goal |
              .
              .
Term      ::= VarSym |
              FuncSym_0 |
              FuncSym_1 `(´ Term `)´ |
              FuncSym_2 `(´ Term `,´ Term `)´ |
              .
              .
LTerm     ::= VarSym |
              `nil´ |
              `cons´ `(´ Term `,´ LTerm `)´
PredSym_0 ::= `!´ |
              .
              .
PredSym_1 ::= `eastbound´ |
              .
              .
PredSym_2 ::= `has_car´ |
              .
              .
FuncSym_0 ::= `triangle´ |
              `jagged´ |
              .
              .
              Number
FuncSym_1 ::= `roof-shape´ |
              .
              .
FuncSym_2 ::= `cons´ |
              .
              .
Number    ::= Integer | Floating-Point
```

(The productions for Integer and Floating Point are omitted.)

A.2 A Prolog program for measuring complexity of theories in Competition 1

The Prolog program for measuring theory complexity follows.

```
% Ashwin Srinivasan´s Prolog code
```

```
% for measuring theory complexity.

% dynamic statements are only for compiled Prologs
% (not Clocksin and Mellish standard)

:- dynamic counts/2.

% count clauses, literals and terms in file FileName
count(FileName):-
        reset_counts,
        see(FileName),
        count_clauses,
        seen,
        print_theory_counts.

count_clauses:-
        repeat,
        read(Clause),
        count_clause(Clause),
        Clause = end_of_file,
        !.

count_clause(end_of_file):- !.
count_clause((Head:-Body)):-
        !,
        inc(clauses,1),
        get_litterm_count((Head,Body)).
count_clause(UnitClause):-
        inc(clauses,1),
        get_litterm_count(UnitClause).


get_litterm_count((LitTerm;LitTerms)):-
        !,
        inc(litterms,1), % for ´;´/2
        get_litterm_count(LitTerm),
        get_litterm_count(LitTerms).
get_litterm_count((LitTerm,LitTerms)):- % no charge for ´,´/2
        !,
        get_litterm_count(LitTerm),
        get_litterm_count(LitTerms).
get_litterm_count(LitTerm):-
        inc(litterms,1), % for Lit
        functor(LitTerm,Name,Arity),
```

```
        get_arg_count(LitTerm,Arity,0,T0),
        inc(litterms,T0).

get_arg_count(_,0,LT,LT).
get_arg_count(Expr,Arg,T,LitTerms):-
        arg(Arg,Expr,Term),
        var(Term), !,
        Arg0 is Arg - 1,
        T1 is T + 1,
        get_arg_count(Expr,Arg0,T1,LitTerms).
get_arg_count(Expr,Arg,T,LitTerms):-
        arg(Arg,Expr,LitTerm),
        functor(LitTerm,LitTermName,LitTermArity),
        inc_term_count(LitTermName/LitTermArity,T,T1),
        get_arg_count(LitTerm,LitTermArity,T1,T2),
        Arg0 is Arg - 1,
        get_arg_count(Expr,Arg0,T2,LitTerms).

inc_term_count(´,´/2,T,T):- % no charge for ´,´/2
        !.
inc_term_count(_,T,T1):-
        T1 is T + 1.

reset_counts:-
        retractall(counts(_,_)),
        asserta(counts(clauses,0)),
        asserta(counts(litterms,0)).

print_theory_counts:-
        counts(clauses,C),
        write(´clauses:´), write(C), nl,
        counts(litterms,LT),
        write(´lits+terms:´), write(LT), nl, nl,
        Total is C + LT,
        write(´total:´), write(Total), nl.


inc(Parse,N):-
        retract(counts(Parse,N1)),
        N0 is N1 + N,
        asserta(counts(Parse,N0)).
```

We close this Appendix with an extensive list of additional examples.

We omit the phrase *occurrence* in presenting the complexity calculations.

```
eastbound(T):-
         findall(C,(has_car(T,C),short(C)),[_,_,_]),
         has_car(T,Car),
         not(arg(6,C,3)).

1 Clause + 7 Atoms + 17 Terms = 25

Prolog program output:

        clauses: 1
        lits+terms: 24

        total: 25


westbound(T):-
         has_car(T,C),
         (
         closed(C),has_load0(C,triangle)
         )
         ;
         has_load0(C,hexagon).

1 Clause + 6 Atoms + 8 Terms = 15

Prolog program output:

        clauses: 1
        lits+terms: 14

        total: 15

eastbound(T):-
         (
         in_front(T,C1,C2),
         has_load0(C1,S),
         has_load0(C2,S)
         )
         ;
         findall(C,(has_car(T,C),not(double(C))),[_,_]).
```

```
1 Clause + 9 Atoms + 17 Terms = 27

Prolog program output:

        clauses: 1
        lits+terms: 26

        total: 27

westbound(T):-
         (
         has_car(T,C),
         arg(6,C,3)
         )
         ;
         (
         append(_,[FirstCar],T),
         short(FirstCar),
         has_load0(FirstCar,rectangle)
         ).

1 Clause + 7 Atoms + 14 Terms = 22

Prolog program output:

        clauses: 1
        lits+terms: 21

        total: 22

westbound(T):-
         append(_,[FirstCar],T),
         (
         has_load0(FirstCar,triangle)
         ;
         hexagon(FirstCar)
         ;
         double(FirstCar)
         ).

1 Clause + 7 Atoms + 10 Terms = 18

Prolog program output:
```

```
        clauses: 1
        lits+terms: 17

        total: 18

eastbound([LastCar|Cars]):-
         short(LastCar),
         rectangle(LastCar),
         findall(C,(has_car(Cars,C),long(C)),LongCars),
         (
         LongCars = []
         ;
         LongCars = [_]
         ).

1 Clause + 9 Atoms + 16 Terms = 26

Prolog program output:

        clauses: 1
        lits+terms: 25

        total: 26

eastbound(T):-
         (
         in_front(T,C1,C2),
         short(C1), short(C2),
         has_load0(C1,S), has_load0(C2,S)
         )
         ;
         (
         has_car(T,C1), has_car(T,C2),
         C1 \= C2,
         has_load0(C1,_), has_load0(C2,_),
         arg(5,C1,jagged), arg(5,C2,jagged)
         ).

1 Clause + 14 Atoms + 26 Terms = 41

Prolog program output:

        clauses: 1
        lits+terms: 40
```

```
        total: 41

eastbound(T):-
         has_car(T,C),
         (
         arg(5,C,peaked)
         ;
         ellipse(C)
         ;
         (closed(C), rectangle(C))
         ;
         T = [_,_]
         ).

1 Clause + 10 Atoms + 15 Terms = 26

Prolog program output:

        clauses: 1
        lits+terms: 25

        total: 26

westbound(T):-
         append(_,[FirstCar],T),
         (
         (open(FirstCar), bucket(FirstCar))
         ;
         hexagon(FirstCar)
         ;
         (rectangle(FirstCar), has_load0(FirstCar,rectangle))
         ).

1 Clause + 9 Atoms + 12 Terms = 22

Prolog program output:

        clauses: 1
        lits+terms: 21

        total: 22

eastbound(T):-
```

```
          (
          (T = [C1,C2],
           (
           short(C1)
           ;
           short(C2)
           )
          )
          ;
          (append(_,[C2,C1],T),
           (
           (has_load0(C2,triangle), has_load0(C1,triangle))
           ;
           (has_load0(C2,rectangle), has_load0(C1,rectangle))
           )
          )
          ).

1 Clause + 12 Atoms + 24 Terms = 37

Prolog program output:

        clauses: 1
        lits+terms: 36

        total: 37

eastbound(T):-
          append(_,[FirstCar],T),
          (
          bucket(FirstCar)
          ;
          (open(FirstCar), rectangle(FirstCar))
          ).

1 Clause + 6 Atoms + 9 Terms = 16

Prolog program output:

        clauses: 1
        lits+terms: 15

        total: 16
```

```
eastbound(T):-
         has_car(T,C),
         (
         double(C)
         ;
         arg(5,C,peaked)
         ;
         ellipse(C)
         ).
```

1 Clause + 7 Atoms + 8 Terms = 16

Prolog program output:

```
        clauses: 1
        lits+terms: 15

        total: 16
```

```
eastbound(T):-
         has_car(T,C),
         (
         long(C), closed(C),
          (
          has_load(C,1)
          ;
          (has_load(C,2),has_load0(C,circle))
          )
         ;
         double(C)
         ).
```

1 Clause + 10 Atoms + 12 Terms = 23

Prolog program output:

```
        clauses: 1
        lits+terms: 22

        total: 23
```

```
westbound(T):-
         (
         (T = [_,_], has_car(T,C), short(C))
```

```
         ;
         (has_car(T,C), arg(5,C,jagged))
         ).
```

1 Clause + 7 Atoms + 14 Terms = 22

Prolog program output:

```
        clauses: 1
        lits+terms: 21

        total: 22
```

```
eastbound(T):-
        findall(C,(has_car(T,C),long(C), rectangle(C), arg(5,C,flat)),[_])
        ;
        findall(C,(has_car(T,C),short(C)),[_,_,_]).
```

1 Clause + 10 Atoms + 23 Terms = 34

Prolog program output:

```
        clauses: 1
        lits+terms: 33

        total: 34
```

Figure 1: Michalski's original set of trains.

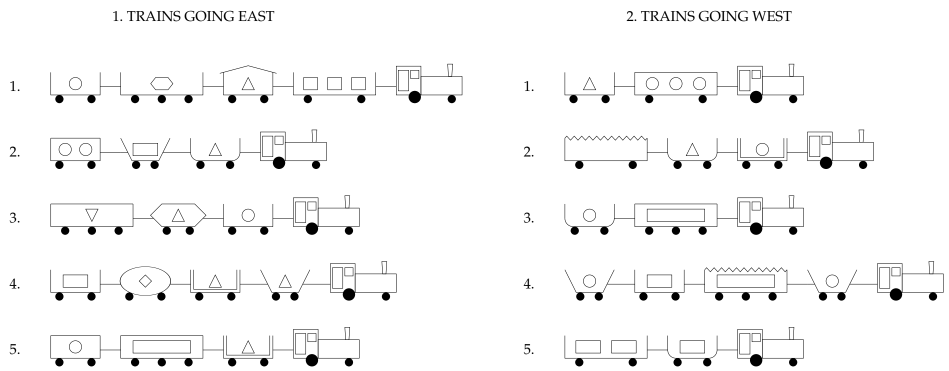


Figure 2: A new set of 10 trains concocted with the aid of Muggleton's train generator.

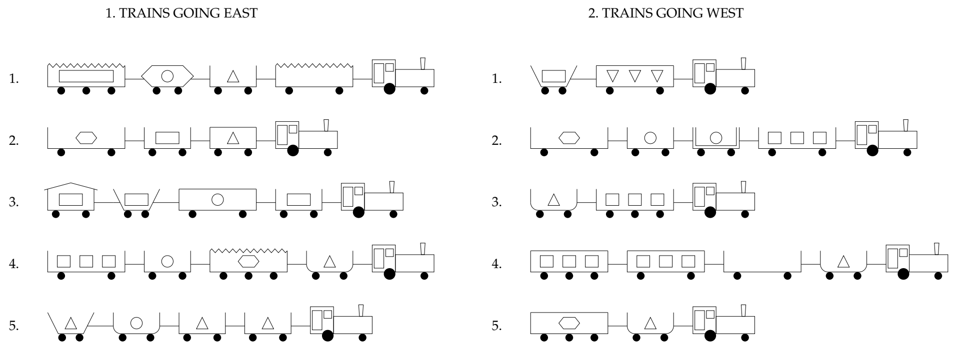
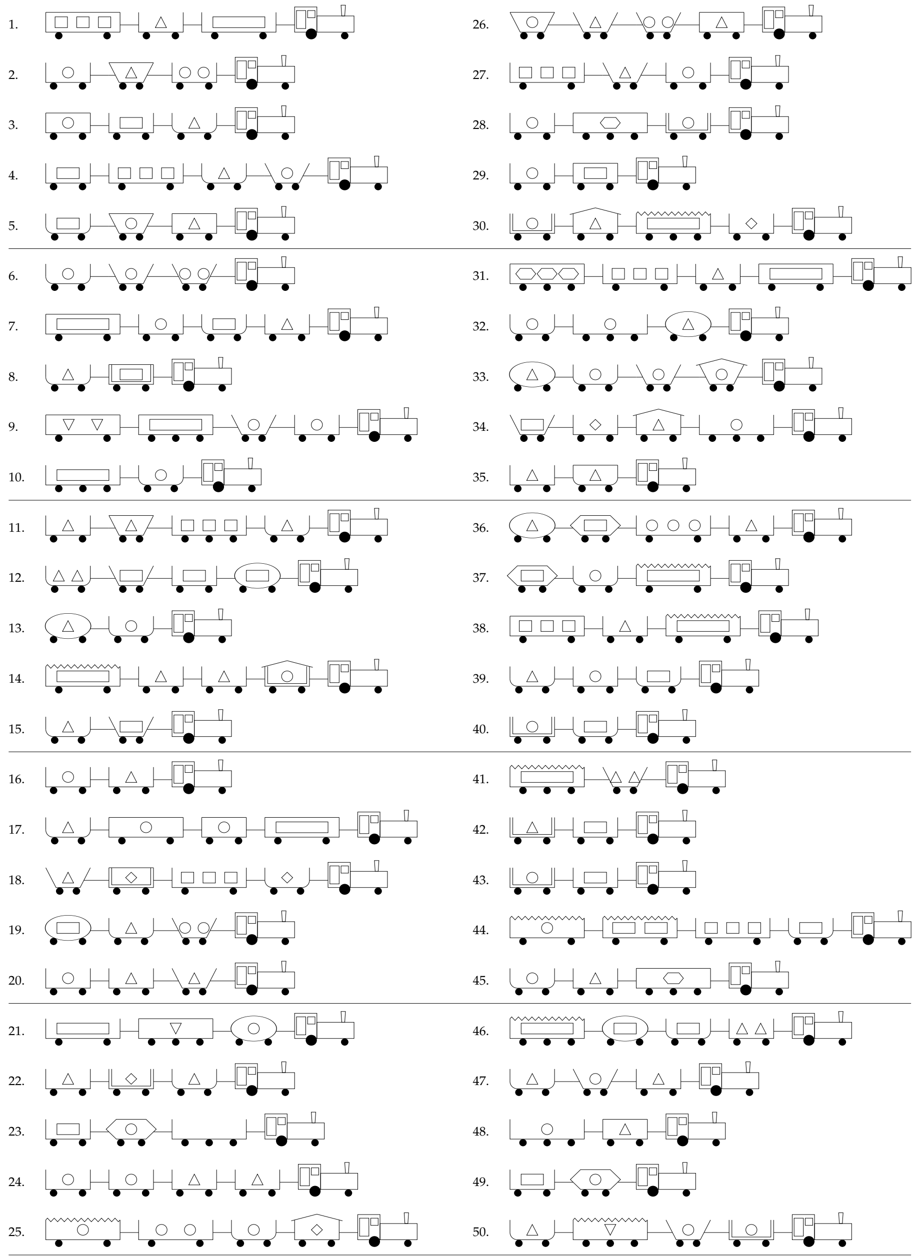
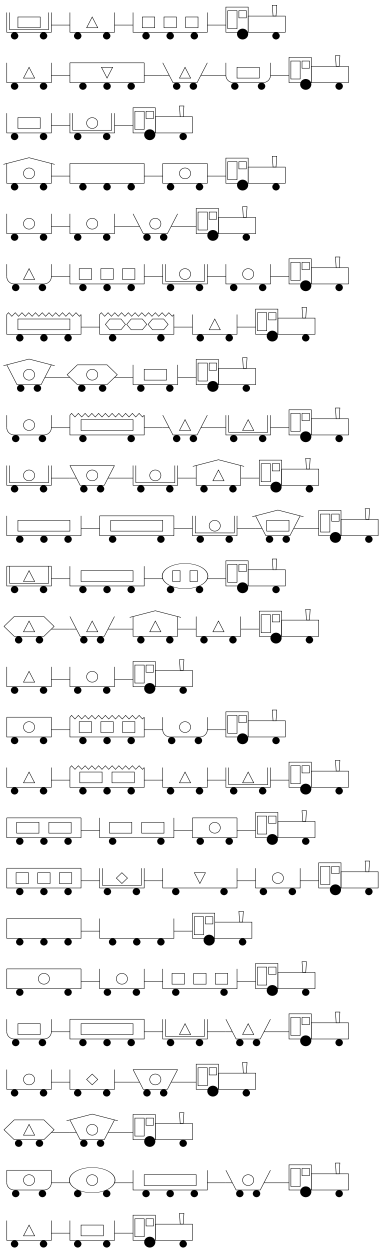


Figure 3: 100 more trains, unclassified, randomly generated by Muggleton's train generator. The cross-ruling added to this page are for purposes of Competition 3.



51.
52.
53.
54.
55.
56.
57.
58.
59.
60.
61.
62.
63.
64.
65.
66.
67.
68.
69.
70.
71.
72.
73.
74.
75.



76.
77.
78.
79.
80.
81.
82.
83.
84.
85.
86.
87.
88.
89.
90.
91.
92.
93.
94.
95.
96.
97.
98.
99.
100.