Programming Research Group

EXPERIMENTS IN NUMERICAL REASONING WITH INDUCTIVE LOGIC PROGRAMMING

Ashwin Srinivasan and Rui Camacho

PRG-TR-22-96

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

Abstract

Using problem-specific background knowledge, computer programs developed within the framework of Inductive Logic Programming (ILP) have been used to construct restricted first-order logic solutions to scientific problems. However, their approach to the analysis of data with substantial numerical content has been largely limited to constructing clauses that: (a) provide qualitative descriptions ("high", "low" etc.) of the values of response variables; and (b) contain simple inequalities restricting the ranges of predictor variables. This has precluded the application of such techniques to scientific and engineering problems requiring a more sophisticated approach. A number of specialised methods have been suggested to remedy this. In contrast, we have chosen to take advantage of the fact that the existing theoretical framework for ILP places very few restrictions of the nature of the background knowledge. In principle, it is therefore possible to use background predicates that implement well-established statistical and numerical analysis procedures to achieve improved analytical sophistication. The experiments reported here are intended to demonstrate this. The experiments utilise artificial datasets. These are concerned with extracting constraints for response variables in the text-book problem of balancing a pole on a cart, and a pilot performing a levelled turn on an aircraft simulator. They illustrate the use as background knowledge of clausal definitions of arithmetic and trigonometric functions, inequalities, elementary plane geometry, multiple linear regression, and numerical derivatives. In all experiments, the predictions obtained compare favourably against benchmarks set by more traditional methods of quantitative methods, namely, regression and neural-network.

1 Introduction

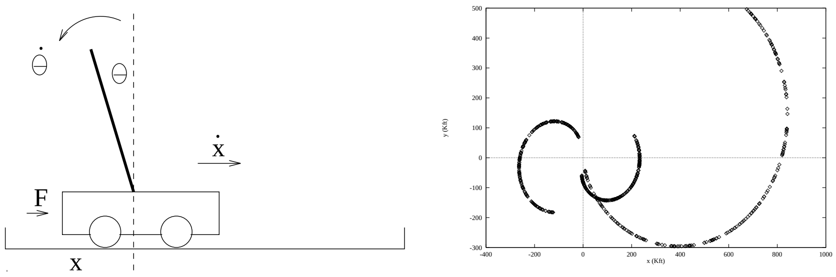
The framework defining Inductive Logic Programming (ILP: see [21]), has seen the advent of efficient, general-purpose programs capable of using problem-specific background knowledge to construct automatically clausal definitions that in some sense, generalise a set of instances. This has allowed a novel form of data analysis in molecular biology [16, 17, 23], stress analysis in engineering [7], electronic circuit diagnosis [10], environmental monitoring [9], software engineering [1], and natural language processing [35]. Of these, some, such as those described in [1, 9, 10, 23, 35], are naturally classificatory. Others, such as those described in [7, 16, 17], are essentially concerned with predicting values of a numerical "response" variable (for example, chemical activity of a compound). For problems of this latter type, ILP programs have largely been restricted to constructing definitions that are only capable of qualitative predictions (for example, "high", "low" etc.). Further, if the definition involves the use of any numerical "predictor" variables, then this usually manifests itself as inequalities that restrict the ranges of such variables. This apparent limitation of ILP programs has been of some concern, and rates highly on the priorities of at least one prominent research programme designed to address the shortcomings of ILP [5].

In theory, any form of numerical reasoning *could be* achieved from first principles by an ILP program. Thus, much of the limitations stated above must stem from practical constraints placed on ILP programs. Some of these constraints pertain to ILP programs like those described in [25, 28], where background knowledge is restricted to ground unit clauses. But what about programs capable of understanding background knowledge that includes more complex logical descriptions? Such programs are in some sense closer to the spirit of the ILP framework defined in [21]. In this paper, we explore the possibility of improving the numerical capabilities of such an ILP program by the straightforward approach of including as background knowledge predicates that perform numerical and statistical calculations. In particular, by the phrase "numerical capabilities" we are referring to the ability to construct descriptions that may require at least the following:

- Arithmetic and trigonometric functions;
- Equalities and inequalities;
- Regression models (including equations constructed by linear or non-linear regression); and
- Geometric models (that is, planar shapes detected in the data).

An ILP program capable of using such primitives would certainly be able to provide more quantitative solutions to the molecular biology and stress analysis problems cited earlier. At various stages, the problems considered in this paper require an ILP program to use some or all of these as background knowledge.

The experiments reported here are concerned with extracting constraints for numerical variables from simulator data records of two control tasks. Balancing a pole on a cart is a text-book problem in control engineering, and has been a test-bed for evaluating the use of machine learning programs to extract comprehensible descriptions summarising extensive simulator records of controller behaviour. A more complex test-bed, with similar objectives, has involved the use of an aircraft simulator. For both problems data records are usually tabulations of the values of numerical-valued variables, and so far, feature-based machine learning programs either equipped with built-in definitions for inequalities or those capable of regression-like behaviour have been used to analyse such data. The constraints to be extracted here are in the form of equations describing: (a) linear and angular accelerations for the pole-and-cart system, using the variables describing the system state (see Figure 1a); and (b) angle of bank adopted by a pilot performing a levelled turn on a F-16 simulator, using trajectory information (see Figure 1b). There are some advantages to the tasks chosen. First, the simulations provide ready access to data records. Second, the nature of the equations to be extracted is relatively straightforward, and in some cases known exactly prior to the experiments (from the dynamics of the physical systems concerned: see Appendix A). This allows us to focus on the question of whether the ILP program is able to reconstruct these equations. Finally, it is expected that an adequate reconstruction of the constraints would require the use of predicates characteristic of the four numerical primitives listed earlier, and thus serve as a good illustration of their use within the ILP framework.



(a) Pole-and-cart (b) Flight simulator

Figure 1: Problem domains considered in this paper. The pole-and-cart problem refers to maintaining a pole balanced on a cart and keeping the cart within pre-specified track limits. The variables annotating the left hand figure are: x, position of the centre of mass of the cart along the X axis; $\dot{x}$, linear velocity of the system; θ, angle of inclination of the pole; and $\dot{\theta}$, the angular velocity of the pole. Also calculable are the linear and angular accelerations $\ddot{x}$ and $\ddot{\theta}$. The mass of the pole is denoted by m and that of the cart by M. F refers to a fixed magnitude force that can be exerted on the cart to push it to the left or right ("bang-bang" control). The right hand figure is the plan view of a number of turns being executed by a pilot on a flight simulator. Each turn is performed at a different angle of bank. The horizontal position of the simulated aircraft are given by its X,Y coordinates. For the purposes of this figure, all turns are normalised to ensure that the origin of cartesian coordinates is at the start of the turn.

The paper is organised as follows. Section 2 clarifies the scope of this study. Materials available are described in Section 3, experimental design and results for the pole-and-cart problem are in Section 4.2 and those for the aircraft simulator are in Section 5.2. A discussion of the results obtained for both tasks is in Section 6. Section 7 concludes this study. Several appendices augment the main content of this paper. Appendix A describes the numerical constraints governing the two problems considered here. Appendix B details features of the ILP program used that play an important role in experiments described here. Appendix C contains a listing of background predicate definitions used by the ILP learner as well the attributes used by the regression and neural-network programs. Appendix D contains a listing of the numerical constraints extracted by the ILP program, and Appendix E lists, where practical, the corresponding output from the propositional learners.

2 Experimental Aims

From simulator records of balancing a pole on a cart, and performing a levelled turn with a simulated F-16 aircraft:

1. Using an ILP program containing general background definitions for some arithmetic and trigonometric functions, inequalities, some elementary definitions from plane geometry, and multiple linear regression, investigate if it is possible to extract equations describing the following: (a) linear and angular accelerations for the pole-and-cart system; and (b) angle of bank adopted by a pilot when performing a levelled turn on the aircraft simulator.

2. Compare the predictions made by any equations so obtained and those made by linear regression, regression-tree and neural-network methods on new data obtained under the same operating conditions.

3 Materials

3.1 Data

Data records used in this study are tabulations of the values of specified numerical variables over a number of time-instants arising from repeated runs of a pole-and-cart simulator under the control of the BOXES program [19] and the second author (R.C.) performing turn manoeuvres on a publically available flight simulator (details of both simulators are in Section 3.2). Data are from 1000 runs of each of 9 pole-and-cart configurations arising from different forces and mass ratios of pole and cart, and 29 "missions" on the flight simulator. Thus, for the two problems considered here, these tabulations are of the form shown in Figure 2. A run of the pole-and-cart simulator terminates when the simulation is unable to keep the pole upright, and a turn mission terminates when the pilot successfully completes the turn. Data are recorded at a pre-specified sampling rate. Examples for analysis of the pole-and-cart data are obtained by randomly selecting 1 record from each run. This yields 1000 randomly selected examples for each configuration of pole and cart. Each turning mission constitutes an example in the flight-simulator data. The examples are represented by a unique mission number and the average bank angle maintained during that mission. This yields 29 examples for the flight data. Independence of each run of the pole-and-cart and each turning mission is assumed.

There is no meaningful notion of negative examples for the constraints being learnt here, and each randomly selected example acts as a positive example for the ILP program.

3.2 Algorithms

The pole-and-cart simulator used here was written by Dr C.A. Sammut. Readers interested in obtaining this program should contact him directly at the Department of Computer Science and Engineering, University of New South Wales, Kensington, NSW 2033, Australia. The flight simulator used was the ACM Flight Simulator [31]. The ILP program used in all experiments implements a version of the *Progol* algorithm [22]. We use a Prolog implementation of *Progol*, called *P-Progol (Version 2.2)*. The implementation described

F	m	M	Run	t	x	$\dot{x}$	$\ddot{x}$	θ	$\dot{\theta}$	$\ddot{\theta}$
N	kg	kg		s	m	m/s	m/s^2	rad	rad/s	rad/s^2
				0.0	...	...	...	...	...	...
			1	...	...	...	...	...	...	...
				...	...	...	...	...	...	...
				...	...	...	...	...	...	...
...	...	...	:	:	:	:	:	:	:	:
			:	:	:	:	:	:	:	:
				0.0	...	...	...	...	...	...
			1000	...	...	...	...	...	...	...
				...	...	...	...	...	...	...
				...	...	...	...	...	...	...

(a) Pole-and-cart

Mission	t	x	y	θ
	s	m	m	rad
	0.0	...	...	...
1	...	...	...	...
	...	...	...	...
	...	...	...	...
:	:	:	:	:
:	:	:	:	:
	0.0	...	...	...
29	...	...	...	...
	...	...	...	...
	...	...	...	...

(b) Flight simulator

Figure 2: Form of data tabulations used in experiments. Variables tabulated refer to quantities shown in the previous figure (in flight data tabulations θ refers to the bank angle). 9 different pole-and-cart configurations are tried arising from using different values for F,m,M. Each configuration gives rise to a tabulation as shown. The 29 flight missions are characterised by pairs of (randomly selected) bearings representing initial and final heading of the simulated aircraft, and (randomly selected) bank angle and altitude of turn.

in [22] is incapable of using predicates like linear regression. The version of *P-Progol* used here implements a general-purpose extension that remedies this. Details of this are available in Appendix B. *P-Progol* is freely available for non-commercial use by ftp (ftp: ftp.comlab.ox.ac.uk, in the directory pub/Packages/ILP/pprogol2.2). For convenience, in the rest of this paper, we shall refer to *P-Progol* as *Progol*.

Experimental results also tabulate the performance of the following propositional procedures that are more usually associated with quantitative analysis: (a) a stepwise regression procedure as implemented within the SPSS software package [26]; (b) a procedure that constructs "regression-trees" as implemented within the CART family of programs [2]; and (c) a backpropogation neural-network algorithm, implemented in the Stuttgart Neural Net Simulator (SNNS) [27].

3.3 Background knowledge for ILP and attributes for propositional analysis

A complete Prolog listing of the background knowledge available to *Progol* is given in Appendix C. The contents can be summarised under the following categories:

1. **Simple arithmetic and trigonometry.** These include "built-in" definitions for $\times, /, ^{**}$ (power),sin, and cos. Of these, $+, -$, and / are unused in experiments here. The relevant predicates in Appendix C are *mult/5*, *pow/5*, *sin/4*, and *cos/4*.

2. **Inequalities.** The inequality $\leq$ is used in both experiments to bound the maximum error allowed by an equation. With equations constructed by linear regression this is usually up to 2 standard deviations from the value predicted. The relevant predicate in Appendix C is *lte/2*.

3. **Regression models.** The definition of multiple linear regression that minimises least-square error is included. The reader will note that the definitions in the appendix ensure that only those regression models that achieve a goodness-of-fit $F - test$ probability of at least 0.01, and coefficient of multiple determination (that is, r^2) of at least 0.80 are deemed acceptable. These two parameters are typically used to determine if linear models obtained are acceptable ([33]), and their values, as adopted here, avoid poorly fitting models from being explored. Non-linear models are obtained naturally by the combined use of the power function (**) and linear regression. The relevant predicates in Appendix C are *lin_regress1/6*, and *lin_regress2/8*.

4. **Geometric models.** A definition of circles is provided and used in the transformation from cartesian to circular coordinates as required for the flight simulator problem. The relevant predicates in Appendix C are *circular/2*, *estimate_radius/5*, and *circle/4*.

5. **Numerical differentiation.** The use of a 5-point formula for calculating first-order derivatives of time-varying quantities [34] is explored when analysing the pole-and-cart data. The definition is constrained to obtain values of linear and angular accelerations from the corresponding velocities. The relevant predicate in Appendix C is *deriv1/4*.

6. **Integrity constraints.** It was stated in Section 3.1 that there is no meaningful notion of negative examples for the problems considered here. It is of course possible to artificially generate such examples by slight perturbations of the data records selected (thus generating "near misses"). We have chosen instead to specify an integrity constraint that should not be violated by any theory constructed by *Progol*. This constraint expresses the straightforward requirement that clauses must contain an equation for the response variable being considered, and furthermore, the clauses must contain error-bounds on the estimates made by any equations. The relevant predicate in Appendix C is *false/0*. See Appendix B for further details on specification of integrity constraints in *Progol*.

7. **Search constraints.** A cost function to be minimised by *Progol* is specified. For the problems considered here, this specification directs *Progol* to find concepts that minimise the mean square error arising from predictions on the training set of the response variable ($\ddot{x}$, $\ddot{\theta}$, or bank angle). Also specified are predicates that define clauses in *Progol*'s hypothesis space that can be admissably dismissed from consideration. Here, such clauses are defined to be precisely those that extend clauses unnecessarily (after having obtained a constraint for the response variable), or those clauses that could not possibly be extended further to include a constraint for the

response variable (due to constraints on the hypothesis language). The relevant predicates in Appendix C are *cost/3*, and *abandon/0*. See Appendix B for further details on specification of costs and search constraints in *Progol*.

For a fair comparison with propositional procedures, it is necessary that these procedures are also provided with attributes that encode the same background information. This listings of these attributes in Appendix C are self-explanatory and are not elaborated any further.

4 The pole-and-cart

Appendix A contains a brief description of the simulator equations that act as target descriptions for experiments here. In these experiments, equations for linear and angular acceleration are constructed separately. When obtaining constraints for $\ddot{x}$, *Progol* has access to tabulated values of $x, \dot{x}, \theta, \dot{\theta}$, and $\ddot{\theta}$ for the examples chosen. Similarly, when obtaining constraints for $\ddot{\theta}$ the ILP program has access to values of $x, \dot{x}, \theta, \dot{\theta}$, and $\ddot{x}$.

4.1 Method

The following method was adopted to generate data, obtain and test any constraints:

1. 1000 runs are performed with each of the following values of force (F in N), mass of pole (m in kg), and mass of cart (M in kg): (i) $F = 0.0, m = 0.1, M = 1.0$; (ii) $F = 0.0, m = 1.0, M = 1.0$; (iii) $F = 0.0, m = 1.0, M = 0.1$; (iv) $F = +10.0, m = 0.1, M = 1.0$; (v) $F = +10.0, m = 1.0, M = 1.0$; (vi) $F = +10.0, m = 1.0, M = 0.1$; (vii) $F = \pm 10.0, m = 0.1, M = 1.0$; (viii) $F = \pm 10.0, m = 1.0, M = 1.0$; and (ix) $F = \pm 10.0, m = 1.0, M = 0.1$. On each run the values of $x, \dot{x}, \theta$, and $\dot{\theta}, \ddot{x}, \ddot{\theta}$ are tabulated once every $0.02s$.

2. For each configuration of F, m, M, a data record is selected at random. This yields 1000 data records for each configuration. The first 500 of these are used for "training" and the remainder are set aside to "test" any constraints obtained.

3. Using $x, \dot{x}, \theta, \dot{\theta}$, and in turn with $\ddot{\theta}$, $\ddot{x}$ values from the training data and background knowledge definitions for: $\times, ^{**}, sin, cos, \leq$, multiple regression (up to 2 independent variables); obtain equations for $\ddot{x}$ and $\ddot{\theta}$ with *Progol*. Record $\ddot{x}$ and $\ddot{\theta}$ values computed by these equations on the test set.

4. Using training-set values for the attributes listed in Appendix C, for each of $\ddot{x}$ and $\ddot{\theta}$: (a) obtain a linear equation using stepwise linear regression; (b) obtain a regression tree; and (c) train the neural net. Record $\ddot{x}$ and $\ddot{\theta}$ values computed by each of these methods on the test set.

5. Compute the degree of agreement of each method to the actual values of $\ddot{x}$ and $\ddot{\theta}$ by computing the mean of the squared difference between actual values and those predicted by each method. In keeping with [30], this statistic is termed the "mean square error of prediction" (MSEP).

We first draw attention to the fact that the ILP program uses linear models with at most 2 independent variables. This choice is directed by the target model in Appendix A.

Parameter settings for the stepwise regression procedure control the inclusion and exclusion of variables in the equation. These parameters relate to the level of significance that has to be achieved above (or below) which a variable is retained in the model (or removed from the model). There is no prescribed setting for these parameters (termed PIN and $POUT$ respectively in SPSS). We have adopted the procedure of setting $PIN, POUT$ to values that result in equations with no more than 2 independent variables on the training data, except when predicting $\ddot{x}$ with $F = \pm 10N$. In this case, up to 3 independent variables are allowed, to enable the program to use values of F in the equation. This is in accordance with the restrictions provided to the ILP program.

The regression-tree procedure implemented within CART is equipped with validation procedures that enable it to estimate automatically the parameters specifying the tree-construction.

For the neural-network algorithm in SNNS, 10000 epochs were used with the "shuffle" option. In each epoch all training examples were presented. The net has one input unit for each attribute, 4 hidden units and 1 output unit and was fully connected. There is no accepted method for determining the number of hidden units. The settings chosen were obtained based on the fact that they yielded the lowest error on the training data across possible settings ranging from 0 to 5 units in the hidden layer.

4.2 Experimental results

Figures 3 and 4 tabulate values of MSEP summarising the difference between actual values of $\ddot{x}, \ddot{\theta}$ and those predicted by each method. The actual equations on which these calculations are based are in Appendix D.

5 Flight simulator

All experiments are concerned with executing a levelled left turn. A turn is specified by a tuple of numbers representing the initial and final bearing of the simulated aircraft, the bank angle and altitude to be maintained, along with constraints for "safe" flying (see below).

5.1 Method

The following method was adopted to generate data, obtain and test any constraints:

Algorithm	$F = 0$			$F = +10$			$F = \pm 10$		
	m/M			m/M			m/M		
	0.1	1.0	10.0	0.1	1.0	10.0	0.1	1.0	10.0
Progol	$9e-6$	$9e-6$	$2e-5$	$6e-5$	$1e-5$	$6e-4$	$4e-5$	$4e-4$	$2e-2$
Regression	$9e-6$	$9e-6$	$2e-5$	$6e-5$	$1e-5$	$6e-4$	$6e-5$	$1e-5$	$1e-2$
Regression tree	$3e-5$	$4e-4$	$3e-3$	$3e-5$	$7e-4$	$2e-2$	$5e-5$	$1e-3$	$3e-2$
Neural network	$2e-5$	$6e-4$	$5e-3$	$3e-5$	$1e-3$	$3e-2$	$8e-4$	$1e-2$	$1e-1$

Figure 3: Mean square error of predicted values of $\ddot{x}$ on 500 test examples. The notation $ae-b$ stands for $a \times 10^{-b}$.

Algorithm	$F = 0$			$F = +10$			$F = \pm 10$		
	m/M			m/M			m/M		
	0.1	1.0	10.0	0.1	1.0	10.0	0.1	1.0	10.0
Progol	$2e-3$	$1e-3$	$2e-3$	$2e-3$	$2e-3$	$1e-3$	$2e-3$	$2e-3$	$2e-3$
Regression	$2e-3$	$4e-4$	$8e-4$	$2e-3$	$2e-3$	$5e-2$	$1e-2$	$8e-3$	$7e-2$
Regression tree	$3e-3$	$5e-3$	$2e-2$	$3e-3$	$1e-2$	$4e-2$	$5e-3$	$1e-2$	$1e-1$
Neural network	$6e-3$	$1e-2$	$3e-2$	$5e-3$	$9e-3$	$1e-1$	$1e-1$	$1e-1$	$4e-1$

Figure 4: Mean square error of predicted values of $\ddot{\theta}$ on 500 test examples. The notation $ae-b$ stands for $a \times 10^{-b}$.

1. 90 turns were specified by random selection of initial and final bearings (from the interval $[0°, 360°]$), bank-angle (from the interval $[-20°, -5°]$) and altitude (from the interval $[2000ft, 3000ft]$). Additionally for safe flying, in all missions and at all times the altitude deviation from the goal altitude value required to be within $100ft$, climb rate was kept in the interval $[-1000ft/min, 1000ft/min]$, and the bank angle deviation from the goal bank angle value was required to be within 3°. 29 of the 90 turns executed by a pilot on the simulator complied with these constraints. On each turn records of X, Y coordinates and the bank angle θ are tabulated once every $0.1s$. A randomly sampled subset of 100 records is used to represent this tabulation.

2. Each mission is represented by a data record containing a unique mission identifier and average bank angle maintained. This yields 29 data records.

3. The following k-fold cross-validation design is adopted:

(a) Randomly assign the 29 records to k (approximately) equal partitions. Each partition will, in turn, be withheld to form a "test" set. The records in the other partitions will provide the "training" data for constructing the equation for predicting the bank angle of a mission.

(b) With each of the k training data sets:

   i. Using X, Y values and background knowledge definitions for: $**, \leq$, a predicate estimating the radii of circular trajectories, multiple regression (up to 2 independent variables); obtain equations for θ with *Progol*. Record θ values computed by these equations on the test data set.

   ii. Using training-set values for the attributes listed in Appendix C, for θ: (a) obtain a linear equation using stepwise linear regression; (b) obtain a regression tree; and (c) train the neural net. Record θ values computed by each of these methods on the test set.

(c) Compute the degree of agreement of each method to the actual values of θ by computing the mean of the squared difference between actual values of θ and those predicted by each method for k test sets.

For the experiments here, k is set to 29 (that is, a leave-one-out procedure is adopted).

It is not known beforehand how closely the model in Appendix A approximates the turning problem addressed here. Consequently the restriction to linear models of at most 2 independent variables for the ILP program is based on the expectation that other terms become vanishingly small for turns of even moderate radii (such as those obtained from the data here).

Parameter settings for the stepwise regression procedure, regression tree, and the neural-network are obtained as in Section 4.1. Given the repeated re-use of the sample within the cross-validation procedure may introduce some biases in the mean-square error value obtained from the k test sets. This, along with the relatively few data points, suggests a cautious interpretation of the values obtained.

5.2 Experimental results

Figure 5 tabulates values of the MSEP of the relationship between actual values of θ and those predicted by each method on data in the cross-validation test sets.

6 Discussion

The tabulations in Sections 4.2 and 5.2 show that for the pole-and-cart data the mean-square-error of prediction (MSEP) from the ILP theory usually lower than all programs other than regression, and is lower than all propositional learners for the flight-simulator data. Figure 6 tabulates a comparison of the

Algorithm	Mean square error of prediction
Progol	1.355
Regression	1.456
Regression tree	2.334
Neural network	1.393

Figure 5: Mean square error resulting from the predictions of values for θ in the flight simulator data. Predictions are on the the test examples left out in turn during cross-validation.

MSEP of *Progol* against that of the propositional learners for the 18 errors tabulated in Figures 3 and 4.

Algorithm	MSEP of *Progol*		
	Better	Worse	Same
Regression	5	4	9
Regression tree	16	1	1
Neural network	17	1	0

Figure 6: A comparison the the MSEPs on the pole-and-cart data. The terms "better", "worse" and "same" denote the cases that the MSEP of Progol is lower, higher, or the same (up to the precision shown in Figures 3 and 4) as the corresponding propositional learner.

In general, we would expect the predictivity of *Progol*'s theories to be at least comparable to those of linear regression given that the ILP program relies on regression definitions provided as background knowledge to construct its equations. Further, since by virtue of its search technique, *Progol* would do an "all-subsets" regression, it would seem to be surprising to find instances where the SPSS regression procedure (which implements "stepwise" regression) actually does perform better. Closer examination reveals that in 3 of the 4 cases that this occurs in Figure 6, *Progol* has identified the correct target equation. This suggests its higher error to be an artifact of the test sample. To this extent, the entries in the first row of Figure 6 are as expected. This is also reflected in that similarity in MSEP values between the two programs on the flight-simulator data.

On both data sets, the regression tree's predictions appear to be considerably worse than those made by *Progol*. By producing axis-parallel partitions and predicting mean values for each partition, the tree program is unable to capture the linear nature of the constraints involved. Definitive statements concerning the apparent poorer performance of the neural network are confounded by the absence of a principled technique for selecting the topology of the network. We can therefore do no more than state that for the particular topology used, the neural network's predictions are consistently worse than *Progol*'s on the pole-and-cart data, and comparable on the flight-simulator data.

Besides a comparison of predictive accuracies, it is instructive to examine, where possible, the nature of the theories obtained by each algorithm. Appendix D shows that for the pole-and-cart problem, *Progol* constructs 24 equations corresponding to different physical parameter settings. Of these, the reader can verify that the correct linear model is identified on 22 occasions. Here, by "correct" we mean that the linear model has the same predictor variables as those in the target model described in Appendix A – we will examine the issue of correctly estimating the coefficients for these variables in greater detail below. In contrast, linear regression constructs 18 equations, of which 9 are correct. The difference in the number of equations highlights an important point of difference between the two programs when constructing theories for the case where $F = \pm 10N$. Here, *Progol* constructs a pair of equations corresponding to the situations arising from $F = +10N$, $F = -10N$. The regression program attempts to explain both situations by using F as an independent variable in equations for $\ddot{x}$.

Given the nature of their theories, we are not in a position to directly compare the output of the regression tree and neural network against the target model. On the pole-and-cart data, the tree program produces reasonably complex theories (see Figure 7), and the latter's "theory" consisting of a listing of 25 floating-point numbers corresponding to the weights associated with each node in the network.

F	m/M when predicting $\ddot{x}$			m/M when predicting $\ddot{\theta}$		
	0.1	1.0	10.0	0.1	1.0	10.0
0	63	161	173	81	143	233
+10	97	157	91	87	135	135
±10	147	151	131	135	207	123

Figure 7: Number of nodes in the regression tree for each pole-and-cart configuration considered.

The absence of an accepted target model for the flight simulator data prevents any in-depth assessment of the theories obtained. It is however worth highlighting the natural use by the ILP program of a clausal definition that estimates the radii of circular trajectories by sampling from the simulator traces (the predicate *circular/2* in Appendix C). In some sense, this approach of "guessing" planar figures in the data is similar in spirit to method of "qualitative regression" described in [24]. In that approach, several circles (and their radii) are conjectured and the ILP program searches for the one that best fits the data. Here we estimate a single circular arc by averaging over several circles obtained by sampling points on the turning trajectory. The resulting theory from *Progol* also demonstrates the use of a predicate calculating powers (**) to achieve curvilinear regression within ILP. For this data the regression tree obtained is simpler (consisting of 7 nodes), and is reproduced in Appendix D. The neural network theory consisting of 11 floating point numbers remains opaque to further inspection.

Unlike the flight-simulator, with the pole-and-cart simulator the availability of an automated controller (here the BOXES program) and a known target theory allows us to investigate further the nature of theories obtainable from *Progol*. In what follows, we restrict attention to a commonly used pole-and-cart configuration, namely $F = \pm 10N, m = 0.1kg, M = 1.0kg$. Within this setting, we are in a position to examine empirically: (a) the convergence of coefficients in *Progol*'s equations to the coefficients in the target theory; (b) bias in prediction and the estimation of the coefficients by *Progol*; and (c) extensions to the background knowledge to allow numerical differentiation by *Progol*.

We first examine how *Progol*'s estimation of the target theory changes with increasing the number of training examples. Target theories for $\ddot{x}$ and $\ddot{\theta}$ are obtained from the equations of motion given in Appendix A. These are of the form $\ddot{x} = C_1 - M_1\ddot{\theta}cos\theta + M_2\dot{\theta}^2 sin\theta$ and $\ddot{\theta} = C_2 + K_1 sin\theta - K_2\ddot{x}cos\theta$. For the given set of physical parameters, namely $F = \pm 10N, m = 0.1kg, M = 1.0kg$, the coefficients take the particular values: $C_1 = \pm 9.091, M_1 = M_2 = 0.045, C_2 = 0.000, K_1 = 14.7, K_2 = 1.50$ (the two values of C_1 result from $F = \pm 10N$). Figure 8 tabulates the progressive change in error of the *Progol* estimates from these expected values, averaged over the two values of F. The tabulation suggests that *Progol*'s approximation of the target theory does improve as the number of training examples is increased.

Training Examples	Error in coefficients for $\ddot{x}$	Error in coefficients for $\ddot{\theta}$
100	–	0.803
250	0.002	0.587
500	0.002	0.876
1000	0.002	0.350
1500	0.001	0.141

Figure 8: Error of coefficients estimated in the linear model obtained by Progol for the pole-and-cart data. For a given number of training examples, this error is the average of the total absolute deviation of estimates from expected values (from the target model) for $F = +10$ and $F = -10$. Estimates and expected values are recorded up to 3 significant figures. An entry of "–" denotes that the target model was not identified by Progol.

We now turn our attention to the question of bias in *Progol*'s predictions and estimates of the coefficients. For the physical settings under consideration,

Figure 9 tabulates the frequencies of positive and negative residuals arising from over and under-estimates of $\ddot{x}$ and $\ddot{\theta}$ on the 500 test examples. Also tabulated are the means of the predicted and actual values for each variable. The entries suggest that on *Progol*'s predictions appear to be largely unbiased.

Dependent	Frequency		Mean	
variable	+ residuals	− residuals	Actual	Predicted
$\ddot{x}$	0.46	0.52	−0.195	−0.196
$\ddot{\theta}$	0.54	0.46	0.309	0.310

Figure 9: Examining residuals for bias in Progol's prediction of the dependent variable on the pole-and-cart data.

Consider now any bias in the estimation of coefficients that appear in equations for the dependent variables. Operating under the control of a two-sided force provides us with two estimates for each coefficient – one for which F is positive (here $+10N$) and the other when it is negative ($-10N$). Calling these two estimates "duplicates" [1], and repeatedly performing the experiment of learning equations for $\ddot{x}$ and $\ddot{\theta}$ allows us to record the number of occasions that (a) both duplicates overestimate a coefficient; (b) both duplicates underestimate a given coefficient; (c) the F-positive duplicate overestimates a coefficient while the F-negative underestimates; (d) the F-positive duplicate underestimates a coefficient while the F-negative overestimates; (e) one duplicate under or over-estimates a coefficient while the other is exact (up to some degree of precision); and (f) both duplicates estimate a coefficient exactly (again, up to some degree of precision). Severe discrepancies between the tallies in (a) and (b) would suggest that *Progol*'s estimation of that coefficient was biased. Discrepancies in the tallies (c) and (d) would suggest bias in the simulator. Tallies of (e) and (f) are not of particular interest here.

Figure 10 tabulates these numbers over 10 repeats of obtaining equations for $\ddot{x}$ and $\ddot{\theta}$ from independent training sets of 500 examples each. The 6 cases (a)–(f) in the tabulation correspond to the situations described above, and the coefficients $C_1 \ldots K_2$ are as before. The comparison of estimates and expected values proceeds to 3 significant figures. It is evident from this figure that for 4 of the 6 coefficients, there is no evidence of any bias. Of the remaining two, the evidence for bias in estimates of M_1 does not appear to be substantial. Some degree of uncertainty does however rest on the estimates of M_2, and suggests further investigation.

The discussion in this section has largely centred on examining the theories constructed by *Progol* for the relatively well-defined world offered by the pole-and-cart problem. In the experiments undertaken, the reader would have noticed that the target equations for $\ddot{x}$ require the values of $\ddot{\theta}$ and vice-versa and that *Progol* obtained these values directly from the simulator. It is of interest

[1] This term and the analysis following are due to Donald Michie

Case	Predicting $\ddot{x}$			Predicting $\ddot{\theta}$		
	C_1	M_1	M_2	C_2	K_1	K_2
(a)	0	0	5	1	0	3
(b)	0	3	0	1	0	3
(c)	0	0	2	5	0	1
(d)	0	0	0	3	0	1
(e)	0	5	3	0	2	2
(f)	10	2	0	0	8	0

Figure 10: Occurrences of cases arising from duplicate estimates over, under or exactly estimating a given coefficient in the linear models describing the pole-and-cart data. The cases (a) – (f) are as described in the text. As before, estimates and expected values are recorded up to 3 significant figures.

to examine whether the ILP program could achieve comparable results by calculating the derivatives required by using background knowledge definitions for numerical differentiation. This would allow the ILP program to emulate other, more special purpose programs like LAGRANGE [8]. We close this discussion with a demonstration that such behaviour is possible by extracting constraints for $\ddot{x}$ and $\ddot{\theta}$ using $x, \dot{x}, \theta$, and $\dot{\theta}$ in conjunction with a well-known 5-point formula for numerical differentiation. The physical parameters remain as before, namely $F = \pm 10N, m = 0.1kg, M = 1.0kg$, and the corresponding mean-square-error on the test sample is in Figure 11. The errors can be seen to be comparable to the corresponding ones obtained earlier in Figures 3 and 4. However, it is important to note here that the 5-point formula for numerical derivatives are only calculable for 266 of the 500 randomly chosen test examples with all other examples being drawn too close to the edges of a run. Prediction of values is thus possible for only a subset of the data – an issue that is not peculiar to the use of *Progol*, but one that arises from the particular numerical method employed to obtain derivatives.

7 Conclusion

By adopting logic programs as its basic representation formalism, an ILP program can use and construct clauses in a Turing-equivalent language. In principle therefore, there is no restriction to the functions that can be used as background knowledge or learnt by such a program. This paper has sought to demonstrate empirically that an exploitation of the power afforded by this framework could go some way towards redressing a perceived limitation of existing ILP programs, namely, the analysis of large quantities of numerical data. This problem has been addressed variously in the literature by either restricting the language of hypotheses to logical atoms [13], using built-in definitions for

Variable	Mean square error
$\ddot{x}$	$3e-5$
$\ddot{\theta}$	$2e-3$

Figure 11: Mean square error of Progol predictions for $\ddot{x}$ and $\ddot{\theta}$ on test examples with programs for numerical differentiation provided as background knowledge. The notation $ae-b$ stands for $a \times 10^{-b}$. Note that the errors reported are calculated only on those examples in the test-set for which numerical derivatives are defined.

inequalities [3, 28, 29] or regression and numerical derivatives [14, 15], transformations to propositional level [18] or constraint satisfaction problems [20, 32], or using background knowledge for qualitative regression-like predicates [24]. In contrast to these methods, experiments reported in this paper have relied on the use of non-ground background predicates for numerical reasoning, and 3 implementation features within the ILP program, namely (1) the lazy evaluation of functional outputs; (2) the use of integrity constraints to avoid over-generalisation; and (3) the use of costs and search constraints. Equipped with these, results from the analysis of data from simulators for two control tasks – balancing a pole on a cart and a pilot turning an aircraft – show theories constructed by the ILP program *Progol* to compare favourably against those from techniques usually considered for the analysis of quantitative data. To this extent, this paper contributes to the literature cited above by providing a controlled investigation of what is achievable with an existing general-purpose ILP program.

Acknowledgements

This research was supported by the EPSRC project 'Experimental Application and Development of ILP'. At the time of conducting these experiments, Rui Camacho was supervised initially by Stephen Muggleton, and later by Pavel Brazdil. Support for Rui Camacho was provided by Universidade do Porto and JNICT (Junta Nacional de Investigação Científica e Tecnológica). Thanks are due to Donald Michie for his invaluable guidance and advice and to Stephen Muggleton and Claude Sammut for patient discussions on the experiments here.

References

[1] I. Bratko and M.Grobelnik. Inductive learning applied to program construction and verification. In *Third International Workshop on Inductive Logic Programming*, pages 279–292, 1993. Available as Technical Report IJS-DP-6707, J. Stefan Inst., Ljubljana, Slovenia.

[2] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, 1984.

[3] R. Camacho. Learning stage transition rules with Indlog. In S. Wrobel, editor, *Proceedings of the Fourth International Inductive Logic Programming W orkshop.* Gesellschaft fur Mathematik und Datenverarbeitung MBH, 1994. GMD-Studien Nr 237.

[4] R. H. Cannon. *Dynamics of physical systems.* McGraw-Hill, New York, 1967.

[5] European Commission. ESPRIT Long Term Research Project 20237 – ILP II. Technical Annex I, European Commission, Directorate General, Industry, Brussels, 1996.

[6] L. DeRaedt and M. Bruynooghe. Interactive concept-learning and constructive induction by analogy. *Machine Learning*, 8(2):107–150, 1992.

[7] B. Dolsak and S. Muggleton. The application of Inductive Logic Programming to finite element mesh design. In S. Muggleton, editor, *Inductive Logic Programming*, pages 453–472. Academic Press, London, 1992.

[8] S. Dzeroski. *Handling Noise in Inductive Logic Programming.* University of Ljubljana, (M.Sc. Thesis), Ljubljana, 1991.

[9] S. Dzeroski, L. Dehaspe, B. Ruck, and W. Walley. Classification of river water quality data using machine learning. In *Proceedings of the Fifth International Conference on the Development and Application of Computer Techniques Environmental Studies*, 1994.

[10] C. Feng. Inducing temporal fault dignostic rules from a qualitative model. In S. Muggleton, editor, *Inductive Logic Programming*, pages 473–486. Academic Press, London, 1992.

[11] I.S. Gradshtein and I.M. Ryzhik. *Table of integrals, series, and products.* Academic Press, Boston, 1994. 5th Edition.

[12] D. Halliday and R. Resnick. *Physics.* Wiley, New York, 1986. 3rd Edition.

[13] C.D. Page Jr. and A.M. Frisch. Generalization and Learnability: A Study of Constrained Atoms. In S. Muggleton, editor, *Inductive Logic Programming*, pages 29–56. Academic Press, London, 1992.

[14] A. Karalic. Relational regression: first steps. Technical report ijs-dp-7001, J. Stefan Institute, Ljubljana, Yugoslavia, 1994.

[15] A. Karalic and I. Bratko. First-Order Regression. *Machine Learning Journal*, 1996. Special Issue on ILP, to appear.

[16] R.D. King, S.H. Muggleton, A. Srinivasan, and M.J.E. Sternberg. Structure-activity relationships derived by machine learning: The use of atoms and their bond connectivities to predict mutagenicity by inductive

logic programming. *Proc. of the National Academy of Sciences*, 93:438–442, 1996.

[17] R.D. King, S.H. Muggleton, and M.J.E. Sternberg. Drug design by machine learning: The use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proc. of the National Academy of Sciences*, 89(23):11322–11326, 1992.

[18] N. Lavrac and S. Dzeroski. *ILP: Techniques and Applications.* Ellis Horwood, London, 1994.

[19] D. Michie and R.A. Chambers. BOXES: An experiment in adaptive control. In E. Dale and D. Michie, editors, *Machine Intelligence 2*, pages 137–152. Oliver and Boyd, Edinburgh, 1968.

[20] F. Mizoguchi and H. Ohwada. An Inductive Logic Programming approach to constraint acquisition for constraint-based problem solving. In L. DeRaedt, editor, *Fifth International Inductive Logic Programming Workshop.* Katholieke Universiteit Leuven, Heverlee, Belgium, 1995.

[21] S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

[22] S. Muggleton. Inverse Entailment and Progol. *New Gen. Comput.*, 13:245–286, 1995.

[23] S. Muggleton, R. King, and M. Sternberg. Predicting protein secondary structure using inductive logic programming. *Protein Engineering*, 5:647–657, 1992.

[24] S. Muggleton and D. Page. Beyond first-order learning: inductive learning with higher-order logic. PRG-TR 13-94, Oxford University Computing Laboratory, Oxford, 1994.

[25] S.H. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, Tokyo, 1990. Ohmsha.

[26] M.J. Norusis. *SPSS: Base System User Guide. Release 6.0.* SPSS Inc., 444 N Michigan Ave, Chicago, Illinois 60611, 1994.

[27] University of Stuttgart. Snns user manual, version 4.1. Technical Report 6/95, Institute for Parallel and Distributed High Performance Systems, Stuttgart, 1995.

[28] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[29] J.R. Quinlan and R.M. Cameron-Jones. Efficient top-down induction of logic programs. *SIGART Bulletin*, 5(1):33–42, 1994.

[30] J.O. Rawlings. *Applied regression analysis : a research tool.* Wadsworth and Brooks, Pacific Grove, California, 1988.

[31] R.Rainey. The acm simulator, version 2.4. Public domain software, Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA, 1991.

[32] M. Sebag and C. Rouveirol. Constraint Inductive Logic Programming. In L. DeRaedt, editor, *Fifth International Inductive Logic Programming Workshop.* Katholieke Universiteit Leuven, Heverlee, Belgium, 1995.

[33] R.E. Walpole and R.H. Myers. *Probability and Statistics for Engineers and Scientists.* Collier Macmillan, New York, 1978. 2nd Edition.

[34] D.M. Young and R.T. Gregory. *A Survey of Numerical Mathematics, Volume I.* Addison-Wesley, Reading, MA, 1972.

[35] J. Zelle and R. Mooney. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 817–822. Morgan Kaufmann, 1993.

A Constraints imposed by the physical systems involved

A.1 Pole-and-cart

Exact equations of motion for the pole-and-cart system are available in the literature, and we do not include their derivation here. For this, the reader is referred to the treatment in [4], pp 703-710. Instead, we merely reproduce the relevant equations here. The state of the system is given by θ: the angle of the pole from the upright position; $\dot{\theta}$: angular velocity of the pole; x: horizontal position of the cart's center; and $\dot{x}$: the velocity of the cart. Given a force on the cart of fixed magnitude F, and a pole of length l, cart and pole masses of M and m, the equations of motion are:

$$(M+m)\ddot{x} + \frac{1}{2}ml(\ddot{\theta}cos\theta - \dot{\theta}^2 sin\theta) = F$$

$$\ddot{x}cos\theta + \frac{2}{3}l\ddot{\theta} = gsin\theta$$

These can be re-arranged to give constraints for $\ddot{x}$ and $\ddot{\theta}$:

$$\ddot{x} = \frac{F - \frac{1}{2}ml(\ddot{\theta}cos\theta - \dot{\theta}^2 sin\theta)}{M+m}$$

$$\ddot{\theta} = \frac{gsin\theta - \ddot{x}cos\theta}{\frac{2}{3}l}$$

A.2 Levelled turn of a simulated aircraft

We have not been able to identify sources in the literature that provide the exact equation for the angle of bank for a levelled turn – descriptions we have encountered in flight instruction manuals are usually of a qualitative nature. In the absence of an exact model, we assume the physics to be analogous to a car performing a turn on a banked road. A solution to the idealised scenario that ignores frictional aspects can be found in a textbook on physics like [12]. For a car of mass m moving with constant linear velocity v, performing a turn of radius r on a road with angle α, by balancing components of gravity and the centrifugal force it is straightforward to show that:

$$mgsin\alpha = \frac{mv^2cos\alpha}{r}$$

where g is the gravitational acceleration constant. This simplifies to:

$$\alpha = tan^{-1}\frac{v^2}{rg}$$

Two series expansions for $tan^{-1}x$ are available in [11]. The first is:

$$tan^{-1}x = x - \frac{x^3}{3} + \frac{x^5}{5} \ldots -1 \leq x \leq 1$$

and the second:

$$tan^{-1}x = \frac{\pi}{2} - \frac{1}{x} - \frac{1}{x^3} \ldots x \geq 1$$

For the data collected here, $x = \frac{v^2}{rg} \leq 1$. Further the convention for a levelled left turn makes the bank angle $\theta = -\alpha$. This gives the following constraint for the value of θ:

$$\theta = -\frac{k}{r} + \frac{k^3}{3r^3} - \frac{k^5}{5r^5} \ldots$$

where $k = v^2/g$. It is difficult to determine how closely the aircraft simulation matches this model. We are able to obtain one boundary condition that states that flying in a straight line is equivalent to performing a turn of infinite radius with a bank angle of 0. This condition is satisfied by the constraint above. Guided by this, we will examine regression equations for θ that are concerned with negative powers of r only.

B Some relevant features of the Progol implementation

The algorithm specified in [22] proceeds as follows. Given background knowledge B, positive and negative examples E^+ and E^-: (1) randomly select a positive example e_i; (2) use inverse resolution to construct the most specific hypothesis $\perp(B, e_i)$ which explains e_i in terms of B; (3) starting from the empty clause $\Box$, add (variants of) literals from $\perp(B, e_i)$ to find a rule D_i which maximally compresses a set of entailed examples E_i; and (4) add D_i to the hypothesis H and repeat from (1) with positive examples not covered so far until no more compression is possible.

The implementation of *Progol* used in this paper contains the following extensions to this procedure.

B.1 Lazy evaluation

Using a most-specific clause from a single example to guide the search for clauses prevents *Progol* from using predicates like linear regression where the functional result (here the coefficients of regression) is determined from a set of values that arise from more than one example. To address this, the *Progol* implementation used here is capable of "lazily evaluating" functional outputs. For a particular function f, this requires that the background knowledge B has the following: (1) a "mode" annotation specifying input and output argument(s) for f along with their types; and (2) a definition specifying the procedure for evaluating the output argument(s) from sets of values for the input argument(s). Provided with this, the following modifications to the basic algorithm are implemented. First, when constructing the most-specific clause in Step (2), negative literals in $\perp(B, e_i)$ representing functions designated for lazily evaluation have skolemised values for output arguments. Second, in Step (3) if the literal selected for addition requires lazy evaluation then its output values are computed before addition.

Lazy computation of output values is accomplished as follows. Without loss of generality, assume that the predicate $f/2$ has been marked for lazy evaluation, and the mode annotations state the first argument to be "input" and the second as "output". This literal now appears as $f(X, SkolemY)$ in $\perp(B, e_i)$. Step (3) is concerned with adding this literal to a definite clause $C : head \leftarrow body$. Let $\theta_p = \{X/p_1, X/p_2, \ldots\}$ be the substitution instances for X from each substitution θ_{p_i} such that $head \cdot \theta_{p_i} \in E^+$ and $body \cdot \theta_{p_i}$ evaluates to *true*. Analogously, let $\theta_n = \{X/n_1, X/n_2, \ldots\}$ be substitution instances of X from each substitution θ_{n_i} such that $head \cdot \theta_{n_i} \in E^-$ and $body \cdot \theta_{n_i}$ evaluates to *true*. Lazy evaluation of the output value of $f/2$ is then accomplished by collecting the answer substitution Y/y_i from the call $f([\theta_p, \theta_n], Y)$. *Progol* then adds the literal $f(X, y_i)$ to the clause C and the search proceeds normally. Of course, this assumes that the background knowledge B is complete to the extent that a value for Y can be computed from the call $f([\theta_p, \theta_n], Y)$. Should this not be the case, the literal is discarded and search proceeds by selecting another literal from $\perp(B, e_i)$.

B.2 User-specified constraints

Progol accepts constraints that should not be violated by the hypothesised clauses. These are written in the background knowledge file (that has extension

".b") and are a restricted form of the type of integrity constraints found ILP programs like Clint [6]. The constraints are distinguished by the fact that they are all rules of the form:

```
false:-
          Body.
```

where Body is a set of literals that specify the condition(s) that should not be violated by a clause found by P-Progol. It is usual to specify constraints on the clause currently being considered by *Progol*. This clause is available by calling the distinguished predicate *hypothesis/3*. For example, the following fragment of background knowledge for the pole-and-cart problem states that acceptable clause must contain an equation for linear acceleration and an inequality literal that bounds the error on the value predicted.

```
false:-
        hypothesis(accel(_,_,_,_,_,_,_,Tdd,Xdd),Body,_),
        not(calc_and_bounds_err(Xdd,Body)), !.

calc_and_bounds_err(Variable,(lin_regress1(Y,_,_,_,_,E),Lits)):-
        Variable == Y, !,
        bounds_error(E,Lits).
...

calc_and_bounds_err(Variable,(_,Lits)):-
        calc_and_bounds_err(Variable,Lits).
```

The use of constraints allows *Progol* to obtain interesting hypothesis without negative examples. Ordinarily, this will result in a single clause that classifies every example as positive. Such clauses can be precluded by constraints. It is also possible to reformulate as constraints *Progol*'s restrictions on maximum literals allowed in a clause, and consistency requirements such the maximum number of negative examples to be covered by a clause. Finally, it should be noted that a constraint does not state that a refinement of a clause that violates one or more constraints will also be unacceptable.

B.3 User-specified costs and pruning

The use of a user-specified cost function is a fundamental construct in statistical decision theory, and provides a general method of scoring descriptions. *Progol* allows the specification of the cost of a clause using a clausal definition for the predicate *cost/3*. This definition is of the form:

```
cost(Clause,ClauseLabel,Cost):-
         ...
```

where ClauseLabel is a list containing summary statistics of the clause including the number of positive examples covered by the clause (P), the number of negative examples covered by the clause (N), and the number of literals in the clause (L). The compression measure defined in [22] can thus be seen as assigning a cost of $P - N - L$ to a clause. It is conceivable of course that in general, the calculation of `Cost` may not require any of these statistics.

It is usually not possible to devise automatically admissable pruning strategies for an arbitrary cost function. Thus, when using a user-defined cost measure, *Progol* places the burden of specifying a pruning strategy on the user. This specification takes the form of clausal definitions for the predicate *abandon/0*. Each such definition specifies conditions under which a user knows for certain that a clause (or its refinements obtained by adding literals) could not possibly be an acceptable hypothesis. The "abandon" definitions are written in the background knowledge file, and are distinguished by the fact that they are all rules of the form:

```
abandon:-
        Body.
```

As in the definition of constraints, it is usual to use the *hypothesis/3* predicate to first obtain the clause currently being considered by *Progol*. For example, the following fragment of background knowledge for the pole-and-cart problem states that for clauses that already have a literal computing linear acceleration we are only interested in extensions that bound the error. All others can be legitimately pruned away:

```
abandon:-
        hypothesis(accel(_,_,_,_,_,_,_,Tdd,Xdd),Body,_),
        calculated(Xdd,Body,Error,NextLit),  % there is an equation for Xdd
        NextLit \= true,                     % there is a literal after the equation
        not(bounds(Error,NextLit)), !. % which does not bound the error


% variable is computed by a regression equation in the body with error E

calculated(Variable,(lin_regress1(Y,_,_,_,_,E),Lits),E,Lits):-
        Variable == Y, !.
...

calculated(Variable,(_,Lits),E,Left):-
        calculated(Variable,Lits,E,Left), !.
```

It should be noted here that both the pruning and constraint specifications of the previous section may encode more than the simple forms of syntactic checks described here. It is conceivable, for example, that semantic considerations relating to coverage of clauses could be relevant in pruning the hypothesis space.

Progol is instructed to utilise the user-defined cost function and associated pruning by issuing the command *set(search,user)*. Once this is done, *Progol* then proceeds as normal and attempts to find a clause that is consistent with the constraints specified. Should there be more than one such consistent clause, then the one with minimum cost is returned.

C Information provided to ILP and propositional programs

C.1 Background knowledge for Progol

Background knowledge for *Progol* consists of the following: (1) Language restrictions which include mode definitions, depth of variable chaining, depth of theorem proving, clause length restrictions, and number of nodes to be explored in the search; (2) Specification of properties of predicates, including commutativity and lazy evaluation; (3) Type definitions; (4) Definitions of background predicates; and (4) Definitions of constraints, costs and pruning. We describe each of these below, noting any details that may be peculiar to the two problems considered in this paper. Unless stated otherwise, the syntax of these statements is as in [22].

Language restrictions

The following restrictions are peculiar to the pole-and-cart.

```
:- modeh(1,accel(+sequence,+time,+x,+xdot,+theta,+thetadot,#force,+angaccel,+linaccel)).

:- modeb(1,cos(+sequence,+time,+theta,-trignum)).
:- modeb(1,sin(+sequence,+time,+theta,-trignum)).

% When using background definitions for differentiation

% :- modeb(1,deriv1(+sequence,+time,+xdot,-linaccel)).
% :- modeb(1,deriv1(+sequence,+time,+thetadot,-angaccel)).

:- modeb(1,pow(+sequence,+time,+thetadot,#number,-power)).
:- modeb(1,pow(+sequence,+time,+theta,#number,-power)).
:- modeb(1,pow(+sequence,+time,+xdot,#number,-power)).
:- modeb(1,pow(+sequence,+time,+x,#number,-power)).

:- modeb(1,value(+sequence,+time,+thetadot,-value)).
:- modeb(1,value(+sequence,+time,+theta,-value)).
:- modeb(1,value(+sequence,+time,+xdot,-value)).
:- modeb(1,value(+sequence,+time,+x,-value)).

:- modeb(1,mult(+sequence,+time,+trignum,+power,-trigpow)).
:- modeb(1,mult(+sequence,+time,+trignum,+angaccel,-trigang)).
:- modeb(1,mult(+sequence,+time,+trignum,+linaccel,-triglin)).

%  Modes used to obtain equations for linear acceleration

:- modeb(1,lin_regress1(+linaccel,+value,#number,#number,#number,-linerror0)).
:- modeb(1,lin_regress1(+linaccel,+trignum,#number,#number,#number,-linerror1)).
:- modeb(1,lin_regress1(+linaccel,+trigang,#number,#number,#number,-linerror1)).
:- modeb(1,lin_regress1(+linaccel,+trigpow,#number,#number,#number,-linerror1)).

:- modeb(1,lin_regress2(+linaccel,+trignum,+trignum,#number,#number,#number,#number,-linerror2)).
:- modeb(1,lin_regress2(+linaccel,+trignum,+trigpow,#number,#number,#number,#number,-linerror2)).
:- modeb(1,lin_regress2(+linaccel,+trignum,+trigang,#number,#number,#number,#number,-linerror2)).
:- modeb(1,lin_regress2(+linaccel,+trigpow,+trigang,#number,#number,#number,#number,-linerror2)).
:- modeb(1,lin_regress2(+linaccel,+value,+value,#number,#number,#number,#number,-linerror2)).

% Equations with 3 and 4 independent variables are not really required, given the model

:- modeb(1,lin_regress3(+linaccel,+value,+value,+value,#number,#number,#number,
                        #number,#number,-linerror3)).
```

```
:- modeb(1,lin_regress4(+linaccel,+value,+value,+value,+value,#number,#number,
                        #number,#number,#number,#number,-linerror4)).

%  Modes used to obtain equations for angular acceleration

:- modeb(1,lin_regress1(+angaccel,+value,#number,#number,#number,-angerror0)).
:- modeb(1,lin_regress1(+angaccel,+trignum,#number,#number,#number,-angerror1)).
:- modeb(1,lin_regress1(+angaccel,+triglin,#number,#number,#number,-angerror1)).
:- modeb(1,lin_regress1(+angaccel,+trigpow,#number,#number,#number,-angerror1)).

:- modeb(1,lin_regress2(+angaccel,+trignum,+trignum,#number,#number,#number,#number,-angerror2)).
:- modeb(1,lin_regress2(+angaccel,+trignum,+trigpow,#number,#number,#number,#number,-angerror2)).
:- modeb(1,lin_regress2(+angaccel,+trignum,+triglin,#number,#number,#number,#number,-angerror2)).
:- modeb(1,lin_regress2(+angaccel,+trigpow,+triglin,#number,#number,#number,#number,-angerror2)).
:- modeb(1,lin_regress2(+angaccel,+value,+value,#number,#number,#number,#number,-angerror2)).

% Equations with 3 and 4 independent variables are not really required, given the model

:- modeb(1,lin_regress3(+angaccel,+value,+value,+value,#number,#number,#number,
                        #number,#number,-angerror3)).
:- modeb(1,lin_regress4(+angaccel,+value,+value,+value,+value,#number,#number,
                        #number,#number,#number,#number,-angerror4)).

:- modeb(1,lte(+linerror0,#number)).
:- modeb(1,lte(+linerror1,#number)).
:- modeb(1,lte(+linerror2,#number)).
:- modeb(1,lte(+linerror3,#number)).
:- modeb(1,lte(+linerror4,#number)).
:- modeb(1,lte(+angerror0,#number)).
:- modeb(1,lte(+angerror1,#number)).
:- modeb(1,lte(+angerror2,#number)).
:- modeb(1,lte(+angerror3,#number)).
:- modeb(1,lte(+angerror4,#number)).

:- set(i,4).            % maximum depth of variable-chaining
:- set(depth,50).       % maximum depth for theorem prover
:- set(clauselength,10).% maximum literals in a clause
:- set(nodes,5000).     % maximum number of nodes
:- set(search,user). % use user-defined cost and pruning
```

The following are peculiar to learning the equation for bank angle from the flight simulator data.

```
:- modeh(1,bank_angle(+mission,+angle)).

:- modeb(1,circular(+mission,-number)).
:- modeb(*,pow(+number,#number,-power)).

:- modeb(1,lin_regress1(+angle,+power,#number,#number,#number,-linerror1)).
:- modeb(1,lin_regress2(+angle,+power,+power,#number,#number,#number,#number,-linerror2)).
:- modeb(1,lin_regress3(+angle,+power,+power,+power,#number,#number,
                       #number,#number,#number,-linerror3)).
:- modeb(1,lin_regress4(+angle,+power,+power,+power,+power,#number,
                       #number,#number,#number,#number,#number,-linerror4)).

:- modeb(1,lte(+linerror1,#number)).
:- modeb(1,lte(+linerror2,#number)).
:- modeb(1,lte(+linerror3,#number)).
:- modeb(1,lte(+linerror4,#number)).
:- modeb(1,lte(+number,#number)).
:- modeb(1,gte(+number,#number)).

:- set(i,4).            % maximum depth of variable-chaining
:- set(depth,50).       % maximum depth for theorem prover
:- set(clauselength,10).% maximum literals in a clause
:- set(nodes,5000).     % maximum number of nodes
```

```
:- set(search,user). % use user-defined cost and pruning
```

Predicate properties

The following statements are used for the pole-and-cart.

```
:- determination(accel/9,value/4).
:- determination(accel/9,sin/4).
:- determination(accel/9,cos/4).
:- determination(accel/9,pow/5).
:- determination(accel/9,mult/5).
:- determination(accel/9,lin_regress1/6).
:- determination(accel/9,lin_regress2/8).
:- determination(accel/9,lte/2).
% :- determination(accel/9,lin_regress3/10). % not needed for model
% :- determination(accel/9,lin_regress4/12). % not needed for model


% Input argument ordering does not matter for multiplication
:- commutative(mult/5).
```

The following statements are used for the flight-simulator data.

```
:- determination(bank_angle/2,circular/2).
:- determination(bank_angle/2,pow/3).
:- determination(bank_angle/2,lte/2).
:- determination(bank_angle/2,gte/2).
:- determination(bank_angle/2,lin_regress1/6).
:- determination(bank_angle/2,lin_regress2/8).
% :- determination(bank_angle/2,lin_regress3/10).  % not used for model
% :- determination(bank_angle/2,lin_regress4/12).  % not used for model
```

The following are common to both problems.

```
:- lazy_evaluate(lin_regress1/6).
:- lazy_evaluate(lin_regress2/8).
:- lazy_evaluate(lin_regress3/10).
:- lazy_evaluate(lin_regress4/12).

%  States that only positive examples are needed to lazily evaluate regression
:- positive_only(lin_regress1/6).
:- positive_only(lin_regress2/8).
:- positive_only(lin_regress3/10).
:- positive_only(lin_regress4/12).

% Input argument ordering does not matter for these predicates
:- commutative(lin_regress1/6).
:- commutative(lin_regress2/8).
:- commutative(lin_regress3/10).
:- commutative(lin_regress4/12).

% General-purpose library function for linear regression
:- add_built_in(l_regressn/6).
```

Type definitions

The following type definitions are peculiar to the pole-and-cart.

```
list([]).
list([_|_]).

nat(X):- integer(X), X > 0.
time(X):- nat(X).

sequence(s(X)):- number(X).

x(x(X)):- number(X).
xdot(xd(X)):- number(X).
theta(t(X)):- number(X).
thetadot(td(X)):- number(X).
force(f(X)):- number(X).

linaccel(X):- number(X).
angaccel(X):- number(X).

derivative(d(X)):- number(X).

linerror0(X):- number(X).
linerror1(X):- number(X).
linerror2(X):- number(X).
linerror3(X):- number(X).
linerror4(X):- number(X).

angerror0(X):- number(X).
angerror1(X):- number(X).
angerror2(X):- number(X).
angerror3(X):- number(X).
angerror4(X):- number(X).

value(X,X):-
        number(X), !.
value(Fx,X):-
        Fx =.. [F,X], number(X), !.
```

The following type definitions are peculiar to the flight-simulator data.

```
list([]).
list([_|_]).

mission(mission(X)):-
        element(X,[14,17,34,37,38,41,42,43,44,52,59,63,67,70,77,81]).

time(time(X)):-
        number(X).

point(p(X,Y)):-
        number(X), number(Y).

points([p(X,Y)]):- !.
points([p(X,Y)|Points]):-
        points(Points).

xval(p(X,Y),X).
yval(p(X,Y),Y).

value(X,X):-
        number(X), !.
```

```
value(Fx,X):-
        Fx =.. [F,X], number(X), !.

power(X):- number(X).
```

Background predicates

The following definitions are peculiar to the pole-and-cart.

```
value(Sequence,Time,x(_),X):-
        trace(Sequence,Time,x(X),_,_,_,_), !.
value(Sequence,Time,xd(_),X):-
        trace(Sequence,Time,_,xd(X),_,_,_), !.
value(Sequence,Time,t(_),X):-
        trace(Sequence,Time,_,_,t(X),_,_), !.
value(Sequence,Time,td(_),X):-
        trace(Sequence,Time,_,_,_,td(X),_), !.
value(Sequence,Time,f(_),X):-
        trace(Sequence,Time,_,_,_,_,f(X)), !.
value(Sequence,Time,d(X),X):- !.
value(_,_,X,X):- number(X).

sin(Sequence,Time,Variable,Y):-
        value(Sequence,Time,Variable,X),
        Y is sin(X).
cos(Sequence,Time,Variable,Y):-
        value(Sequence,Time,Variable,X),
        Y is cos(X).

mult(Sequence,Time,A,B,C):-
        value(Sequence,Time,A,X1), value(Sequence,Time,B,X2),
        C is X1*X2.
pow(Sequence,Time,Variable,Pow,Val):-
        value(Sequence,Time,Variable,X),
        power(Pow),
        Val is X^Pow.

power(2). % only second power allowed

% 5-point formula for first derivative
% used to differentiate linear and angular velocity
deriv1(Sequence,T,Variable,d(D1)):-
        sampling_rate(Sequence,H),
        Tm2 is T - 2, % Tm2 is T - 2*H,
        Tm1 is T - 1, % Tm1 is T - H,
        Tp1 is T + 1,   % Tp1 is T + H,
        Tp2 is T + 2,   % Tp2 is T + 2*H,
        value(Sequence,Tm2,Variable,M2), value(Sequence,Tm1,Variable,M1),
        value(Sequence,Tp1,Variable,P1), value(Sequence,Tp2,Variable,P2),
        D1 is (M2 - 8*M1 + 8*P1 - P2)/(12*H).

sampling_rate(_,0.02).

% traces of sequences consulted from a file
:- consult(trace).
```

The following definitions are peculiar to the flight-simulator data.

```
% Only negative powers of radius allowed by the model
% Here up to 7th negative power of radius is allowed
```

```
pow(N,-7,N1):- N > 0, N1 is N^(-7).
pow(N,-6,N1):- N > 0, N1 is N^(-6).
pow(N,-5,N1):- N > 0, N1 is N^(-5).
pow(N,-4,N1):- N > 0, N1 is N^(-4).
pow(N,-3,N1):- N > 0, N1 is N^(-3).
pow(N,-2,N1):- N > 0, N1 is N^(-2).
pow(N,-1,N1):- N > 0, N1 is N^(-1).

% estimate circular trajectory by stratified sampling of points on the trajectory
% samples are from each of the 1st, 2nd and 3rd segments of the arcs

% check to see if radius is cached away
circular(Mission,Radius):-
        recorded(cache,estimate_circle(Mission,Radius),_), !.
circular(Mission,Radius):-
        sample_size(N),
        estimate_radius(Mission,0,N,0.0,Radius),
recorda(cache,estimate_circle(Mission,Radius),_). % cache result for efficiency

estimate_radius(_,N,N,RSum,Average):-
        !,
        Average is RSum/N.
estimate_radius(Mission,SNum,SampleSize,RSum,Radius):-
        segment(1,Start1,Finish1),
        get_random_point(Mission,Start1,Finish1,P1),
        segment(2,Start2,Finish2),
        get_random_point(Mission,Start2,Finish2,P2),
        segment(3,Start3,Finish3),
        get_random_point(Mission,Start3,Finish3,P3),
        circle(P1,P2,P3,Rad),
        RSum1 is RSum + Rad,
        SNum1 is SNum + 1,
        estimate_radius(Mission,SNum1,SampleSize,RSum1,Radius).

sample_size(10000).

% At most 100 points on each trajectory
% are divided into 3 segments
segment(1,1,34).
segment(2,34,67).
segment(3,67,101).


circle(p(X0,Y0),p(X1,Y1),p(X2,Y2),Radius):-
        not(X0/Y0=X1/Y1),not(X0/Y0=X2/Y2),not(X1/Y1=X2/Y2),
        Alpha is 2*(X1-X0), Beta is 2*(Y1-Y0),
        Gamma is 2*(X2-X0), Delta is 2*(Y2-Y0),
        C1 is (X1) * (X1) - (X0) * (X0) + (Y1) * (Y1) - (Y0) * (Y0),
        C2 is (X2) * (X2) - (X0) * (X0) + (Y2) * (Y2) - (Y0) * (Y0),
        B is (C2*Alpha-C1*Gamma)/(Alpha*Delta-Gamma*Beta),
        A is (C1-Beta*B)/Alpha,
        RSq is (X0-A) * (X0-A) + (Y0-B) * (Y0-B),
        Radius is RSq^0.5, !.

get_random_point(Mission,Start,Finish,p(X,Y)):-
        get_random(Start,Finish,Time),
        position(time(Time),Mission,X,Y).

% get a random integer in range [First,Last)
get_random(First,Last,INum):-
        X is random,
        Num is X*(Last-First) + First,
        INum1 is integer(Num),
        (INum1 = Last -> INum is Last-1; INum = INum1),
        INum >= First, INum < Last.
```

```
% traces of trajectory consulted from a file
:- consult(position).
```

The following definitions are common to both problems.

```
% linear regression with 4 independent variables
% testing
lin_regress4(Yval,X1val,X2val,X3val,X4val,M1,M2,M3,M4,C,Sigma,Error):-
        nonvar(Yval), nonvar(X1val), nonvar(X2val), nonvar(X3val), nonvar(X4val),
        value(Yval,Y), value(X1val,X1), value(X2val,X2), value(X3val,X3), value(X4val,X4),
        number(M1), number(M2), number(M3), number(M4), number(C), number(Sigma), !,
        Y0 is M1*X1 + M2*X2 + M3*X3 + M4*X4 + C,
        Residual is (Y - Y0)/Sigma,
        abs_val(Residual,Error).

% call to a C function to obtain regression line
lin_regress4(Y,X1,X2,X3,X4,M1,M2,M3,M4,C,Sigma,0.0):-
        list(Y), list(X1), list(X2), list(X3), list(X4),
        var(M1), var(M2), var(M3), var(M4), var(C), !,
        l_regressn(5,[Y,X1,X2,X3,X4],[C,M1,M2,M3,M4],Sigma,Fval,Rsq),
        write('[Parameters = '), write(C/M1/M2/M3/M4/Fval/Rsq), write(']'), nl,
        Fval > 6.63,
        Rsq > 0.80.

% for prediction
lin_regress4(Yval,X1val,X2val,X3val,X4val,M1,M2,M3,M4,C,_,0.0):-
        var(Yval), nonvar(X1val), nonvar(X2val), nonvar(X3val), nonvar(X4val),
        value(X1val,X1), value(X2val,X2), value(X3val,X3), value(X4val,X4),
        number(M1), number(M2), number(M3), number(M4), number(C), !,
        Yval is M1*X1 + M2*X2 + M3*X3 + M4*X4 + C.

% otherwise: for bottom clause
lin_regress4(Y,X1,X2,M1,M2,C,S,E):-
        X1 \= Y, X2 \= Y, X1 \= X2,
        skolem_var(M1), skolem_var(M2),
        skolem_var(C), skolem_var(S), skolem_var(E).

% linear regression with 3 independent variables
% testing
lin_regress3(Yval,X1val,X2val,X3val,M1,M2,M3,C,Sigma,Error):-
        nonvar(Yval), nonvar(X1val), nonvar(X2val), nonvar(X3val),
        value(Yval,Y), value(X1val,X1), value(X2val,X2), value(X3val,X3),
        number(M1), number(M2), number(M3), number(C), number(Sigma), !,
        Y0 is M1*X1 + M2*X2 + M3*X3 + C,
        Residual is (Y - Y0)/Sigma,
        abs_val(Residual,Error).

% call to a C function to obtain regression line
lin_regress3(Y,X1,X2,X3,M1,M2,M3,C,Sigma,0.0):-
        list(Y), list(X1), list(X2), list(X3),
        var(M1), var(M2), var(M3), var(C), !,
        l_regressn(4,[Y,X1,X2,X3],[C,M1,M2,M3],Sigma,Fval,Rsq),
        write('[Parameters = '), write(C/M1/M2/M3/Fval/Rsq), write(']'), nl,
        Fval > 6.63,
        Rsq > 0.80.

% for prediction
lin_regress3(Yval,X1val,X2val,X3val,M1,M2,M3,C,_,_):-
        var(Yval), nonvar(X1val), nonvar(X2val), nonvar(X3val),
        value(X1val,X1), value(X2val,X2), value(X3val,X3),
        number(M1), number(M2), number(M3), number(C), !,
        Yval is M1*X1 + M2*X2 + M3*X3 + C.

% otherwise: for bottom clause
```

```
lin_regress3(Y,X1,X2,M1,M2,C,S,E):-
        X1 \= Y, X2 \= Y, X1 \= X2,
        skolem_var(M1), skolem_var(M2),
        skolem_var(C), skolem_var(S), skolem_var(E).


% linear regression with 2 independent variables
% testing
lin_regress2(Yval,X1val,X2val,M1,M2,C,Sigma,Error):-
        nonvar(Yval), nonvar(X1val), nonvar(X2val),
        value(Yval,Y), value(X1val,X1), value(X2val,X2),
        number(M1), number(M2), number(C), number(Sigma), !,
        Y0 is M1*X1 + M2*X2 + C,
        Residual is (Y - Y0)/Sigma,
        abs_val(Residual,Error).

% call to a C function to obtain regression line
lin_regress2(Y,X1,X2,M1,M2,C,Sigma,0.0):-
        list(Y), list(X1), list(X2),
        var(M1), var(M2), var(C), !,
        l_regressn(3,[Y,X1,X2],[C,M1,M2],Sigma,Fval,Rsq),
        write('[Parameters = '), write(C/M1/M2/Fval/Rsq), write(']'), nl,
        Fval > 6.63,
        Rsq > 0.80.

% for prediction
lin_regress2(Yval,X1val,X2val,M1,M2,C,_,0.0):-
        var(Yval), nonvar(X1val), nonvar(X2val),
        value(X1val,X1), value(X2val,X2),
        number(M1), number(M2), number(C), !,
        Yval is M1*X1 + M2*X2 + C.

% otherwise: for bottom clause
lin_regress2(Y,X1,X2,M1,M2,C,S,E):-
        X1 \= Y, X2 \= Y, X1 \= X2,
        skolem_var(M1), skolem_var(M2),
        skolem_var(C), skolem_var(S), skolem_var(E).


% linear regression with 1 independent variable
% testing
lin_regress1(Yval,Xval,M,C,Sigma,Error):-
        nonvar(Yval), nonvar(Xval), value(Yval,Y), value(Xval,X),
        number(M), number(C), number(Sigma), !,
        Y1 is M*X + C,
        Residual is (Y - Y1)/Sigma,
        abs_val(Residual,Error).

% call to a C function to obtain regression line
lin_regress1(Y,X,M,C,Sigma,0.0):-
        list(X), list(Y), !,
        l_regressn(2,[Y,X],[C,M],Sigma,Fval,Rsq),
        write('[Parameters = '), write(C/M/Fval/Rsq), write(']'), nl,
        Fval > 6.63,
        Rsq > 0.80.

% for prediction
lin_regress1(Yval,Xval,M,C,_,0.0):-
        var(Yval), nonvar(Xval), value(Xval,X),
        number(M), number(C), !,
        Yval is M*X + C.

% otherwise: for bottom clause
lin_regress1(Y,X,M,C,S,E):-
        X \= Y,
        skolem_var(M), skolem_var(C),
```

```
skolem_var(E), skolem_var(E).


abs_val(X,X):- X >= 0, !.
abs_val(X,Y):- Z is 1.0 * X, Z < 0, Y is -Z.


lte(X,Y):- number(X), not(var(Y)), number(Y), !, X =< Y.
lte(X,Y1):-
        ubound(X,Y1).

gte(X,Y):- number(X), not(var(Y)), number(Y), !, X >= Y.
gte(X,Y1):-
        lbound(X,Y1).

ubound(0.0,2.0):- !.
ubound(X,X):- !.
lbound(0.0,-2.0):- !.
lbound(X,X):- !.

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

Constraints, costs, and pruning

The following are peculiar to the pole-and-cart.

```
false:-
        hypothesis(accel(_,_,_,_,_,_,_,Tdd,Xdd),Body,_),
        not(calc_and_bounds_err(Xdd,Body)), !.
        % not(calc_and_bounds_err(Tdd,Body)), !. % when calculating Tdd

% user-specified pruning

% do not extend clauses after computing
% linear or angular accelerations except to bound error
abandon:-
        hypothesis(accel(_,_,_,_,_,_,_,Tdd,Xdd),Body,_),
        calculated(Xdd,Body,Error,NextLit),        % there is an equation for Xdd
        % calculated(Tdd,Body,Error,NextLit),      % there is an equation for Tdd
        NextLit \= true,    % there is a literal after equation
        not(bounds(Error,NextLit)), !.             % but does not provide error-bounds

% do not pursue clauses that cannot possibly obtain equations and
% error bounds within clauselength restrictions
abandon:-
        hypothesis(accel(_,_,_,_,_,_,_,Tdd,Xdd),Body,_),
        not(calculated(Xdd,Body,_,_)),
        % not(calculated(Tdd,Body,_,_)),
        set(clauselength,C),
        number_of_lits(Body,N),
        LitsLeft is C - N - 1, % 1 for the head of the clause
        LitsLeft < 2, !. % 2 literals needed for equation and error

% get mean-square error of a clause over positive examples
mse(Head,Body,inf):-
```

```
        ground((Head:-Body)), !.
mse(accel(Seq,Sample,X,Xd,T,Td,F,Tdd,Xdd),Body,inf):-
        % not(calculated(Xdd,Body,_,_)), !.
        not(calculated(Tdd,Body,_,_)), !.
mse(accel(Seq,Sample,X,Xd,T,Td,F,Tdd,Xdd),Body,E):-
        retract_all(mse,sse(_)),
        retract_all(mse,count(_)),
        recorda(mse,sse(0.0),_),
        recorda(mse,count(0),_),
        example(pos,_,accel(Seq,Sample,X,Xd,T,Td,F,Tdd,Xdd1)),
        % example(pos,_,accel(Seq,Sample,X,Xd,T,Td,F,Tdd1,Xdd)),
        progol_call(Body),
        SE is (Xdd - Xdd1)^2,
        % SE is (Tdd - Tdd1)^2,
        recorded(mse,sse(S),Key),
        recorded(mse,count(N),Key1),
        erase(Key), erase(Key1),
        S1 is S + SE,
        N1 is N + 1,
        recorda(mse,sse(S1),_),
        recorda(mse,count(N1),_),
        fail.
mse(_,_,E):-
        recorded(mse,sse(S),Key),
        recorded(mse,count(N),Key1),
        erase(Key), erase(Key1),
        N > 0, !,
        E is S/N.
mse(_,_,inf).
```

The following are peculiar to the flight-simulator data.

```
false:-
        hypothesis(bank_angle(_,Angle),Body,_),
        not(calc_and_bounds_err(Angle,Body)), !.

% do not extend clauses after computing
% linear or angular accelerations except to bound error
abandon:-
hypothesis(bank_angle(_,Angle),Body,_),
        calculated(Angle,Body,Error,NextLit),      % there is an equation for Angle
        NextLit \= true,    % there is a literal after equation
        not(bounds(Error,NextLit)), !.             % but does not provide error-bounds

% do not pursue clauses that cannot possibly obtain equations and
% error bounds within clauselength restrictions
abandon:-
hypothesis(bank_angle(_,Angle),Body,_),
        not(calculated(Angle,Body,_,_)),
        set(clauselength,C),
        number_of_lits(Body,N),
        LitsLeft is C - N - 1, % 1 for the head of the clause
```

```
        LitsLeft < 2, !. % 2 literals needed for equation and error

% get mean-square error of a clause over positive examples
mse(Head,Body,inf):-
        ground((Head:-Body)), !.
mse(bank_angle(_,Angle),Body,inf):-
        not(calculated(Angle,Body,_,_)), !.
mse(bank_angle(Mission,Angle),Body,E):-
        retract_all(mse,sse(_)),
        retract_all(mse,count(_)),
        recorda(mse,sse(0.0),_),
        recorda(mse,count(0),_),
        example(pos,_,bank_angle(Mission,Angle1)),
        progol_call(Body),
        SE is (Angle - Angle1)^2,
        recorded(mse,sse(S),Key),
        recorded(mse,count(N),Key1),
        erase(Key), erase(Key1),
        S1 is S + SE,
        N1 is N + 1,
        recorda(mse,sse(S1),_),
        recorda(mse,count(N1),_),
        fail.
mse(_,_,E):-
        recorded(mse,sse(S),Key),
        recorded(mse,count(N),Key1),
        erase(Key), erase(Key1),
        N > 0, !,
        E is S/N.
mse(_,_,inf).
```

The following are common to both domains.

```
% user-specified cost of a clause: here the same as the mean square error
cost((Head:-Body),Label,Cost):-
        mse(Head,Body,Cost).

number_of_lits((Lit,Lits),N):-
        !,
        number_of_lits(Lits,N1),
        N is N1 + 1.
number_of_lits(true,0):- !.
number_of_lits(Lit,1).


% variable is computed by an equation in the body with error E

calculated(Variable,(lin_regress1(Y,_,_,_,_,E),Lits),E,Lits):-
        Variable == Y, !.
calculated(Variable,(lin_regress2(Y,_,_,_,_,_,_,E),Lits),E,Lits):-
        Variable == Y, !.
calculated(Variable,(lin_regress3(Y,_,_,_,_,_,_,_,_,E),Lits),E,Lits):-
```

```
        Variable == Y, !.
calculated(Variable,(lin_regress4(Y,_,_,_,_,_,_,_,_,_,E),Lits),E,Lits):-
        Variable == Y, !.
calculated(Variable,(_,Lits),E,Left):-
        calculated(Variable,Lits,E,Left), !.
calculated(Variable,lin_regress1(Y,_,_,_,_,E),E,true):-
        Variable == Y, !.
calculated(Variable,lin_regress2(Y,_,_,_,_,_,_,E),E,true):-
        Variable == Y, !.
calculated(Variable,lin_regress3(Y,_,_,_,_,_,_,_,_,E),E,true):-
        Variable == Y, !.
calculated(Variable,lin_regress4(Y,_,_,_,_,_,_,_,_,_,E),E,true):-
        Variable == Y, !.

% error bounds exist on value predicted by an equation
calc_and_bounds_err(Variable,(lin_regress1(Y,_,_,_,_,E),Lits)):-
        Variable == Y, !,
        bounds_error(E,Lits).
calc_and_bounds_err(Variable,(lin_regress2(Y,_,_,_,_,_,_,E),Lits)):-
        Variable == Y, !,
        bounds_error(E,Lits).
calc_and_bounds_err(Variable,(lin_regress3(Y,_,_,_,_,_,_,_,_,E),Lits)):-
        Variable == Y, !,
        bounds_error(E,Lits).
calc_and_bounds_err(Variable,(lin_regress4(Y,_,_,_,_,_,_,_,_,_,E),Lits)):-
        Variable == Y, !,
        bounds_error(E,Lits).
calc_and_bounds_err(Variable,(_,Lits)):-
        calc_and_bounds_err(Variable,Lits).

bounds_error(E,(Lit,_)):-
        bounds(E,Lit), !.
bounds_error(E,(_,Lits)):-
        !,
        bounds_error(E,Lits).
bounds_error(E,Lit):-
        bounds(E,Lit), !.

bounds(E,lte(Y,N)):-
        E == Y, !,
        number(N).

retract_all(Area,Fact):-
        recorded(Area,Fact,Key),
        erase(Key),
        fail.
retract_all(_,_).
```

C.2 Attributes for propositional programs

The attributes provided for the propositional programs were obtained by computing all terms sanctioned by the mode language given to the ILP program.

For the pole-and-cart data, these were as follows: x, $\dot{x}$, θ, $\dot{\theta}$, $\ddot{x}$, $\ddot{\theta}$, $sin\theta$, $cos\theta$, θ^2, $\dot{\theta}^2$, x^2, $\dot{x}^2$, $\theta^2 cos\theta$, $\dot{\theta}^2 cos\theta$, $x^2 cos\theta$, $\dot{x}^2 cos\theta$, $\theta^2 sin\theta$, $\dot{\theta}^2 sin\theta$, $x^2 sin\theta$, $\dot{x}^2 sin\theta$, $\ddot{x} cos\theta$, $\ddot{x} sin\theta$, $\ddot{\theta} cos\theta$, $\ddot{\theta} sin\theta$, F. In keeping with the procedure adopted with the ILP program, all attributes containing $\ddot{x}$ were ignored when constructing a theory for that variable. Similarly, all attributes containing $\ddot{\theta}$ were ignored when constructing a theory for that variable. The attribute F is only used when force is allowed to take the values $\pm 10N$.

The attributes used in analysing the flight-simulator data were as follows: $r, r^{-1}, r^{-2}, r^{-3}, r^{-4}, r^{-5}, r^{-6}, r^{-7}$. Here r stands for the estimated radius of the circular trajectory. The estimation is done using the same sampling strategy used by the ILP program.

D Equations obtained by Progol

D.1 Pole-and-cart

We first present a mathematical representation of the equations obtained for each pole-and-cart configuration. The actual Prolog definitions found by *Progol* follow.

F	m/M	Target equation	Equation obtained
0	0.1	$\ddot{\theta} = 0.0 + 14.7 sin\theta - 1.5\ddot{x} cos\theta$ $\ddot{x} = 0.0 - 0.045\ddot{\theta} cos\theta + 0.045\dot{\theta}^2 sin\theta$	$\ddot{\theta} = -0.002 + 12.644 sin\theta - 4.543\ddot{x} cos\theta$ $\ddot{x} = 0.0 - 0.045\ddot{\theta} cos\theta + 0.043\dot{\theta}^2 sin\theta$
	1.0	$\ddot{\theta} = 0.0 + 14.7 sin\theta - 1.5\ddot{x} cos\theta$ $\ddot{x} = 0.0 - 0.25\ddot{\theta} cos\theta + 0.25\dot{\theta}^2 sin\theta$	$\ddot{\theta} = -0.0003 - 4.099\ddot{x} cos\theta + 1.067\dot{\theta}^2 sin\theta$ $\ddot{x} = 0.0 - 0.25\ddot{\theta} cos\theta + 0.247\dot{\theta}^2 sin\theta$
	10.0	$\ddot{\theta} = 0.0 + 14.7 sin\theta - 1.5\ddot{x} cos\theta$ $\ddot{x} = 0.0 - 0.45\ddot{\theta} cos\theta + 0.45\dot{\theta}^2 sin\theta$	$\ddot{\theta} = 0.0 + 13.182 sin\theta - 1.581\ddot{x} cos\theta$ $\ddot{x} = 0.0 - 0.455\ddot{\theta} cos\theta + 0.457\dot{\theta}^2 sin\theta$
+10	0.1	$\ddot{\theta} = 0.0 + 14.7 sin\theta - 1.5\ddot{x} cos\theta$ $\ddot{x} = 9.09 - 0.045\ddot{\theta} cos\theta + 0.045\dot{\theta}^2 sin\theta$	$\ddot{\theta} = 0.253 + 14.68 sin\theta - 1.526\ddot{x} cos\theta$ $\ddot{x} = 9.091 - 0.045\ddot{\theta} cos\theta + 0.045\dot{\theta}^2 sin\theta$
	1.0	$\ddot{\theta} = 0.0 + 14.7 sin\theta - 1.5\ddot{x} cos\theta$ $\ddot{x} = 5.0 - 0.25\ddot{\theta} cos\theta + 0.25\dot{\theta}^2 sin\theta$	$\ddot{\theta} = 0.041 + 14.647 sin\theta - 1.510\ddot{x} cos\theta$ $\ddot{x} = 5.0 - 0.25\ddot{\theta} cos\theta + 0.251\dot{\theta}^2 sin\theta$
	10.0	$\ddot{\theta} = 0.0 + 14.7 sin\theta - 1.5\ddot{x} cos\theta$ $\ddot{x} = 9.09 - 0.45\ddot{\theta} cos\theta + 0.45\dot{\theta}^2 sin\theta$	$\ddot{\theta} = 0.0004 + 14.692 sin\theta - 1.5\ddot{x} cos\theta$ $\ddot{x} = 9.094 - 0.454\ddot{\theta} cos\theta + 0.453\dot{\theta}^2 sin\theta$
±10	0.1	$\ddot{\theta} = 0.0 + 14.7 sin\theta - 1.5\ddot{x} cos\theta \quad (F = +10)$ $\ddot{\theta} = 0.0 + 14.7 sin\theta - 1.5\ddot{x} cos\theta \quad (F = -10)$ $\ddot{x} = 9.09 - 0.045\ddot{\theta} cos\theta + 0.045\dot{\theta}^2 sin\theta \quad (F = +10)$ $\ddot{x} = -9.09 - 0.045\ddot{\theta} cos\theta + 0.045\dot{\theta}^2 sin\theta \quad (F = -10)$	$\ddot{\theta} = 0.583 + 14.667 sin\theta - 1.56\ddot{x} cos\theta$ $\ddot{\theta} = -0.633 + 14.686 sin\theta - 1.565\ddot{x} cos\theta$ $\ddot{x} = 9.088 - 0.046\ddot{\theta} cos\theta + 0.046\dot{\theta}^2 sin\theta$ $\ddot{x} = -9.09 - 0.046\ddot{\theta} cos\theta + 0.047\dot{\theta}^2 sin\theta$
	1.0	$\ddot{\theta} = 0.0 + 14.7 sin\theta - 1.5\ddot{x} cos\theta \quad (F = +10)$ $\ddot{\theta} = 0.0 + 14.7 sin\theta - 1.5\ddot{x} cos\theta \quad (F = -10)$ $\ddot{x} = 5.0 - 0.25\ddot{\theta} cos\theta + 0.25\dot{\theta}^2 sin\theta \quad (F = +10)$ $\ddot{x} = -5.0 - 0.25\ddot{\theta} cos\theta + 0.25\dot{\theta}^2 sin\theta \quad (F = -10)$	$\ddot{\theta} = 0.556 + 14.338 sin\theta - 1.57\ddot{x} cos\theta$ $\ddot{\theta} = -0.574 + 14.314 sin\theta - 1.573\ddot{x} cos\theta$ $\ddot{x} = 5.103 - 0.016\ddot{\theta} cos\theta + 0.241\dot{\theta}^2 sin\theta$ $\ddot{x} = -5.001 - 0.250\ddot{\theta} cos\theta + 0.249\dot{\theta}^2 sin\theta$
	10.0	$\ddot{\theta} = 0.0 + 14.7 sin\theta - 1.5\ddot{x} cos\theta \quad (F = +10)$ $\ddot{\theta} = 0.0 + 14.7 sin\theta - 1.5\ddot{x} cos\theta \quad (F = -10)$ $\ddot{x} = 9.09 - 0.45\ddot{\theta} cos\theta + 0.45\dot{\theta}^2 sin\theta \quad (F = +10)$ $\ddot{x} = -9.09 - 0.45\ddot{\theta} cos\theta + 0.45\dot{\theta}^2 sin\theta \quad (F = -10)$	$\ddot{\theta} = 0.075 + 14.639 sin\theta - 1.503\ddot{x} cos\theta$ $\ddot{\theta} = -0.044 + 14.634 sin\theta - 1.502\ddot{x} cos\theta$ $\ddot{x} = 9.096 - 0.454\ddot{\theta} cos\theta + 0.451\dot{\theta}^2 sin\theta$ $\ddot{x} = -9.096 - 0.454\ddot{\theta} cos\theta + 0.453\dot{\theta}^2 sin\theta$

Figure 12: Descriptions of equations obtained by Progol on the pole-and-cart data.

Prolog descriptions

```
% F = 0, m = 0.1, M = 1.0

accel(A,B,C,D,E,F,f(0),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,H,K),
        lin_regress2(G,I,K,12.644,-4.543,-0.00222,0.040,L), lte(L,2.000).

accel(A,B,C,D,E,F,f(0),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,G,K), pow(A,B,F,2,L),
        mult(A,B,I,L,M), lin_regress2(H,M,K,0.043,-0.045,0,0.003,N), lte(N,2.000).

% F = 0, m = 1.0, M = 1.0

accel(A,B,C,D,E,F,f(0),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,H,K), pow(A,B,F,2,L), mult(A,B,I,L,M),
        lin_regress2(G,M,K,1.067,-4.099,-0.000251,0.031,N), lte(N,2.000).
```

```
accel(A,B,C,D,E,F,f(0),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,G,K), pow(A,B,F,2,L),
        mult(A,B,I,L,M), lin_regress2(H,M,K,0.247,-0.250,0,0.003,N), lte(N,2.000).

% F = 0, m = 1.0, M = 0.1

accel(A,B,C,D,E,F,f(0),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,H,K),
        lin_regress2(G,I,K,13.182,-1.581,0,0.037,L), lte(L,2.000).

accel(A,B,C,D,E,F,f(0),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,G,K), pow(A,B,F,2,L),
        mult(A,B,I,L,M), lin_regress2(H,M,K,0.457,-0.455,0,0.004,N), lte(N,2.000).

% F = +10, m = 0.1, M = 1.0

accel(A,B,C,D,E,F,f(10),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,H,K),
        lin_regress2(G,I,K,14.681,-1.526,0.253,0.039,L), lte(L,2.000).

accel(A,B,C,D,E,F,f(10),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,G,K), pow(A,B,F,2,L),
        mult(A,B,I,L,M), lin_regress2(H,M,K,0.045,-0.045,9.091,0.003,N), lte(N,2.000).

% F = +10, m = 1.0, M = 1.0

accel(A,B,C,D,E,F,f(10),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,H,K),
        lin_regress2(G,I,K,14.647,-1.510,0.083,0.041,L), lte(L,2.000).

accel(A,B,C,D,E,F,f(10),G,H) :-
    sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,G,K), pow(A,B,F,2,L),
    mult(A,B,I,L,M), lin_regress2(H,M,K,0.251,-0.250,5.000,0.004,N), lte(N,2.000).

% F = +10, m = 1.0, M = 0.1

accel(A,B,C,D,E,F,f(10),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,H,K),
        lin_regress2(G,I,K,14.692,-1.500,3.795e-04,0.038,L), lte(L,2.000).

accel(A,B,C,D,E,F,f(10),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,G,K), pow(A,B,F,2,L),
        mult(A,B,I,L,M), lin_regress2(H,M,K,0.453,-0.454,9.094,0.013,N), lte(N,2.000).

% F = +/- 10, m = 0.1, M = 1.0

accel(A,B,C,D,E,F,f(10.000),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,H,K),
        lin_regress2(G,I,K,14.667,-1.560,0.583,0.044,L), lte(L,2.000).
accel(A,B,C,D,E,F,f(-10.000),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,H,K),
        lin_regress2(G,I,K,14.686,-1.565,-0.633,0.040,L), lte(L,2.000).

accel(A,B,C,D,E,F,f(10.000),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,G,K), pow(A,B,F,2,L),
        mult(A,B,I,L,M), lin_regress2(H,M,K,0.046,-0.046,9.088,0.003,N), lte(N,2.000).
accel(A,B,C,D,E,F,f(-10.000),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,G,K), pow(A,B,F,2,L),
        mult(A,B,I,L,M), lin_regress2(H,M,K,0.047,-0.046,-9.090,0.003,N), lte(N,2.000).

% F = +/- 10, m = 1.0, M = 1.0

accel(A,B,C,D,E,F,f(10.000),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,H,K),
        lin_regress2(G,I,K,14.338,-1.570,0.556,0.041,L), lte(L,2.000).
accel(A,B,C,D,E,F,f(-10.000),G,H) :-
```

```
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,H,K),
        lin_regress2(G,I,K,14.314,-1.573,-0.574,0.039,L), lte(L,2.000).

accel(A,B,C,D,E,F,f(10.000),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,G,K), pow(A,B,C,2,L),
        mult(A,B,I,L,M), lin_regress2(H,M,K,-0.016,-0.241,5.103,0.025,N), lte(N,2.000).
accel(A,B,C,D,E,F,f(-10.000),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,G,K), pow(A,B,F,2,L),
        mult(A,B,I,L,M), lin_regress2(H,M,K,0.249,-0.250,-5.001,0.003,N), lte(N,2.000).

% F = +/- 10, m = 1.0, M = 0.1

accel(A,B,C,D,E,F,f(10.000),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,H,K),
        lin_regress2(G,I,K,14.639,-1.503,0.075,0.043,L), lte(L,2.000).
accel(A,B,C,D,E,F,f(-10.000),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,H,K),
        lin_regress2(G,I,K,14.634,-1.502,-0.044,0.040,L), lte(L,2.000).

accel(A,B,C,D,E,F,f(10.000),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,G,K), pow(A,B,F,2,L),
        mult(A,B,I,L,M), lin_regress2(H,M,K,0.451,-0.454,9.096,0.009,N), lte(N,2.000).
accel(A,B,C,D,E,F,f(-10.000),G,H) :-
        sin(A,B,E,I), cos(A,B,E,J), mult(A,B,J,G,K), pow(A,B,F,2,L),
        mult(A,B,I,L,M), lin_regress2(H,M,K,0.453,-0.454,-9.096,0.009,N), lte(N,2.000).
```

D.2 Flight simulator

A mathematical representation of the equation obtained from the flight simulator data is:

$$\theta = -\frac{2 \times 10^3}{r} + \frac{4 \times 10^6}{r^3} + 0.33$$

where r is the radius of the turn. The actual Prolog clause found is:

```
bank_angle(A,B) :-
        estimate_circle(A,C), pow(C,-1,D), pow(C,-3,E),
        lin_regress2(B,D,E,-2162.297,3.918e+06,0.337,1.077,F),
        lte(F,2.000).
```

E Equations obtained from propositional learners

In this section, the neural network "theory", namely a listing of the weights is not reproduced. Further, the regression tree output for the pole-and-cart data is too complicated to reproduce here (see Figure 7 for sizes of these trees).

E.1 Pole-and-cart

Linear regression produces the following results for the 9 pole-and-cart configurations.

F	m/M	Target equation	Equation obtained
0	0.1	$\ddot{\theta} = 0.0 + 14.7sin\theta - 1.5\ddot{x}cos\theta$	$\ddot{\theta} = -0.002 + 12.528\theta - 4.63\ddot{x}cos\theta$
		$\ddot{x} = 0.0 - 0.045\ddot{\theta}cos\theta + 0.045\dot{\theta}^2 sin\theta$	$\ddot{x} = 0.0 - 0.045\ddot{\theta}cos\theta + 0.043\dot{\theta}^2 sin\theta$
	1.0	$\ddot{\theta} = 0.0 + 14.7sin\theta - 1.5\ddot{x}cos\theta$	$\ddot{\theta} = -0.0003 - 4.049\ddot{x} + 1.034\dot{\theta}^2 sin\theta$
		$\ddot{x} = 0.0 - 0.25\ddot{\theta}cos\theta + 0.25\dot{\theta}^2 sin\theta$	$\ddot{x} = 0.0 - 0.25\ddot{\theta}cos\theta + 0.247\dot{\theta}^2 sin\theta$
	10.0	$\ddot{\theta} = 0.0 + 14.7sin\theta - 1.5\ddot{x}cos\theta$	$\ddot{\theta} = 0.001 - 2.22\ddot{x} + 1.084\dot{\theta}^2 sin\theta$
		$\ddot{x} = 0.0 - 0.45\ddot{\theta}cos\theta + 0.45\dot{\theta}^2 sin\theta$	$\ddot{x} = 0.0 - 0.455\ddot{\theta}cos\theta + 0.457\dot{\theta}^2 sin\theta$
+10	0.1	$\ddot{\theta} = 0.0 + 14.7sin\theta - 1.5\ddot{x}cos\theta$	$\ddot{\theta} = 0.213 + 14.68sin\theta - 1.522\ddot{x}cos\theta$
		$\ddot{x} = 9.09 - 0.045\ddot{\theta}cos\theta + 0.045\dot{\theta}^2 sin\theta$	$\ddot{x} = 9.091 - 0.045\ddot{\theta}cos\theta + 0.045\dot{\theta}^2 sin\theta$
	1.0	$\ddot{\theta} = 0.0 + 14.7sin\theta - 1.5\ddot{x}cos\theta$	$\ddot{\theta} = 0.070 + 14.647sin\theta - 1.509\ddot{x}cos\theta$
		$\ddot{x} = 5.0 - 0.25\ddot{\theta}cos\theta + 0.25\dot{\theta}^2 sin\theta$	$\ddot{x} = 5.0 - 0.25\ddot{\theta}cos\theta + 0.251\dot{\theta}^2 sin\theta$
	10.0	$\ddot{\theta} = 0.0 + 14.7sin\theta - 1.5\ddot{x}cos\theta$	$\ddot{\theta} = 0.0004 + 13.650sin\theta - 1.49\ddot{x}$
		$\ddot{x} = 9.09 - 0.45\ddot{\theta}cos\theta + 0.45\dot{\theta}^2 sin\theta$	$\ddot{x} = 9.094 - 0.454\ddot{\theta}cos\theta + 0.453\dot{\theta}^2 sin\theta$
±10	0.1	$\ddot{\theta} = 0.0 + 14.7sin\theta - 1.5\ddot{x}cos\theta \quad (F = + \pm 10)$	$\ddot{\theta} = 0.002 + 14.639sin\theta - 1.49\ddot{x}$
		$\ddot{x} = 9.09 - 0.045\ddot{\theta}cos\theta + 0.045\dot{\theta}^2 sin\theta \quad (F = +10)$	$\ddot{x} = 0.046\ddot{\theta}cos\theta + 0.047\dot{\theta}^2 sin\theta + 0.909F$
		$\ddot{x} = -9.09 - 0.045\ddot{\theta}cos\theta + 0.045\dot{\theta}^2 sin\theta \quad (F = -10)$	
	1.0	$\ddot{\theta} = 0.0 + 14.7sin\theta - 1.5\ddot{x}cos\theta \quad (F = \pm 10)$	$\ddot{\theta} = 0.0003 + 14.651sin\theta - 1.49\ddot{x}$
		$\ddot{x} = 5.0 - 0.25\ddot{\theta}cos\theta + 0.25\dot{\theta}^2 sin\theta \quad (F = +10)$	$\ddot{x} = -0.25\ddot{\theta}cos\theta + 0.25\dot{\theta}^2 sin\theta + 0.499F$
		$\ddot{x} = -5.0 - 0.25\ddot{\theta}cos\theta + 0.25\dot{\theta}^2 sin\theta \quad (F = -10)$	
	10.0	$\ddot{\theta} = 0.0 + 14.7sin\theta - 1.5\ddot{x}cos\theta \quad (F = \pm 10)$	$\ddot{\theta} = 0.022 + 14.3sin\theta - 1.489\ddot{x}$
		$\ddot{x} = 9.09 - 0.45\ddot{\theta}cos\theta + 0.45\dot{\theta}^2 sin\theta \quad (F = +10)$	$\ddot{x} = 0.0007 + 3.955sin\theta - 0.539\ddot{\theta} + 0.546F$
		$\ddot{x} = -9.09 - 0.45\ddot{\theta}cos\theta + 0.45\dot{\theta}^2 sin\theta \quad (F = -10)$	

Figure 13: Descriptions of equations obtained by linear regression on the pole-and-cart data.

E.2 Flight simulator

The equation obtained by linear regression on the flight simulator data is:

$$\theta \approx -\frac{-2 \times 10^3}{r} + \frac{4 \times 10^6}{r^3} + 0.3$$

where r is the radius of the turn.

The regression tree program applied to the same data yielded the following tree:

```
                          REGRESSION TREE DIAGRAM
                                     |
                                     |
            +-------------r**{-2}=<3.154e-05 -----------------+
            |                                                 |
+---r**{-1}=<4.500e-03---+                          +---r**{-1}=<8.500e-03---+
|                        |                          |                        |
|                        |                          |                        |

theta = -6.6       theta = -10             theta = -13             theta = -18
```