How does Predicate Invention affect Human Comprehensibility?

Ute Schmid[1], Christina Zeller[1], Tarek Besold[2], Alireza Tamaddoni-Nezhad[3], Stephen Muggleton[3]

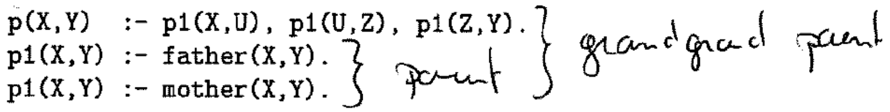
[1] University of Bamberg, Germany
[2] University of Bremen, Germany
[3] Imperial College London, UK

Abstract. During the 1980s Michie defined Machine Learning in terms of two orthogonal axes of performance: *predictive accuracy* and *comprehensibility of generated hypotheses*. Since predictive accuracy was readily measurable and comprehensibility not so, later definitions in the 1990s, such as that of Mitchell, tended to use a one-dimensional approach to Machine Learning based solely on predictive accuracy, ultimately favouring statistical over symbolic Machine Learning approaches. In this paper we provide a definition of comprehensibility of hypotheses which can be estimated using human participant trials. We present the results of experiments testing human comprehensibility of logic programs learned with and without predicate invention. Results indicate that comprehensibility is affected not only by the complexity of the presented program but also by the existence of anonymous predicate symbols.

1 Introduction

Within Artificial Intelligence (AI) *comprehensibility* of symbolic knowledge is viewed as one of the defining factors which distinguishes logic-based representations from statistical or neural ones. However, to the authors' knowledge, no operational criterion of comprehensibility exists in the literature. This paper addresses the issue by introducing such a definition which is inspired by "Comprehension Tests", administered to children at primary school. Such a test comprises the presentation of a piece of text, followed by questions which probe the child's understanding. Answers to questions in some cases may not be directly stated, but instead inferred from the text. Once the test is scored, the degree of the pupil's answers can be assessed numerically.

In the same fashion, our operational definition of comprehensibility is based on presentation of a logic program to an experimental participant, who is given time to study it, after which the score is used to assess their degree of comprehension. The detailed results of such a test can be used to identify factors in the program which affect its comprehensibility both for individuals and for groups of participants. The existence of an experimental methodology for testing comprehensibility has the potential to provide empirical input for improvement of Machine Learning systems for which the generated hypotheses are intended to provide insights.



```
p(X,Y)  :- p1(X,U), p1(U,Z), p1(Z,Y).
p1(X,Y) :- father(X,Y).
p1(X,Y) :- mother(X,Y).
```

parent
grandgrand parent

Fig. 1. Example of student giving meaningful names to predicate symbols.

Figure 1 provides an example of such a test in which students were asked about a given program in which some predicate names were meaningful (i.e., had publicly recognisable names like *father* and *mother*) and others were anonymous (i.e., had privately defined names like *p* and *p1*). In this case, high-scoring students often unexpectedly annotated the answer scripts to indicate the name they believed to be correct.

Given renewed interest within Inductive Logic Programming (ILP) in the use of predicate invention [11, 12, 7, 1, 2] this paper explores the effects on comprehensibility of using anonymous definitions within logic programs. Within experiments we assess students' understanding of such programs within the kinship domain. Empirical results indicate that comprehensibility is positively correlated to the degree with which new predicates produce compact descriptions. Additionally comprehensibility correlates with the degree to which participants can successfully match the presented predicate with one they are already familiar with. However, somewhat surprisingly, it is negatively correlated with the amount of time taken to inspect the definitions.

The paper is arranged as follows. In Section 2 we discuss existing work relevant to the paper. The framework, including relevant definitions and their relationship to experimental hypotheses is described in Section 3. Section 4 describes the experiments, including details of the questionnaires, experimental procedure, results and discussion. Finally in Section 5 we conclude the paper and discuss further work.

2 Related Work

2.1 Comprehensibility

In the late 1980s Michie [8] suggested the idea of using both comprehensibility of hypotheses and predictive accuracy as performance indicators for Machine Learning. He proposed three criteria. The *weak criterion* defines Machine Learning as occurring whenever a system generates an updated basis building on sample data for improving its performance on subsequent data. The focus is put exclusively on the prediction and problem-solving aspects. The *strong criterion* expands the weak version in a second direction, requiring the system to be able additionally to "communicate its internal updates in explicit symbolic form". Lastly his *ultra-strong criterion* additionally requires the communication of updates to be "operationally effective", in which case the user is required to understand the update and any consequences to be drawn from it. While ILP systems clearly meet the weak and strong criterion (since learning outcomes are represented as symbolic logic programs), only very limited attention has been given to checking whether the ultra-strong criterion holds, which requires testing whether the user comprehends generated hypotheses.

One Machine Learning approach which engages with issues related to comprehensibility is Argument-Based Machine Learning (ABML) [9]. ABML applies methods from argumentation in combination with a rule-learning approach. Explanations provided by domain experts concerning positive or negative arguments serve to enrich selected learning examples by being included in the learning data. Although ABML enhances the degree of explanation within a Machine Learning context, like ILP, ABML fails to pass Michie's ultra-strong test since no demonstration of user comprehensibility of learned hypotheses is guaranteed.

Issues related to comprehensibility have been gaining more wide-spread attention recently in the study of classification models [4, 6]. However, while these studies emphasise the need for comprehensibility, they do not offer a definitive test of the kind provided by our definition in Section 3. For classification models augmented comprehensibility of the classification model promises to impact positively on the trust users have in the model's prediction. For example, in medical decision-making and in the case of unexpected system outputs, comprehensibility generally increases the acceptance of the models by users. Finally, comprehensible models can unveil new insights about the internal structure of the data or its domain of origin.

In the context of AI testing and evaluation the importance of human comprehensibility of intelligent systems has very recently been emphasised in [3]. Forbus makes a case for AI as a research endeavour being equivalent to learning how to create smart *software social organisms* which should exhibit increasing abilities to participate in human culture and daily life. The comprehensibility of the systems behaviour and outputs is paramount in this context, since only efficient communication enables participation in human society. In [3] this is then tied into the general context of assessing the capacities of AI systems by measuring the progress which has been made, for instance, in domain generality, acquired knowledge levels, or the flexibility across different interaction modalities. When looking back at the original Turing Test [19] and present-day discussions surrounding new and updated versions or substitutes for it, comprehensibility of systems plays a crucial role. While there is frequent discussion about abandonment of the Turing Test and focusing on more clearly specified tasks in well-defined domains, putting emphasis on making systems and their output comprehensible for humans offers an alternative approach to overcoming limitations of the original test, while still maintaining domain and task generality.

2.2 Predicate Invention

Predicate Invention, the automated introduction of auxiliary predicates, has been viewed as an important problem since the early days of ILP (e.g. [10, 15, 17]), though limited progress has been made in this topic recently [13]. Early approaches were based on the use of W-operators within the inverting resolution framework (e.g., [10, 15]). However, the completeness of these approaches was never demonstrated, partly because of the lack of a declarative bias to delimit the hypothesis space. Failure to address these issues has, until recently, led to limited progress being made in this important topic and many well-known ILP systems such as ALEPH [16] and FOIL [14] have no predicate invention. In the recently introduced Meta-Interpretive Learning (MIL) framework [11, 12], predicate invention is conducted via construction of substitutions for meta-rules applied by a meta-interpreter. The use of the meta-rules clarifies the declarative bias being employed. New predicate names are introduced as higher-order skolem constants, a finite number of which are added during every iterative deepening of the search.

3 Framework

3.1 General Setting

We assume sets of constants, predicate symbols and first-order variables are denoted $\mathcal{C}, \mathcal{P}, \mathcal{V}$. We assume definite clause programs to be defined in the usual way. Furthermore we assume a human as possessing background knowledge B expressed as a definite program. We now define the distinction between private and public predicate symbols.

Definition 1. [Public and private predicate symbols]. *We say that a predicate symbol $p \in \mathcal{P}$ found in definite program P is* public *with respect to a human population S in the case that p forms part of the background knowledge of each human $s \in S$. Otherwise p is private.*

Now we define Predicate Invention as follows.

Definition 2. [Predicate Invention]. *In the case background knowledge B of an ILP is extended to $B \cup H$, where H is a definite program we call predicate symbol $p \in \mathcal{P}$ an Invention iff p is defined in H but not in B.*

3.2 Comprehensibility

Next we provide our operational definition of comprehensibility.

Definition 3. [Comprehensibility, $C(S, P)$]. *The comprehensibility of a definition (or program) P with respect to a human population S is the mean accuracy with which a human s from population S after brief study and without further sight can use P to classify new material sampled randomly from the definition's domain.*

Note that this definition allows us to define comprehensibility in a way which allows its experimental determination given a set of human participants. However, in order to clarify the term ”after brief study” we next define the notion of inspection time.

Definition 4. **[Inspection time** $T(S, P)$**].** *The inspection time T of a definition (or program) P with respect to a human population S is the mean time that a human s from S spends studying P before applying P to new material.*

Since, in the previous subsection, we assume humans as having background knowledge which is equivalent to a definite program, we next define the idea of humans mapping privately defined predicate symbols to ones found in their own background knowledge.

Definition 5. **[Predicate recognition** $R(S, p)$**].** *Predicate recognition R is the mean proportion of times that a human s from population S gives the correct public name to a predicate symbol p presented as a privately named definition q.*

For each of these mappings from privately defined predicate symbols to elements from the background knowledge we can now experimentally determine the required naming time.

Definition 6. **[Naming time** $N(S, p)$**].** *For a predicate symbol p presented as a privately named definition q in definite program P the naming time N with respect to a human population S is the mean time that a human s from S spends studying P before giving a public name to p.*

Lastly we provide a simple definition of the textual complexity of a definite program.

Definition 7. **[Textual complexity,** $Sz(P)$**].** *The textual complexity Sz of a definition of definite program P is the sum of the occurrences of predicate symbols, functions symbols and variables found in P.*

3.3 Experimental Hypotheses

We are now in a position to define and explain the motivations for the experimental hypotheses to be tested in Section 4. Below $C(S, P)$, $T(S, P)$, $R(S, p)$, $N(S, p)$, $Sz(P)$ are denoted by C, T, R, N and Sz respectively.

Hypothesis H1, $C \propto \frac{1}{T}$. This hypothesis relates to the idea of using inspection time as a proxy for incomprehension. That is, we might expect that humans take a long time to commit to an answer in the case they find the program hard to understand. As a proxy, inspection time is easier to measure than comprehension.

Hypothesis H2, $C \propto R$. This hypothesis is related to the idea that humans understand private predicate symbols, such as *p1/2*, generated during predicate invention, by mapping them to public ones in their own background knowledge.

Hypothesis H3, $C \propto \frac{1}{Sz}$. This hypothesis is motivated by the idea that a key property of predicate invention is its ability to compress a description by introducing new predicates which are used multiply within the definition. We are interested in whether the resultant compression of the description leads to increased comprehensibility.

Hypothesis H4, $R \propto \frac{1}{N}$. This hypothesis relates to the idea that if humans take a long time to recognise and publicly name a privately named predicate they are unlikely to correctly identify it. Analogous to H1, this allows naming time to be used as a proxy for recognition of an invented predicate.

In the next section we describe experiments which test these four hypotheses. Table 1 shows the mapping between the measurable properties defined in this section and the independent variables used in the experiment.

Table 1. Mapping defined properties from this section and independent variables in the experiment.

Defined property	**Experimental variable**
Comprehensibility C	Score
Inspection time T	Time
Recognition R	CorrectNaming
Naming Time N	NamingTime

4 Experiment

To investigate the hypotheses concerning comprehensibility and predicate invention, we conducted an experiment with human participants. In the following, we first present the material. Afterwards we present the independent and dependent variables and re-formulate the hypotheses with respect to these variables. Then we present the design and the results of the experiment. Finally, we relate the findings to the hypotheses of the framework.

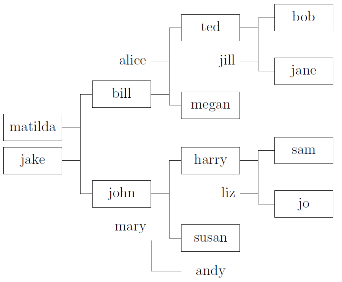
4.1 Material

Material construction is based on the well-known family tree examples used to teach Prolog [18] and also used in the context of ILP [12]. Based on the *grandparent/2* predicate, three additional problems were defined: *grandfather/2* which is more specific than *grandparent/2*, *greatgrandparent/2* which needs the double amount of rules if defined without an additional (invented) predicate, that is, which has a high textual complexity, and the recursive predicate *ancestor/2* which has small textual but high cognitive complexity [5]. Instead of these meaningful names, target predicates are called *p/2*. Given facts are identical to the family tree presented in [12].[4] In the rule bodies, either public names (*mother*, *father*)—that is, names which relate to the well-known semantics of family relations—or private names (*q1/2*, *q2/2*) were used. Furthermore, programs were either presented with or without the inclusion of an additional (invented) predicate for *parent/2* which was named *p1/2*. The trees for the public and the private name space and the predicate definitions for the public name space are given in Figure 2.

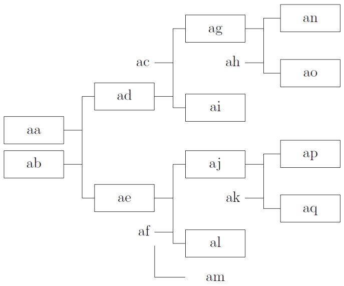
In Section 3 we defined comprehensibility of a program as the accuracy with which a human can classify new material sampled from the domain. To assess comprehensibility, we defined seven questions for each of the four predicates (see Fig. 3). For five questions, it has to be determined whether a relation for two given objects is true. For two further questions, it has to be determined for which variable bindings the relation can be fulfilled. In addition, an open question was included, where a meaningful name had to be given to predicate *p/2* for each of the four problems and—if applicable—also to the additional predicate *p1/2*.

To evaluate the material, we ran a pilot study (March 2016) at Imperial College London with 16 students of computer science with a strong background in programming, Prolog, and logic. The pilot study was conducted as a paper-and-pencil experiment where for each problem first the seven questions had to be answered and afterwards a meaningful name had to be given to the program. 13 out of the 16 students solved all questions correctly and most students were able to give the correct public names to all of the programs, regardless whether they had to work with the public or with the private names. Participants needed about 20 minutes for the four problems. Thus, the instructions and the material

[4] Please note that the relation *mother(matilda,bill)* needs to be changed to *mother(matilda,alice)* and the relation *father(jake,bill)* needs to be changed to *father(jake,alice)* to cover all cases necessary to invent the predicate parent/2 in the context of different target predicates.



```
father(jake,bill).      mother(matilda,bill).
father(jake,john).      mother(matilda,john).
father(bill,ted).       mother(alice,ted).
father(bill,megan).     mother(alice,megan).
father(john,harry).     mother(mary,harry).
father(john,susan).     mother(mary,susan).
                        mother(mary,andy).
father(ted,bob).        mother(jill,bob).
father(ted,jane).       mother(jill,jane).
father(harry,sam).      mother(liz,sam).
father(harry,jo).       mother(liz,jo).
```



```
q1(ab,ad).   q2(aa,ad).
q1(ab,ae).   q2(aa,ae).
q1(ad,ag).   q2(ac,ag).
q1(ad,ai).   q2(ac,ai).
q1(ae,aj).   q2(af,aj).
q1(ae,al).   q2(af,al).
             q2(af,am).
q1(ag,an).   q2(ah,an).
q1(ag,ao).   q2(ah,ao).
q1(aj,ap).   q2(ak,ap).
q1(aj,aq).   q2(ak,aq).
```

```
; grandfather without invented predicate
p(X,Y) :- father(X,Z), father(Z,Y).
p(X,Y) :- father(X,Z), mother(Z,Y).

; grandfather with invented predicate
p(X,Y) :- p1(X,Z), father(Z,Y).
p1(X,Y) :- father(X,Y).
p1(X,Y) :- mother(X,Y).

; grandparent without invented predicate
p(X,Y) :- father(X,Z), father(Z,Y).
p(X,Y) :- father(X,Z), mother(Z,Y).
p(X,Y) :- mother(X,Z), mother(Z,Y).
p(X,Y) :- mother(X,Z), father(Z,Y).

; grandparent with invented predicate
p(X,Y) :- p1(X,Z), p1(Z,Y).
p1(X,Y) :- father(X,Y).
p1(X,Y) :- mother(X,Y).

; ancestor without invented predicate
p(X,Y) :- father(X,Y).
p(X,Y) :- mother(X,Y).
p(X,Y) :- father(X,Z), p(Z,Y).
p(X,Y) :- mother(X,Z), p(Z,Y).
```

```
; greatgrandparent without invented predicate
p(X,Y) :- father(X,U), father(U,Z), father(Z,Y).
p(X,Y) :- father(X,U), father(U,Z), mother(Z,Y).
p(X,Y) :- father(X,U), mother(U,Z), father(Z,Y).
p(X,Y) :- father(X,U), mother(U,Z), mother(Z,Y).
p(X,Y) :- mother(X,U), father(U,Z), mother(Z,Y).
p(X,Y) :- mother(X,U), father(U,Z), father(Z,Y).
p(X,Y) :- mother(X,U), mother(U,Z), mother(Z,Y).
p(X,Y) :- mother(X,U), mother(U,Z), father(Z,Y).

; greatgrandparent with invented predicate
p(X,Y) :- p1(X,U), p1(U,Z), p1(Z,Y).
p1(X,Y) :- father(X,Y).
p1(X,Y) :- mother(X,Y).

; ancestor with invented predicate
p(X,Y) :- p1(X,Y).
p(X,Y) :- p1(X,Z), p(Z,Y).
p1(X,Y) :- father(X,Y).
p1(X,Y) :- mother(X,Y).
```

Fig. 2. Public tree (left), private tree (right) and the Prolog programs for *grandfather/2*, *grandparent/2*, *greatgrandparent/2*, and *ancestor/2* with and without use of an additional (invented) predicate *parent*. In the corresponding programs for the private name space, *father/2* is replaced by *q1/2* and *mother/2* is replaced by *q2/2*.

- What is the result of `p(bill,bob)`?
  □ true □ false □ don't know
- What is the result of `p(jake,harry)`?
  □ true □ false □ don't know
- What is the result of `p(bob,bill)`?
  □ true □ false □ don't know
- What is the result of `p(mary,jo)`?
  □ true □ false □ don't know
- What is the result of `p(john,sam)`?
  □ true □ false □ don't know
- What is the result of `p(X,bob)`?
  □ false □ X = bill □ X = alice
  □ X = bill; alice □ don't know
- What is the result of `p(john,X)`?
  □ false □ X = sam □ X = jo
  □ X = sam; jo □ don't know

Fig. 3. Questions for the *grandparent/2* problem with public names.

are understandable and coherent. A very interesting outcome of the study was that about a third of the students made notes on the questionnaires. Some of the notes showed that students first named the target predicates and the invented predicate (see Fig. 1) and then answered the questions. That is, students gave a meaningful name without being instructed to do so and one can assume that they used this strategy because it made answering the questions easier.

4.2 Variables and Empirical Hypotheses

To assess the influence of meaningful names and of predicate invention on comprehensibility, we introduced the following three independent variables:

NameSpace: The name space in which context the problems is presented is either **public** or **private** as shown in Figure 2.

PredicateInvention: The problems are given either **with** or **without** an additional (invented) predicate *p1/2* which represents the *parent/2* relation.

NamingInstruction: The open question to give a meaningful name to predicate *p/2* is either given **before** or **after** the seven questions given in Figure 3 had to be answered.

The variation of the independent variables results in a $2 \times 2 \times 2$ factor design which was realised between-participants for factors NameSpace and NamingInstruction and within-participants for factor PredicateInvention. Problem presentation with PredicateInvention was either given for the first and the third or the second and the fourth problem.

The **textual complexity** varies over problems and in dependence of the introduction of the additional predicate *p1/2*. The textually most complex program is *greatgrandparent/2* without the use of *p1/2*. The least complex program is *grandfather/2* without the use of *p1/2* as can be seen in Figure 2.

The following dependent variables were assessed:

Score: For each problem, the score is calculated as the sum of correctly answered questions (see Fig. 3). That is, score has minimal value 0 and maximal value 7 for each problem.

Time: The time to inspect a problem is measured from presenting the problem until answering the seven questions.

CorrectNaming: The correctness of the given public name for a predicate definition *p/2* was judged by two raters. In addition, it was discriminated between clearly incorrect answers and responses where participants wrote nothing or stated that they do not know the correct meaning.

NamingTime: The time for naming is measured from presenting the question until indication that the question is answered by going to the next page. For condition PredicateInvention/with both *p/2* and *p1/2* had to be named.

Given the independent and dependent variables, hypotheses can now be formulated with respect to these variables:

H1: Score is inverse proportional to Time, that is, participants who comprehend a program, give more correct answers in less time than such participants who do not comprehend the program.

H2: CorrectNaming is proportional to Score, that is, participants who can give the intended public—that is, meaningful—name to a program have higher scores than participants who do not get the meaning of the program.

H3: Score is inverse proportional to textual complexity, that is, for problem *greatgrandparent/2* the differences of score should be greatest between the PredicateInvention/with and PredicateInvention/without condition because here the difference in textual complexity is highest.

H4: CorrectNaming is inverse proportional to NamingTime, that is, if participants need a long time to come up with a meaningful name for a program, they probably will get it wrong.

4.3 Participants and Procedure

The experiment was conducted in April 2016 with cognitive science students of the University of Osnabrueck. All students had passed at least one previous one-semester course on Prolog programming and all have a background in logic. That is, their background in Prolog is less strong than for the Imperial College sample but they are no novices. From the originally 87 participants, three did not finish the experiment and six students were excluded because they answered “don’t know” for more than 50% of the questions. All analyses were done with the remaining 78 participants (43 male, 35 female; mean age 23.55 years, sd = 2.47).[5]

The experiment was realised with the `soscisurvey.de` system and was conducted online during class. After a general introduction, students worked through an example problem (“sibling”) to get acquainted with the domain—that is either the family tree or the abstract tree shown in Figure 2—and with the types of questions they needed to answer. Afterwards, the four test problems were presented in one of the eight experimental conditions. For each problem, on the first page the facts and the tree and the predicate definition was presented. On the next page, this information was given again together with the first question or the naming instruction. If the “next”-button was pressed, it was not possible to go back to a previous page.

Working through the problems was self-paced. The four problems were presented in the sequence *grandfather/2*, *grandparent/2*, *greatgrandparent/2*, *ancestor/2* for all participants. That is, we cannot control for sequence effects, such as performance gain due to getting acquainted with the style of the problems and questions or performance loss due to decrease in motivation or fatigue. However, since problem type is not used as an experimental condition, possible sequence effects do not affect statistical analyses of the effects of the independent variables introduced above.

4.4 Results

Scores and Times. When considering time for question answering and naming together, participants needed about 5 minutes for the first problem and got faster over the problems. One reason for this speed-up effect might be, that participants needed less time to inspect the tree or the facts for later problems. There is no speed-accuracy trade-off, that is, there is no systematic relation between (low) number of correct answers and (low) solution time for question answering. In the following, time is given in seconds and for statistical analyses time was logarithmically transformed.

Giving meaningful names. In the public name condition, the names the participants gave to the programs were typically the standard names, sometimes their inverse, such as “grandchildren”, “child of child”, or “parent of parent” for the *grandparent/2* problem. In the condition with private names, the standard names describing family relations were also used by most participants, however, some participants gave more abstract descriptions, such as “X and Y are connected via an internode” for *grandparent/2*. Among the incorrect answers for the *grandparent/2* problem often were over-specific interpretations such as “grandson” or “grandfather”. The same was the case for *greatgrandparent/2* with incorrect answers such as “greatgrandson”. Some participants restricted the description to the given tree, for example, “parent of parent with 2 children” for *grandparent/2*. Incorrect answers for the *ancestor/2* problem typically were overly general, such as “related”.

Impact of NameSpace, PredicateInvention, and NamingInstruction on Score and Time. An overview of the impact of all factors on score is given in Figure 4. There it can be seen that NameSpace/public results in higher scores for all four problems. The effects of PredicateInvention and NamingInstruction are less obvious. It is not the case that having to think about the meaning of a

[5] A comprehensive description of all analyses and results can be found at `http://www.cogsys.wiai.uni-bamberg.de/publications/comprAnalysesDoc.pdf`.

predicate before question answering has a general positive effect on Score. PredicateInvention is helpful for some problems, for others not. We will give a closer look on the effect of PredicateInvention for the textually most complex problem *greatgrandparent/2* below (H3). Statistical analyses were done with general linear models with NameSpace, PredicateInvention, and NamingInstruction as predictor variables and Score as criterion variable. Predictor variables were dummy coded as contrasts. The effect of NameSpace/public is significant for *grandfather/2* ($b = 1.55$, $p = 0.03$) and marginally significant for *greatgrandparent/2* ($b = 1.12$, $p = 0.069$). In addition, for *grandfather/2* the interaction of NameSpace and PredicateInvention is significant ($b = -2.52$, $p = 0.017$).

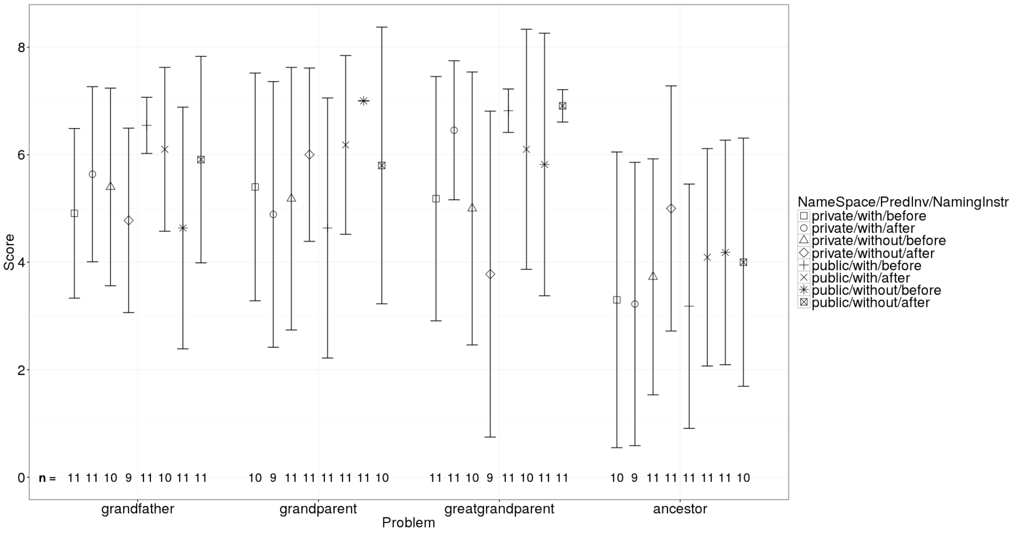


Fig. 4. Scores distributed over NameSpace, PredicateInvention, and NamingInstruction (arithmetic means and standard deviations are given; for significant differences, see text).

Inverse proportional relation between Score and Time (H1). There is a significant negative Pearsons product-moment correlation between Time and Score over all problems ($r = -.38$, $p \leq 0.001$).
Effect of CorrectNaming on Score (H2). To assess the impact of being able to give a meaningful name to a problem (CorrectNaming) on comprehensibility (Score), answers were classified as "correct", "incorrect" and "no answer" which covers answers where participants either did not answer or explicitly stated that they do not know the answer. Participants who were able to give meaningful names to the programs answered significantly more questions correctly. Statistical analyses were again performed with general linear models with dummy coding (contrast) for the predictor variable CorrectNaming. The results are given in Table 2.
Impact of textual complexity on the effect of PredicateInvention on Score (H3). For the *greatgrandparent/2* problem, there is a marginally significant effect of PredicateInvention for NameSpace/private and NamingInstruction/after with a higher score for the PredicateInvention/with condition ($b = -1.59$, $p = 0.09$).
Relation of CorrectNaming and NamingTime (H4). Participants who give a correct meaningful name to a problem do need less time to do so than participants who end up giving an incorrect name for all problems except *ancestor/2*. This relation is given in Figure 5 accumulated over all factors

Table 2. Means and standard deviations of Score in dependence of CorrectNaming, where “no answer” covers answers where participants either did not answer or explicitly stated that they do not know the answer. Results for linear models are given as b-estimates and p-values for the contrast between correct and incorrect naming.

	Correct	Incorrect	No answer	Test
Grandfather	n = 28	n = 46	n = 4	
Score	Mean 6.68 (sd = 0.61)	5.15 (1.81)	4.75 (1.71)	-1.53, $p < 0.001$
Grandparent	50	23	5	
Score	6.56 (1.23)	5.04 (2.12)	3.4 (1.82)	-1.52, $p < 0.001$
Greatgrandparent	54	18	6	
Score	6.76 (0.66)	5.78 (1.66)	3 (1.67)	-1, $p < 0.001$
Ancestor	32	39	7	
Score	5.75 (1.44)	3.08 (1.8)	2.86 (1.57)	-2.67, $p < 0.001$

per problem. Statistical analyses were done separately for conditions PredicateInvention/with and PredicateInvention/without because in the first case two names—for target predicate *p/2* and for the additional predicate *p1/2*—had to be given. Differences between correct and incorrect are significant for *grandfather/2* in the condition PredicateInvention/without ($b = 0.31$, $p = 0.007$) and marginally significant for *grandparent/2* in the condition PredicateInvention/with ($b = 0.2$, $p = 0.084$). For *ancestor/2* in the condition PredicateInvention/with there is a significant difference between correct naming and “no answer” ($b = -0.49$, $p = 0.039$).

4.5 Interpretation and Discussion

Results show that presenting programs in relation to a public name space facilitates comprehension. Contrary to our expectations, being instructed to first think about a meaningful name for a program before answering questions in general does not facilitate generation of answers. We would have expected that having a (denotational) semantic interpretation for a predicate supports working on classification and variable bindings of new material from a given domain because mental evaluation of a program can be—at least partially—avoided. Furthermore, as expected, the use of additional (invented) predicates does not facilitate program comprehension in general but only under specific conditions which are discussed below (H3).

Results concerning our hypotheses are summarised in Table 3. Hypothesis H1 is confirmed by our empirical data: if a person comprehends a program, she or he can come up with correct answers in short time. Hypothesis H2 is also confirmed: participants who can give a meaningful name to a program give more correct answers than participants who give incorrect answers or state that they do not know the answer. In addition, participants who give a correct name give answers faster. As hypothesis H3 we assumed that predicate invention supports comprehensibility if it reduces the textual complexity of a program. For the four

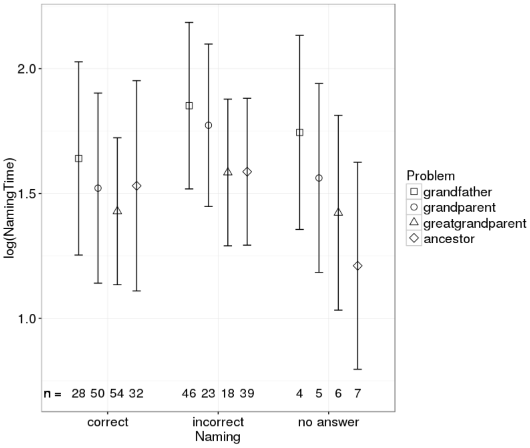


Fig. 5. Relation between time needed for giving a meaningful name and correctness of naming, where “no answer” covers answers where participants either did not answer or explicitly stated that they do not know the answer (averaged over PredicateInvention with/without).

problems we investigated, the reduction in complexity is greatest for *greatgrandparent/2*. Here we get a partial confirmation: predicate invention results in more correct answers for the private name space and if the instruction for naming was given after question answering. This experimental condition is the most challenging, because comprehensibility is not supported by public names and because participants were not encouraged to think about the meaning of the presented predicate before they had to answer questions about it.

Finally, we assumed that persons who have problems to come up with a meaningful name for a predicate spend a longer amount of time to come up with an (incorrect or no) answer (H4). Results show that this is the case—with the exception of the *ancestor/2* problem. However, the differences are only significant under specific conditions. The observation that long answering time can indicate a problem with comprehensibility could be exploited for the design of the interaction of a person with an ILP system: if a person does not come up quickly with a name for a predicate, the system could offer examples of the predicates behaviour. For example, for the *ancestor/2* problem, pairs for which this predicate is true could be highlighted in the given tree.

It can be assumed that the empirical results depend on the level of expertise of the participants. As we saw, the highly experienced sample of students of Imperial College did not profit from public name space or from the use of invented predicates. They answered most questions correctly under all conditions. In contrast, for the moderately experienced sample of students from University of Osnabrueck, presenting predicates in relation to a public name space and under some conditions with invented predicates resulted in better comprehensibility. For a sample of Prolog novices, the experimental variations might result in stronger or different effects.

Table 3. Hypotheses concerning comprehensibility, meaningful names, and predicate invention.

Hypothesis		Confirmation
H1	Comprehensibility manifests itself in high scores and fast solution times.	confirmed
H2	Comprehensibility means to be able to give a meaningful name to a program.	confirmed
H3	Predicate invention helps comprehensibility if it reduces textual complexity of the program.	partially
H4	If coming up with a meaningful name needs a long time, it will probably be the false concept.	partially

5 Conclusions and Further Work

This paper is, to our knowledge, the first paper in the literature which provides an operational definition of the comprehensibility of a logic program. The definition is used within the experiments in Section 4 to identify factors which affect comprehension. These factors include the time required to inspect the program, the accuracy with which a participant can recognise a predicate to be equivalent to one already known and the textual complexity of the program.

As expected, the four problems presented in the experiment differ with respect to comprehensibility. The problem most participants had difficulty with is the recursive *ancestor/2*. For this problem less than half of the participants (32) gave the correct meaningful name and for this problem participants have the lowest scores. However, since this problem was positioned last in the sequence, the results might also be due to loss of motivation or exhaustion. Astonishingly, *ancestor/2* is also the only of the four problems where participants reached the highest score in the private naming condition without predicate invention (cf. Fig. 4). We plan a follow-up experiment where problem sequences are varied to determine whether this is a systematic effect.

The kinship predicates presented to human participants in our experiments are all ones which could be expected to be equivalent to ones already known to the participant. In further work we hope also to study the effects of human users being presented with definitions of predicates which are novel for the user.

In closing we believe the operational definition of comprehensibility has enormous potential to both clarify one of the central concepts of AI research, as well as to provide a bridge to the study of factors affecting the design of AI systems which improve human understanding.

References

1. A. Cropper and S.H. Muggleton. Learning efficient logical robot strategies involving composable objects. In *Proceedings of the 24th International Joint Conference Artificial Intelligence (IJCAI 2015)*, pages 3423–3429. IJCAI, 2015.
2. A. Cropper and S.H. Muggleton. Learning higher-order logic programs through abstraction and invention. In *Proceedings of the 25th International Joint Conference Artificial Intelligence (IJCAI 2016)*, pages 1418–1424. IJCAI, 2016.
3. K. D. Forbus. Software social organisms: Implications for measuring AI progress. *AI Magazine*, 37(1), 2016.
4. A. A. Freitas. Comprehensible classification models: A position paper. *SIGKDD Explor. Newsl.*, 15(1):1–10, March 2014.
5. H. Kahney. What do novice programmers know about recursion? In E. Soloway and J. C. Spohrer, editors, *Studying the Novice Programmer*, pages 209–228. Lawrence Erlbaum, 1989.
6. B. Letham, C. Rudin, T. H. McCormick, and D. Madigan. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *Annals of Applied Statistics*, 9(3):1350–1371, September 2015.
7. D. Lin, E. Dechter, K. Ellis, J.B. Tenenbaum, and S.H. Muggleton. Bias reformulation for one-shot function induction. In *Proceedings of the 23rd European Conference on Artificial Intelligence (ECAI 2014)*, pages 525–530. IOS Press, 2014.
8. D. Michie. Machine learning in the next five years. In *Proceedings of the Third European Working Session on Learning*, pages 107–122. Pitman, 1988.
9. M. Mozina, J. Zabkar, and I. Bratko. Argument based machine learning. *Artificial Intelligence*, 171(10–15):922 – 937, 2007.
10. S.H. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning*, pages 339–352. Kaufmann, 1988.
11. S.H. Muggleton, D. Lin, N. Pahlavi, and A. Tamaddoni-Nezhad. Meta-interpretive learning: Application to grammatical inference. *Machine Learning*, 94:25–49, 2014.
12. S.H. Muggleton, D. Lin, and A. Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Machine Learning*, 100(1):49–73, 2015.
13. S.H. Muggleton, L. De Raedt, D. Poole, I. Bratko, P. Flach, and K. Inoue. ILP turns 20: Biography and future challenges. *Machine Learning*, 86(1):3–23, 2011.
14. J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.
15. C. Rouveirol and J-F. Puget. A simple and general solution for inverting resolution. In *Proceedings of the fourth European Working Session on Learning (EWSL-89)*, pages 201–210. Pitman, 1989.
16. A. Srinivasan. The ALEPH manual. *Machine Learning at the Computing Laboratory, Oxford University*, 2001.
17. I. Stahl. Constructive induction in inductive logic programming: An overview. Technical report, Fakultät Informatik, Universität Stuttgart, 1992.
18. L. Sterling and E. Y. Shapiro. *The art of Prolog: advanced programming techniques.* MIT Press, 1994.
19. A. M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.