Calhoun: The NPS Institutional Archive
DSpace Repository

Faculty and Researchers

Faculty and Researchers' Publications

1999-11

A Computer Tutor for Logic Semantics

Rowe, Neil C.

Monterey, California. Naval Postgraduate School

IEEE Frontiers in Education Conference, San Juan, PR, November 10-13, 1999
<https://hdl.handle.net/10945/35971>

Downloaded from NPS Archive: Calhoun

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

<http://www.nps.edu/library>

A Computer Tutor for Logic Semantics

Neil C. Rowe

Code CS/Rp, U. S. Naval Postgraduate School

Monterey, CA 93943 USA

(831)656-2462, rowe@cs.nps.navy.mil

Abstract

We report on a computer tutor for the semantics of the Prolog subset of predicate calculus. It gives students statements in English to represent in a single line of Prolog, parses their answers, and compares the parses to the parses of correct answers. The tutor focuses on the correct choice of predicates, variables, and links between expressions. Tests show that students learn predicate calculus significantly better using the tutor than with paper and pencil exercises.

This paper appeared in the IEEE Frontiers in Education Conference, San Juan, PR, November 10-13, 1999.

1. Introduction

A variety of computer software helps students learn Boolean algebra and predicate calculus. But these tools heavily emphasize logical correctness instead of semantics (how the notation relates to the real world), and they often do an inadequate job of understanding the reasons that student make mistakes. Several tutors like [3] provide an introduction to digital electronics including what they call "combinatorial logic", but this is basically low-level circuit design using only Boolean algebra. Other tutors like [5] and [6] are designed for mathematics and philosophy courses and provide proof-checkers for predicate-calculus proofs constructed by students, but neither the exercises nor the skills taught themselves have much real-world relevance. An innovative and different tutor is Tarski's World [2] where semantics is provided by an artificial world of ideal shapes in three dimensions, which students can query and manipulate using logical expressions. This is likely the first logic tutor with a significant semantic component, but it may be hard for students to relate lessons involving cubes and spheres to useful real-world problems where logical representations are necessary.

We have developed a different kind of computer tutor LOGTUTOR for the semantics of predicate calculus. This intelligent tutor is useful for courses in introductory switching theory, applied logic, and artificial intelligence. By "intelligent" we mean that the tutor reasons about why the student got the wrong answer and tutors appropriately, as in other "intelligent tutoring systems" [1, 4, 7], rather than just complaining to the student. The tutor requires the student to provide answers in the syntax of the programming language Prolog, and teaches the subset of predicate calculus supported directly in the syntax of that language [8]. However, it is not truly a programming language tutor since answers must be confined to one line (hence prohibiting loops and most programming constructs), and many student errors are not in syntax or structure but in the relating of symbols to the real world.

2. Parsing and interpretation of student input

The tutor is itself written in Quintus Prolog and its source code contains 30,000 symbols. It assigns random problems to students for fourteen modules of six problems each. Each problem asks the student to construct a logical expression. The expression must be written in the syntax of the Prolog to permit compatibility with later programming exercises in the course in which the tutor is used. The student's input is checked for lexical errors (impermissible characters, unmatched parentheses, and incorrect symbols) and chunked. If any lexical errors are simple enough to be quite certain of the correction, that correction is made. Then if no fatal errors remain, the chunked input is parsed with an augmented-transition network parser. Syntax errors are noted, only some of which are fatal. Parsing also assigns semantics (meanings) to the parts of the student input.

Then the parse of the student's input is compared to a parse of a supplied correct answer (not necessarily the only correct answer),

constructed analogously to the comparison of procedures in the METUTOR procedural tutoring systems [9]. The two answers are recursively parsed and their parts matched. Since the student chooses their own variable names, one of the main challenges is matching the student's variables to the tutor's variables, which can require significant combinatorial search even when the student's answer is correct. A rule-based system with 308 high-level rules analyzes the differences between the parses and assigns the errors to one or more of 156 different types. The types emphasize semantic distinctions like those between facts and implications, relationships and properties, symptoms and diagnoses, premises and conclusions, predicates and arguments, necessary variables and constants, negative constants and explicit negation, and metalogical and logical terms. They also cover incorrect argument types, missing conditional probabilities, useless constants, variables unconnected to any others, expressions combined into one, and nonstandard forms of recursion. Specific tutoring strategies are then invoked appropriate to each type. The student must redo the problem until they supply an acceptable answer before they can proceed to the next question. Students can ask for hints at any time, for which they get told the correct predicate names and the number of predicate expressions required.

3. Tutoring methods

LOGTUTOR tries to handle all the major errors that college students make in its subset of predicate calculus. Error types have been formulated after studying runs with actual students in a course setting. To handle the variety of errors, the tutor has extensive "mal-rules" [10] modeling observed incorrect ways of solving problems. For instance, one mal-rule says that if the student input has one more expression than the tutor solution, and the student has an assertion of a variable type not in the tutor solution, and you can replace the variable by the type name in the student input to get something that better matches the tutor input, then the student has an unnecessary variable and unnecessary type assertion.

When a student error is found, the student input is corrected if possible and analysis proceeds in an attempt to find further errors. For instance, students often reverse implications and reverse arguments, so reversals are checked in the process of trying to match student input and the correct answer; if found, the reversal is undone, and parsing and tutoring continues with the modified student input. Similar modification of the student's answer to a canonical form is done for logically equivalent forms such as commutations of conjunctions and disjunctions (but warning students when they violate Prolog's restrictions on commutivity since this matters later in the course). Significant backtracking and test bindings of variables can be required to match the student's and tutor's parses to the appropriate tutoring rule since there are many very-similar errors, but it can all be done within less than a second of real time since one-line student answers are required.

4. Record-keeping

Detailed records and statistics are recorded on the performance of each student, and a score is assigned to each student on a nonlinear sigmoid scale (with hints entailing a small penalty), so that additional errors matter less when a student has very few or very many errors.

Overall statistics on errors are computed to show the instructor where guidance is needed. In the most recent run of 24 students (the third class of students to use the tutor) through seven tutor modules of eleven problems each, the following counts were recorded on error types occurring five times or more:

```
212-missing period
216-incorrect predicate expression
102-misspelling
95-arguments reversed
80-symptom not diagnosis
75-spaces needed around implication
75-plural used instead of singular
68-wrongly uncapitalized
63-typed variable needed
65-incorrect second argument
58-wrong conclusion
56-incorrect first argument
39-rule confused with fact
45-wrong conclusion predicate
43-constant not specific
```

36-constant not variable
31-invalid predicate name
31-one-argument predicate inadequate
30-rule reversed
29-mischosen predicate name: owns
28-conclusion too specific
27-unused probability variable
26-unnecessary typed variable
26-invalid argument list
23-variable not constant
23-wrong inheritance format
22-illegal control character
22-diagnosis not symptom
20-first argument must be list
20-new variable needed
20-missing expressions
19-inverse number used
17-"append" bindings wrong
15-mischosen predicate name: part_of
14-decimal needs a first zero
14-unnecessary variable
13-wrong number
13-hyphen for underscore
13-more than two arguments
12-constant needs to be split
12-missing rule strength
11-fact confused with query
11-second argument must be list
11-negative constant for probability
11-first argument cannot be list
11-wrongly capitalized
10-bar symbol cannot be used
10-missing negation
10-needs three arguments
9-mischosen predicate name: a_kind_of
9-gap between predicate and arguments 7-fact not rule
7-negation predicate not allowed
7-inconsistent argument position
7-invalid character
7-avoid negative constants
7-extra expression
7-parenthesized expression
6-wrong combination
6-extra expressions
6-second argument must be a list
6-product not "andcombine"
6-missing probability combination
6-missing right parenthesis
5-multiple periods
5-"append" needs a variable
5-misused apostrophe

These errors are not mutually exclusive – some like “Incorrect predicate expression” and “Wrong conclusion” are general diagnostics given also when more specific errors are recognized. And different errors can be found at different places in the same student input. On the average, an incorrect student input generated two diagnosis messages.

Note from the statistics that while a number of syntactic errors were common, most of the errors were semantic. The students using the tutor during these runs were mostly adults in the third quarter of a Computer Science M.S. degree program, so they already had some programming skills and were less prone to syntax errors than beginning programmers. But even so, semantic errors initiate most of the key tutoring. “Plural used instead of singular” is an example, important because we want students to reason in terms of types rather than sets. Similarly, “Constant not specific” usually signals a student failure to represent all of the given problem. Even many of the seemingly syntactic errors listed above are tutored as semantic errors, as for instance capitalization errors because Prolog notation assigns capitalized words to variables and uncapitalized words to constants.

5. Testing

Tests show that students learn significantly better using the tutor than with traditional paper-and-pencil exercises. Scores on the subsequent test focusing mostly on predicate calculus went from an average of 75.8% for 93 students in 1995-1997 to 87.7% for 58 students in 1998. Alternative explanations are that students were drilled more using the tutor than they could be with paper-and-pencil exercises, and there is undoubtedly a novelty effect in using the tutor, but the improvement is still considerable. Students still occasionally get frustrated when they cannot find a solution even with hints; the continued addition of mal-rules should reduce this.

6. Demonstration

The best way to understand the tutor is to see a demonstration. First we show the introductory material printed when a student logs in to the tutor.

*You are to answer each problem with a single line of Prolog.
You should use only the words given and their grammatical variants.
Don't volunteer conditions not explicitly mentioned.
You may use parentheses only for arguments, not expressions.*

Only the following predicate expressions may be used (with further restrictions for each problem set):

*a_kind_of(X,Y): X is a kind of Y
part_of(X,Y): X is part of Y
before(X,Y): Event X happened before event Y
owns(X,Y): X (a person or organization) possesses
 object Y
color(X,Y): Object X is of color Y
material(X,Y): Object X is comprised of material Y
rating(X,Y): Object X's evaluation or status is Y
courses(D,L): L is the list of course numbers (integers) in department with code D
member(X,L): X is a member of list L
append(L1,L2,L): L is the result of appending lists L1
 and L2
delete(X,L,NL): NL is the result of deleting X from list L
length(L,N): List L has N items in number
symptom(X,P): Symptom X occurred with probability P
diagnosis(X,P): Diagnosis X is likely with probability P
inverse(X,Y): Y is I-X
product(L,X): X is the product of the numbers in the list L
andcombine(L,X): X is the andcombine of the
 probabilities in the list L
orcombine(L,X): X is the orcombine of the probabilities
 in the list L*

Here are excerpts from actual student interactions, with some lines broken to enhance readability. The “?-“ and “|:” precede lines of student input. The “:-“ symbol represents a right-to-left implication symbol and can be read as “if”. The “\+” symbol represents negation. Parentheses delimit arguments to predicates, and square brackets delimit singly-linked lists. Commas separate arguments,

and also mean “and” when used between expressions; semicolons mean “or”. Capitalized words are variables.

The first example shows a student learning how to split a composite concept into logical pieces. After they realized that “Tom” and “car” had to be separated into separate assertions of ownership and color, they then had to see the need for a variable for the car because the statement is not talking about all cars in the world.

Write as a query: "What color is Tom's car?"

(Allowed predicates: [a_kind_of, part_of, before, owns, color, material]).

Type your solution (or "info", "hint", "quit"):

?- color(toms_car, X).

Constant should be split into pieces: toms_car

Try again.

?- color(car, tom), color(car, X).

"color" is used inappropriately: color(car, tom)

Try again.

?- color(car, X), owns(tom, X).

A variable here should be a constant: owns(tom, X)

Something is wrong in this expression: color(car, X)

Try again.

?- color(car, C), owns(tom, car).

You need an a_kind_of with variable as first arg and this second arg: car

Try again.

?- color(X, C), owns(tom, X), a_kind_of(X, car).

I accept that (with any corrections mentioned).

The next example shows tutoring of an inference rule. It shows a student choosing incorrect predicate names and trying to do the implication backward. They then needed to express negation logically and introduce a new variable.

Note that the conclusion (left side) of the inference rule is always checked first, because unless it makes sense, the rest of the rule will not make sense.

Write as a rule:

"Anything is good that has no parts."

(Allowed predicates: [a_kind_of, part_of, before, owns, color, rating]).

Type your solution (or "info", "hint", "quit"):

|: owns(X, no_parts) : - rating(X, good).

The predicate name for the conclusion is wrong: owns

Try again.

|: part_of(X, no_parts) : - rating(X, good).

The predicate name for the conclusion is wrong: part_of

Try again.

|: rating(X, good) : - part_of(X, no_parts).

Left side looks acceptable....

This negative constant should be done by an explicit negation: no_parts

Try again.

|: rating(X, good) : - \+ part_of(X, part).

Left side looks acceptable....

You need a new variable at one place where you reuse this: X

Try again.

|: rating(X, good) : - \+ part_of(X, Part).

Left side looks acceptable....

Arguments should be reversed: part_of(X, Part)

I accept that (with any corrections mentioned).

This example shows a student omitting a variable and then using an unnecessary one through a capitalization error.

Write as a rule:

"An aircraft owned by NPS is owned by the Navy."

(Allowed predicates: [a_kind_of, part_of, before, owns, color, rating]).

Type your solution (or "info", "hint", "quit"):

|: owns(navy, X) : - owns(nps, aircraft).

Left side looks acceptable....

*You need an a_kind_of with variable as first arg and this
second arg: aircraft*

Try again.

|: owns(navy, X) : - owns(NPS, X),

a_kind_of(X, aircraft).

Left side looks acceptable....

Incorrect capitalization: NPS

I accept that (with any corrections mentioned).

This shows the special handling of inheritance rules, which are tricky for students to write. Particular features incompatible with inheritance rules are flagged in the student input. We need a number of explicit mal-rules to handle all the possible mistaken inheritance forms.

Write as a rule:

*"Something owned by something is also owned
by any larger something of which it is part."*

*(Allowed predicates: [a_kind_of, part_of, before, owns,
color, rating]).*

Type your solution (or "info", "hint", "quit"):

|: owns(X, T) : - part_of(T, T2), owns(X, T2).

Not the proper form needed for an inheritance rule: part_of(T, T2)

Not the proper form needed for an inheritance rule: owns(X, T2)

Left side looks acceptable....

*You need a new variable at one place where you reuse
this: T*

Try again.

|: owns(X, T) : - part_of(X, Y), owns(Y, T2).

Not the proper form needed for an inheritance rule: owns(Y, T2)

Left side looks acceptable....

Arguments should be reversed: part_of(X, Y)

The second argument here is wrong: owns(Y, T2)

Try again.

|: owns(X, T) : - part_of(X, Y), owns(Y, T).

Left side looks acceptable....

Arguments should be reversed: part_of(X, Y)

I accept that (with any corrections mentioned).

This example shows handling of list-processing predicates. The tutor checks type consistency of each argument to these specialized predicates and comments on the most obvious problems first. Tutoring of list processing is complex because there are many equivalent forms.

Write as a query:

"Is 3600 immediately followed by 3605 in the CS listing"

(Hint: split the list into three parts)?"

(Allowed predicates: [append, member, delete, length, courses]).

Type your solution (or "info", "hint", "quit"):

?- courses(cs, L), append(L1, 3600, 3605).

The second argument must be a bound list:

append(L1, 3600, 3605)

Try again.

?- courses(cs, L), append(L1, L2, L),

append(L3, 3600, L).

The second argument must be a bound list:

append(L3, 3600, L)

Try again.

?- courses(cs, L), append(L1, L2, L),

append(L, [3600, 3605], L3).

You put lists together when you should split a list: append(L, [3600, 3605], L3)

Try again.

?- courses(cs, L), append(L1, L2, L),

append(L, [3600, 3605], L2).

You need at least one new variable here:

append(L, [3600, 3605], L2)

Try again.

?- courses(cs, L), append(L1, L2, L),

append(L4, [3600, 3605], L2).

First two arguments should be reversed:

append(L4, [3600, 3605], L2)

I accept that (with any corrections mentioned).

This example shows handling of expressions with probabilities. The student made a few careless errors and forgot how to do negation with probabilities; they also needed to write a more general rule than they originally envisioned.

Write as a rule with a probability variable in every expression:

"There is a threat with 95% certainty if it is approaching and is not identified (assume evidence independence)."

(Allowed predicates: [diagnosis, symptom, andcombine, orcombine, product, inverse]).

Type your solution (or "info", "hint", "quit"):

! : diagnosis(threat, 0.95) : -

diagnosis(approaching, P1), diagnosis(identified, P2),

andcombine([P1, P2], P).

The conclusion predicate expression needs one or more variables: diagnosis(threat, 0.95)

Try again.

! : diagnosis(threat, P) : - diagnosis(approaching, P1),

diagnosis(identified, P2), andcombine([P1, P2], P).

Left side looks acceptable....

This should be a symptom: diagnosis(approaching, P1)

This should be a symptom:

diagnosis(identified, P2)

Something is wrong in this expression:

andcombine([P1, P2], P)

You must invert a probability before combining: [andcombine([P1, P2], P)]

Try again.

|: *diagnosis(threat, P) :- symptom(approaching, P1),
not(symptom(identified, P2)),
andcombine([P1, P2], P).*
Use the \+ construct instead of a "not" predicate: *not(symptom(identified, P2))*
Left side looks acceptable....
Negation with probabilities needs "inverse", not "not": *not(symptom(identified, P2))*
Try again.
|: *diagnosis(threat, P) :- symptom(approaching, P1),
symptom(identified, P2), NP2 is 1-P2,
andcombine([P1, NP2], P).*
Left side looks acceptable....
You forgot the rule strength:
andcombine([P1, NP2], P)
Try again.
|: *diagnosis(threat, P) :- symptom(approaching, P1),
symptom(identified, P2), NP2 is 1-P2,
andcombine([P1, NP2, 0.95], P).*
Left side looks acceptable....
I accept that (with any corrections mentioned).

7. References

- [1] Anderson, J., Corbett, A., Koedinger, K., and Pelletier, R., "Cognitive Tutors: Lessons Learned." *The Journal of the Learning Sciences*, 4 (2), 1995, 162-207.
- [2] Barwise, J. and Etchemendy, J., *Tarski's World Version 4.0*, software, CSLI Publications, Cambridge University Press, 1994.
- [3] Foster, G., *Digital Logic Tutor 1: An Introduction to Combinatorial Logic*, software, Prentice-Hall, Englewood Cliffs, NJ, 1993.
- [4] McCalla, G., Greer, J., and the SCENT Research Team, "SCENT-3: An Architecture for Intelligent Advising in Problem Solving Domains." In *Intelligent Tutoring Systems: At the Crossroad of Artificial Intelligence and Education*, Ablex Publishing, Norwood, NJ, 1990.
- [5] Pole, N., *LogicCoach*, software, Wadsworth Publishing, 1997.
- [6] Portoraro, F., and Tulley, R., *Logic with Symlog: Learning Symbolic Logic by Computer*, software, Prentice-Hall, Englewood Cliffs, NJ, 1994.
- [7] Psotka, J., Massey, L., and Mutter, S., *Intelligent Tutoring Systems: Lessons Learned*. Lawrence Erlbaum, Hillsdale, NJ, 1987.
- [8] Rowe, N., *Artificial Intelligence through Prolog*. Prentice-Hall, Englewood Cliffs, NJ, 1988.
- [9] Rowe, N. and Galvin, T., "An Authoring System for Intelligent Tutors for Procedural Skills." *IEEE Intelligent Systems*, 13, 3, May/June 1998, pp. 61-69.
- [10] Sleeman, D., "PIXIE: A Shell for Developing Intelligent Tutoring Systems." In *Artificial Intelligence in Education*, volume 1, Ablex, Norwood NJ, 1987, pp. 239-265.