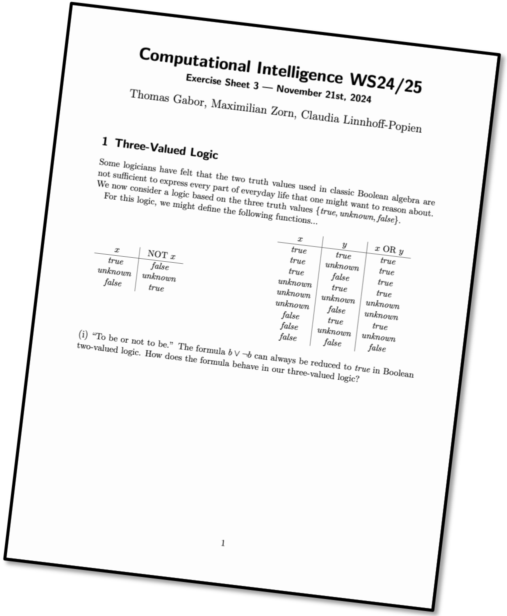
Computational Intelligence

LMU
LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

LMU Munich
winter term 2024/2025

Thomas Gabor
Claudia Linnhoff-Popien

writing exercise
2024-11-21



Computational Intelligence WS24/25

Exercise Sheet 3 — November 21st, 2024

Thomas Gabor, Maximilian Zorn, Claudia Linnhoff-Popien

1 Three-Valued Logic

Some logicians have felt that the two truth values used in classic Boolean algebra are not sufficient to express every part of everyday life that one might want to reason about. We now consider a logic based on the three truth values {*true*, *unknown*, *false*}.

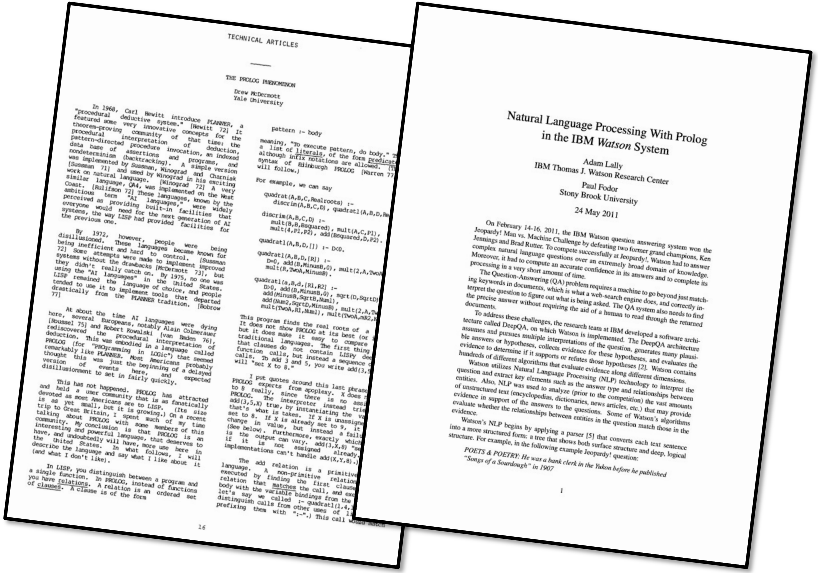
For this logic, we might define the following functions...

x	NOT x
true	*false*
unknown	*unknown*
false	*true*

x	y	x OR y
true	*true*	*true*
true	*unknown*	*true*
true	*false*	*true*
unknown	*true*	*true*
unknown	*unknown*	*unknown*
unknown	*false*	*unknown*
false	*true*	*true*
false	*unknown*	*unknown*
false	*false*	*false*

(i) "To be or not to be." The formula $b \vee \neg b$ can always be reduced to *true* in Boolean two-valued logic. How does the formula behave in our three-valued logic?

reading exercise
2024-11-28



THE PROLOG PHENOMENON

Drew McDermott
Yale University

In 1968, Carl Hewitt introduce PLANNER, a "procedural deductive system." [Hewitt 72] It featured some very innovative concepts for the theorem-proving community of that time: the procedural interpretation of deduction, pattern-directed procedure invocation, an indexed data base of assertions and programs, and nondeterminism (backtracking). A simple version was implemented by Sussman, Winograd and Charniak [Sussman 71] and used by Winograd in his exciting work on natural language. [Winograd 72] A very similar language, QA4, was implemented on the West Coast. [Rulifson 72] These languages, known by the ambitious term "AI languages," were widely perceived as providing built-in facilities that everyone would need for the next generation of AI systems, the way LISP had provided facilities for the previous one.

By 1972, however, people were being disillusioned. These languages became known for being inefficient and hard to control. [Sussman 72] Some attempts were made to implement improved systems without the drawbacks [McDermott 73], but they didn't really catch on. By 1975, no one was using the "AI languages" in the United States. LISP remained the language of choice, and people tended to use it to implement tools that departed drastically from the PLANNER tradition. [Bobrow 77]

At about the time AI languages were dying here, several Europeans, notably Alain Colmerauer [Roussel 75] and Robert Kowalski [van Emden 76], rediscovered the procedural interpretation of deduction. This was embodied in a language called PROLOG (for "PROgramming in LOGic") that seemed remarkably like PLANNER. Most Americans probably thought this was just the beginning of a delayed version of events here, and expected disillusionment to set in fairly quickly.

This has not happened. PROLOG has attracted and held a user community that is as fanatically devoted as most Americans are to LISP. (Its size is as yet small, but it is growing.) On a recent trip to Great Britain, I spent much of my time talking about PROLOG with some members of this community. My conclusion is that PROLOG is an interesting and powerful language, that deserves to have, and undoubtedly will have, more use here in the United States. In what follows, I will describe the language and say what I like about it (and what I don't like).

In LISP, you distinguish between a program and a single function. In PROLOG, instead of functions you have relations. A relation is an ordered set of clauses. A clause is of the form

```
pattern :- body
```

meaning, "To execute pattern, do body." T
a list of literals, of the form predicat
although infix notations are allowed. (T
syntax of Edinburgh PROLOG [Warren 77
will follow.)

For example, we can say

```
quadrat(A,B,C,Realroots) :-
   discrim(A,B,C,D), quadratl(A,B,D,Re

discrim(A,B,C,D) :-
   mult(B,B,Bsquared), mult(A,C,P1),
   mult(4,P1,P2), add(Bsquared,D,P2).

quadratl(A,B,D,[]) :- D<0.

quadratl(A,B,D,[R]) :-
   D=0, add(B,MinusB,0), mult(2,A,TwoA
   mult(R,TwoA,MinusB).

quadratl(a,B,d,[R1,R2] :-
   D>0, add(B,MinusB,0), sqrt(D,SqrtD)
   add(MinusB,SqrtB,Num1),
   add(Num2,SqrtD,MinusB), mult(2,A,Tw
   mult(TwoA,R1,Num1), mult(TwoA,mR2,N
```

This program finds the real roots of a
It does not show PROLOG at its best (or i
but it does make it easy to compare
traditional languages. The first thing
that clauses do not contain LISPy dee
function calls, but instead a sequence o
calls. To add 3 and 5, you write add(3,5
will "set X to 8."

I put quotes around this last phrase
PROLOG experts from apoplexy. X does n
to 8 really, since there is no assi
PROLOG. The interpreter instead trie
add(3,5,X) true, by instantiating the va
that's what is takes. If X is unassigne
set to 8. If X is already set to 9, it
change in value, but instead a failu
(See below). Furthermore, exactly which
is the output can vary. add(3,X,8) "se
if it is not assigned already.
implementations can't handle add(X,Y,8).)

The add relation is a primitive
language. A non-primitive relation
executed by finding the first clause
relation that matches the call, and exe
body with the variable bindings from the
let's say we called :- quadratl(1,4,1
distinguish calls from other uses of li
prefixing them with ":-".) This call would match

Natural Language Processing With Prolog in the IBM *Watson* System

Adam Lally
IBM Thomas J. Watson Research Center

Paul Fodor
Stony Brook University

24 May 2011

On February 14-16, 2011, the IBM Watson question answering system won the Jeopardy! Man vs. Machine Challenge by defeating two former grand champions, Ken Jennings and Brad Rutter. To compete successfully at Jeopardy!, Watson had to answer complex natural language questions over an extremely broad domain of knowledge. Moreover, it had to compute an accurate confidence in its answers and to complete its processing in a very short amount of time.

The Question-Answering (QA) problem requires a machine to go beyond just matching keywords in documents, which is what a web-search engine does, and correctly interpret the question to figure out what is being asked. The QA system also needs to find the precise answer without requiring the aid of a human to read through the returned documents.

To address these challenges, the research team at IBM developed a software architecture called DeepQA, on which Watson is implemented. The DeepQA architecture assumes and pursues multiple interpretations of the question, generates many plausible answers or hypotheses, collects evidence for these hypotheses, and evaluates the evidence to determine if it supports or refutes those hypotheses [2]. Watson contains hundreds of different algorithms that evaluate evidence along different dimensions.

Watson utilizes Natural Language Processing (NLP) technology to interpret the question and extract key elements such as the answer type and relationships between entities. Also, NLP was used to analyze (prior to the competition) the vast amounts of unstructured text (encyclopedias, dictionaries, news articles, etc.) that may provide evidence in support of the answers to the questions. Some of Watson's algorithms evaluate whether the relationships between entities in the question match those in the evidence.

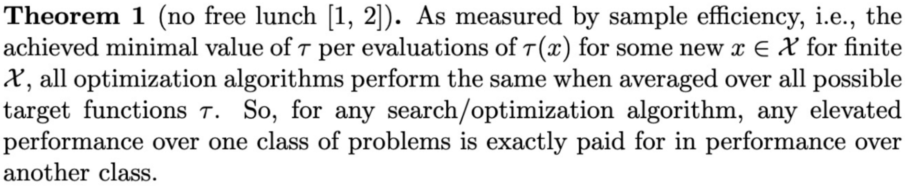
Watson's NLP begins by applying a parser [5] that converts each text sentence into a more structured form: a tree that shows both surface structure and deep, logical structure. For example, in the following example Jeopardy! question:

POETS & POETRY: He was a bank clerk in the Yukon before he published "Songs of a Sourdough" in 1907

Which optimization algorithm is the best?

Exploration/Exploitation Dilemma

The No Free Lunch Theorem



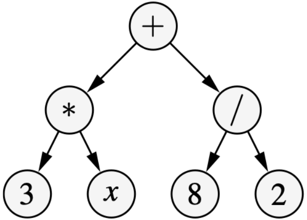
Theorem 1 (no free lunch [1, 2])**.** As measured by sample efficiency, i.e., the achieved minimal value of τ per evaluations of $\tau(x)$ for some new $x \in \mathcal{X}$ for finite $\mathcal{X}$, all optimization algorithms perform the same when averaged over all possible target functions τ. So, for any search/optimization algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class.

encoding policies…

What do we need
from policy encoding
to run
optimization algorithms?

Genetic Programming

Fig. 13.8 Parse tree of the symbolic expression $(+\ (*\ 3\ x)\ (/\ 8\ 2))$



source: Kruse et al. Computational Intelligence. 2022

Genetic Programming (Genetic Operators)

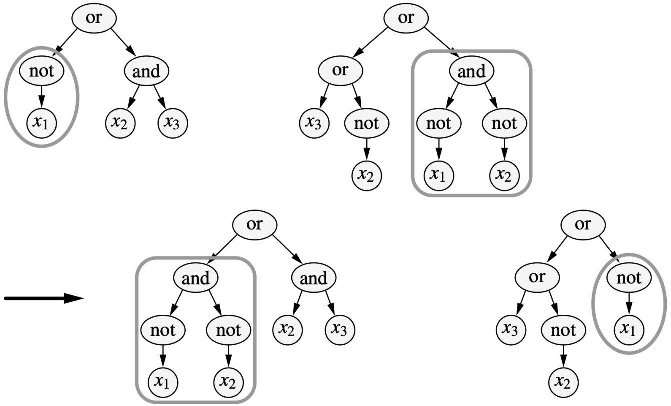


Fig. 13.9 Crossover of two sub-expressions or sub-trees

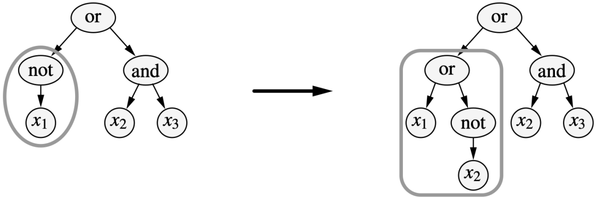
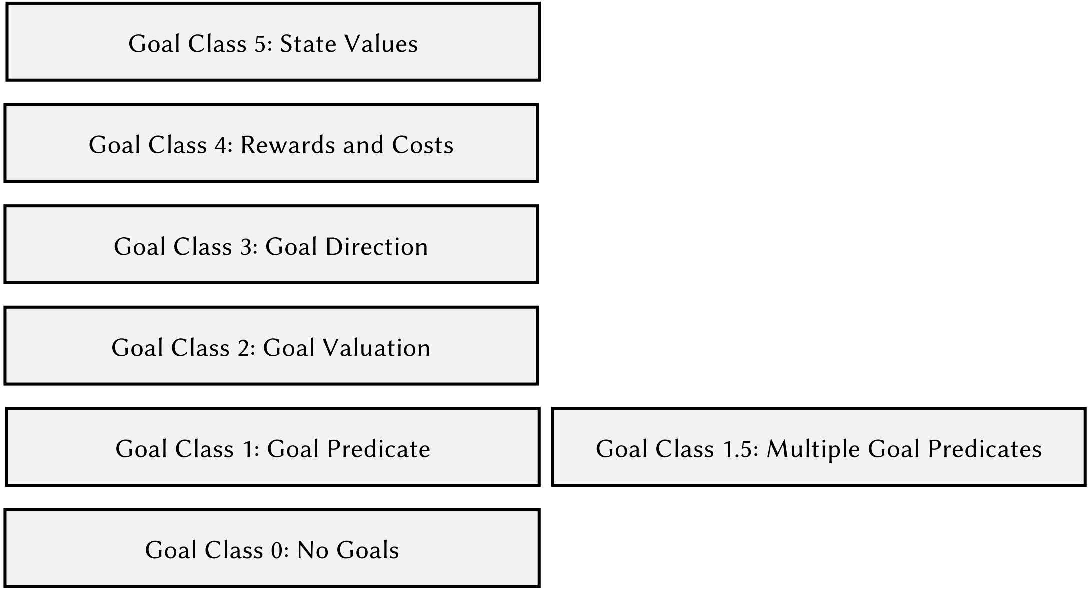


Fig. 13.10 Mutation of a sub-expression or sub-tree

source: Kruse et al. Computational Intelligence. 2022

The Goal Class Hierarchy



Goal Class 5: State Values

Goal Class 4: Rewards and Costs

Goal Class 3: Goal Direction

Goal Class 2: Goal Valuation

Goal Class 1: Goal Predicate

Goal Class 1.5: Multiple Goal Predicates

Goal Class 0: No Goals

Goal Class 2.5: Multiple Goal Valuation

“I know how good it is when I see it —
but I have have multiple criteria!”

Multi-Objective Optimization

source: Kruse et al. Computational Intelligence. 2022