An 11th century proof search algorithm

Wilfrid Hodges
Okehampton, England
July 2013
http://wilfridhodges.co.uk

Al-Khwārizmī, early 9th century,
gave algorithms their name but not their definition:

Al-Khwārizmī
⇓
algorismus
⇓
algorithm

Al-Khwārizmī's algorithms were uniform deterministic procedures for solving any equations within a given class.

As Roshdi Rashed has emphasised,
Al-Khwārizmī not only described the procedures;
he proved their correctness (i.e. that they give correct solutions for all equations in the class).
For this he used Euclid's geometry.

Al-Khalīl, the inventor of dictionaries (8th century),
had already given some *enumeration algorithms*,
i.e. uniform deterministic procedures for listing the elements of any set in a given class of sets,
but without correctness proofs.

Typical example: lexicographic ordering of the finite strings of symbols from an ordered set of symbols.
(Well known to later Arabic mathematicians.)

Ibn Sīnā (Avicenna) in the 1020s gave what may be the first *search algorithm*: a uniform procedure for finding objects of a certain kind within sets of a certain kind.

Typical modern example: *Dijkstra's algorithm* for finding shortest path between two nodes in a weighted graph.

Ibn Sīnā's algorithm was a *proof search algorithm*,
for finding a complete formal proof,
given an incomplete proof of the same conclusion.
His logical calculus was compound syllogisms.
No other non-trivial proof search algorithms are known before the 20th century.

In a sense, proof search algorithms are the most general kind of search algorithm.
The programming language PROLOG searches for an object O by searching for a proof that O exists.
It uses the logic of Horn clauses,
a fragment of first-order logic that includes syllogisms.

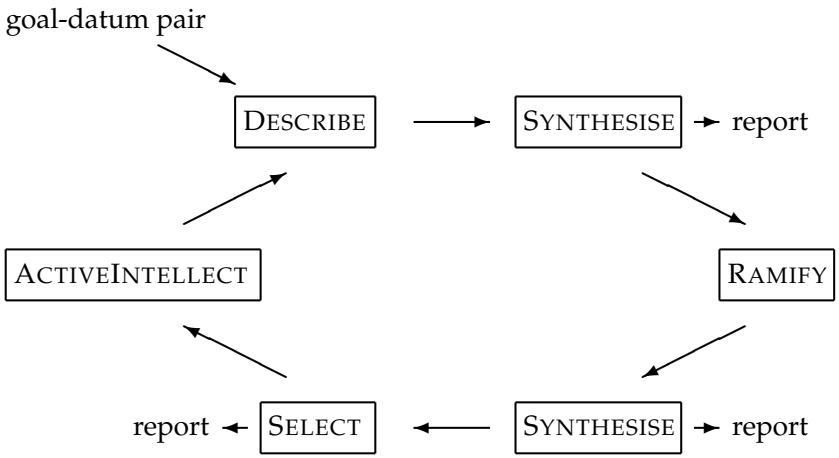
The classic specification of PROLOG's search procedure by Börger and Rosenzweig (1995) serves as a standard for description of proof search algorithms.

Compound syllogisms are of arbitrarily high complexity:

> 'In demonstrative reasoning there is nothing wrong with having proofs that run to a thousand steps.' (Ibn Sīnā *Burhān* 201 (Afifi))

So a proof search algorithm has to be built around recursion,
typically with a backtracking procedure that uses an enumeration of all possible search routes.

Ibn Sīnā's recursion loop (my analysis):



The goal-datum pair consists of a goal (the conclusion) and a datum (parts of a supposed proof).

The module DESCRIBE identifies a hole in the proof, and describes the lefthand and righthand edges of the hole. (Ibn Sīnā remarks that a more complicated procedure, to be described in his *Appendices*, can handle proofs with more than one hole.)

The module SYNTHESISE shrinks down the proof to left and right of the hole, starting at the left (Arabic right) and working towards the right (Arabic left).
It reports failure if it finds a logical error.

The module RAMIFY lists the finitely many single sentences that could close the gap, and creates new goal-datum pairs by putting each of these in the gap.

The module SYNTHESISE (second appearance) uses the shrinking technique to check the logical validity of each filled-in proof. If none are valid, it reports failure. Otherwise it outputs the valid proof with weakest fill ϕ.

The module SELECT checks whether ϕ is a known fact. If yes, it reports success. If no, it removes the fill and outputs the unfilled goal-datum pair.

The module ACTIVEINTELLECT suggests sentences that will narrow the gap without closing it.

This step is not deterministic. Ibn Sīnā himself suggests alcohol, prayer and sleep as useful aids.
It would be deterministic if we were reasoning from a specified theory, as for example in PROLOG.

Finally ACTIVEINTELLECT passes back to DESCRIBE the new goal-datum pairs got by adding in the suggested sentences, and the cycle starts over again.

Ibn Sīnā never describes the algorithm explicitly.

Instead, in his *Qiyās* ix.6 he gives his students 64 exercises with hints for the solutions.

He doesn't say what the exercises are for,
but in fact virtually all of the algorithm can be extracted from his hints for solution.

Samples:

> [Problem 1.] Suppose the goal is universally quantified affirmative, namely 'Every *C* is an *A*', and suppose that the found premises are 'Every *C* is a *B*' and 'Every *D* is an *A*'. Then if it's clear that 'Every *B* is a *D*', your syllogism is in good order; otherwise it needs a middle.
>
> [Problem 4, goal 'No *C* is an *A*'.] Suppose the found [premises] are 'No *C* is a *B*' and 'Every *D* is an *A*'. Then it can't be used.
>
> [Problem 37.] If the goal is universally quantified affirmative [thus: 'Every *C* is an *A*']; and you have [the premises] 'Every *D* is a *B*' and 'Every *B* is an *A*', and 'Every *C* is a *D*' is attached, this makes [the syllogism] determinate.
>
> Work through the remaining cases of this kind for yourself.

A precise description of the algorithm,
as an abstract state machine
(the format used by Börger and Rosenzweig),
takes four pages in

Wilfrid Hodges, 'Ibn Sīnā on Analysis I: Proof Search'
in *Fields of Logic and Computation; Essays dedicated to Yuri Gurevich*, edited by Blass et al., Springer Lecture Notes in Computer Science 6300 (2010) 354–404.