AUTOMATED REASONING

Krysia Broda
Room 378
kb@imperial.ac.uk

SLIDES 0:

INTRODUCTION and OVERVIEW
Introduction to the course -
what it covers and what it doesn't cover
Examples of problems for a theorem prover
Prolog – an example of a theorem prover

Automated Reasoning (what this course is about)

machine does 'thinking' user does nothing	$\longleftrightarrow$	user does 'thinking' machine keeps the books

• Concerned with **GENERAL DEDUCTION** - applicable in many areas.
Specifically First order logic / equality - not modal or temporal logics

PROBLEM: DATA |= CONCLUSION (|= read as "logically implies")
ANSWER: YES/NO/DON'T KNOW (i.e. give up

• YES + 'proof' - usually just one and smallest if possible, or
• YES + all proofs (or all answers) - (c.f. logic programming)

• Data is generally expressed either:
in standard first order logic, or in clausal form, or as equalities only.

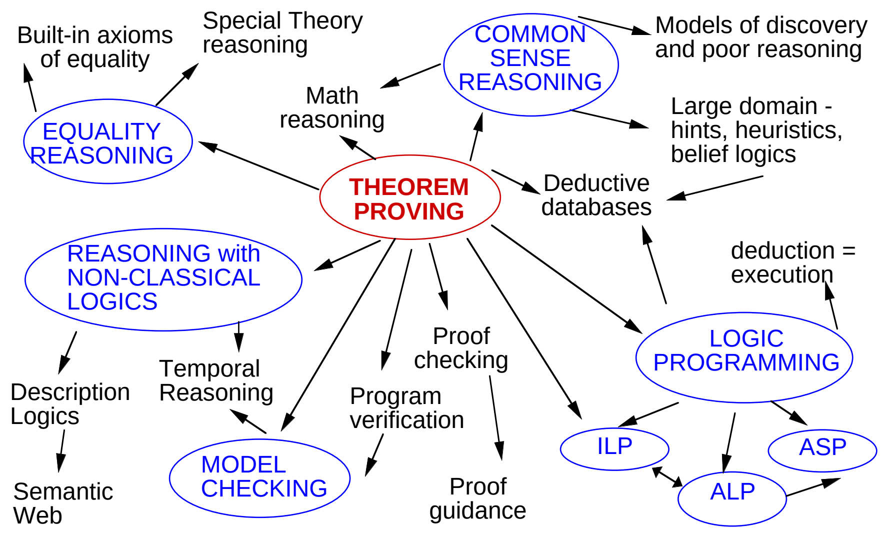
Problem can be answered:

√ with refutation methods (show data + ¬ conclusion give a contradiction) – resolution and tableau methods covered here;

√ for equality using special deduction methods (covered here);

x directly, reasoning forwards from data using inference rules, or backwards from conclusion using procedural rules; eg natural deduction (but not here)

The Spread of Automated Reasoning



Prolog is a Theorem Prover (1)

0aiii

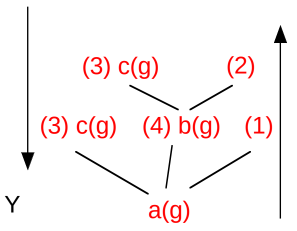
Data: (1) a(Y):-b(Y),c(Y). (2) b(X):-c(X). (3) c(g).
(Variables X,Y universally quantified)

Conclusion: a(g)
Show Data implies Conclusion.

Can work *forwards* from facts eg

from c(g) and b(X):-c(X) for all X, can derive b(g);

from b(g), c(g) and a(Y):-b(Y),c(Y) for all Y can derive a(g)



can also read tree *backwards*:

to show c(g), again note c(g) is a fact

to show b(g), by (2) show c(g)
c(g) is a fact, so ok

to show a(g), by (1) show (b(g) and c(g)) (i.e. show b(g) and show c(g))

Question:
The proof is the same, whether read top-down or bottom-up. But the processes of searching for it are different. Which is better? Why?

Prolog is a Theorem Prover (2)

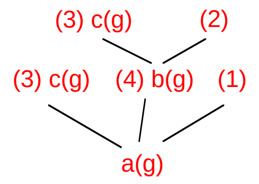
0aiv

Data: (1) a(Y):-b(Y),c(Y). (2) b(X):-c(X). (3) c(g).
(Variables X,Y universally quantified)

Conclusion: a(g)
Show Data implies Conclusion.

Prolog reading
(procedural interpretation)

?a(g) — show a(g)
(1)
?b(g), c(g) — show b(g) and c(g)
(2)
?c(g), c(g) — show c(g) and c(g)
(3)
?c(g) — show(c(g)
(3)
[ ] — done



to show a(g), show (b(g) and c(g))
(i.e. show b(g) and show c(g))
to show b(g), show c(g)
c(g) is a fact, so ok
to show c(g), again note c(g) is a fact

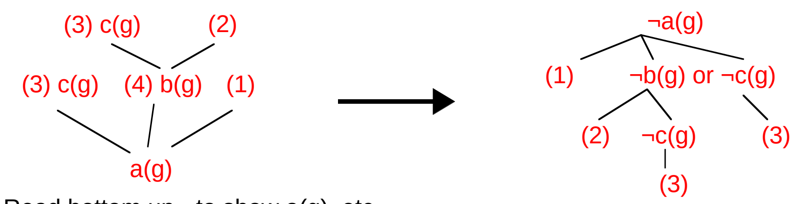
Prolog is a Theorem Prover (3)

0av

Data: (1) a(Y):-b(Y),c(Y). (2) b(X):-c(X). (3) c(g).
Conclusion: a(g)

In effect Prolog assumes conclusion false (i.e. ¬a(g))
and derives a contradiction – called **a refutation**

From ¬a(g) and a(Y):-b(Y),c(Y) derive ¬(b(g) and c(g)) ≡ ¬b(g) or ¬c(g) (*)
Case 1 of (*) if ¬b(g), then from b(X):-c(X) derive ¬c(g)
¬c(g) contradicts fact c(g)
Case 2 of (*) if ¬c(g) again it contradicts c(g)
Hence ¬b(g) or ¬c(g) leads to contradiction;
Hence ¬a(g) leads to contradiction, so a(g) must be true



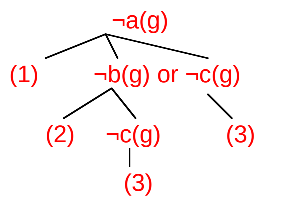
Read bottom up - to show a(g), etc.

Read top-down - assume ¬a(g), etc.

Prolog is a Theorem Prover (4)

0avi

Data: (1) a(Y):-b(Y),c(Y). (2) b(X):-c(X). (3) c(g).
Conclusion: a(g)



Prolog reading
(procedural interpretation)

?a(g) — show a(g)
(1)
?b(g), c(g) — show b(g) and c(g)
(2)
?c(g), c(g) — show c(g) and c(g)
(3)
?c(g) — show(c(g)
(3)
[ ] — done

¬a(g) may be read as "show a(g)" then the refutation and procedural interpretation are isomorphic
to show a(g), show (b(g) and c(g))
(i.e. show b(g) and show c(g))
to show b(g), show c(g)
c(g) is a fact, so ok
to show c(g), again note c(g) is a fact

The various interpretations are similar because the Data uses Horn clauses

Introduction:

The course slides will generally be covered “as is” in class. Course notes (like this one) amplify the slides. There are also some longer, and older , (“chapter”) notes with more background material (not examinable) on my webpage (www.doc.ic.ac.uk/~kb).

In this course we'll be concerned with methods for automating reasoning.
We'll look at the "father" of all methods, *resolution*, as well as at *tableaux* methods.
We'll also consider *reasoning with equality*, especially when the data consists only of equations. All data will be in first order logic.

Of course, there are special systems for automating other kinds of logics, for example modal logics. The methods may be extensions of methods for first order logic, or reduce to using theorem provers for first order logic in special ways. It is mainly for these reasons (but also because there isn't enough time) that we restrict things.

Logic programming (LP) is a familiar, useful and simple automated reasoning system which we will occasionally use to draw analogies and contrasts. e.g in LP we usually desire all solutions to a goal. In theorem proving terms this usually amounts to generating all proofs, although often just one proof might be enough. In LP reasoning is done by *refutation*. The method of refutation assumes that the conclusion is false and attempts to draw a contradiction from this assumption and the given data. Nearly always, refutations are sought in automated deduction.

A definite logic program clause A:-B,C can be used *forwards*: “from A and B conclude C” to reason directly from data to conclusion. Reasoning may be directed *backwards* from the conclusion, as in "to show A, show B and C". Deductions produced by LP are sometimes viewed in this way, when it is referred to as a *procedural interpretation*.

Introduction (continued):

Very often, derivations use *resolution*. Backwards reasoning: “to show C, show A and B” is implemented using resolution, in which the initial conclusion is negated. These things will be shown for the simple case of definite logic programming, and then more generally.

What we won't be concerned with: particular methods of knowledge representation, user interaction with systems, reasoning in special domains (except equational systems), or model checking. All of these things are important; a problem can be represented in different ways, making it more or less difficult to solve; clever algorithms can be used in special domains; user interactive systems are important too - e.g. *Isabelle* or *Perfect Developer*, as they usually manage to prove the easy and tedious things and require intervention only for the hard proofs. Isabelle and its relations are now very sophisticated. If a proof seems to be hard it may mean that some crucial piece of data is missing (so the proof doesn't even exist!) and user interaction may enable such missing data to be detected easily. Satisfiability checking (satsolvers), and Symbolic model checking (eg Alloy) can also be used to show a conclusion is not provable by finding a model of the data that falsifies the conclusion.

In order to see what kinds of issues can arise in automated reasoning, here are 3 problems for you to try for yourself. Are they easy? How do **you** solve them? Are you sure your answer is correct? What are the difficulties? You should translate the first two into logic. In the third, assume all equations are implicitly quantified over variables x, y and z and that a, b, c and e are constants.

EXAMPLE PROBLEMS

0bi

- A general theorem prover might be expected to solve all of the following 3 problems easily.
- The user would translate the data into logic first.
- How would YOU solve the problems?
- Are they easy? What are the difficulties?
- Are you sure the answer is correct?

Three naughty children:

Dolly Ellen or Frances was the culprit and only one.
The culprit was in the house.
Dolly said " It wasn't me, I wasn't in the house; Ellen did it."
Ellen said " It wasn't me and it wasn't Frances;
but if I did do it then (Dolly did it too or was in the house)."
Frances said " I didn't do it, Dolly was in the house;
if Dolly was in the house and did it so did Ellen."

None of the three told the truth. Who did it?

A harmonious household (!):

0bii

Someone who lived in the house stole from Aunt Agatha.
Agatha the butler and James live in the house and are the only people that do.
A thief dislikes, and is never richer than, his victim.
James dislikes no-one whom Aunt Agatha dislikes.
Agatha dislikes everyone except the butler.
The butler dislikes anyone not richer than Agatha and everyone she dislikes.
No-one dislikes everyone.
Agatha is not the butler.

Therefore Agatha stole from herself
(and also that neither James nor the butler stole from her)

A Mathematical Problem:

$a \circ b = c$
$\circ$ is an associative binary operator: $x \circ (y \circ z) = (x \circ y) \circ z$
$x \circ x = e$
$x \circ e = e \circ x = x$ (e is the identity of $\circ$)

Show $b \circ a = c$

Some hints for the problems.

0biii

For the naughty children:
Let the constants be d, e and f (for Dolly, Ellen and Francis);
You need predicates C(x) – x is a culprit, and H(x) – x is in the house;
The second sentence is $\forall x(C(x) \rightarrow H(x))$.

For Aunt Agatha's Burglary:
You can ignore "lives in the house".
Let the constants be a, b, j and predicates be s(x,y): y is the victim of thief x,
d(x,y): x dislikes y, r(x,y): x is richer than y and = (usual meaning);
The first sentence is $\exists x(s(x,a))$, which can be simplified to s(m,a);
The last piece of data is $\neg$ (a=b);
You'll need to reason about equality:
if data S holds for x, and x=y, then S holds for y.

Mathematics:
There are 4 constants, a, b, c and e, and predicate P(x,y,z) meaning $x \circ y = z$;
You can either translate the data using P
(remembering to translate the associativity property usng P as well)
or keep with = only and reason directly with equality;
Try both!

Some Useful References and Websites

0ci

(See Chapter Notes for additional papers etc.)

Alan Bundy: The Computer Modelling of Mathematical reasoning (Academic)

Chang & Lee: Symbolic Logic and Mechanical Theorem Proving (Academic)

Mel Fitting: First order Logic and Automated Theorem Proving (Springer)

Alan Robinson: Logic:Form and Function (Edinburgh)

Larry Wos: Automated Reasoning: Introduction and Applications (McGrawHill)

J. Kalman: Automated reasoning with Otter (Rinton)

Blasius & Burckert: Deduction Systems in Artificial Intelligence (Ellis Horwood)

Lassez & Plotkin (Eds): Computational Logic (MIT)

Kakas & Sadri (Eds): Computational Logic: Logic Prog and Beyond (Springer)

R. Veroff: Automated Reasoning and its Applications (MIT)

J. Gallier: Logic for Computer Science: Foundations of Automated Theorem Proving (Harper Row)
Several "freetech" books from www.freetech**books**.com

Useful References and Websites Continued

0cii

CADE: Conference on Automated Deduction (Springer)
http://www.cadeconference.org/ (old: http://www.cs.albany.edu/~nvm/cade.html)

TABLEAUX: Conference on Analytic Tableau and Related Methods (Springer)
http://i12www.ira.uka.de/TABLEAUX/

Workshop on First Order Theorem Proving http://www.csc.liv.ac.uk/FTP-WS/
(Old: http://www.mpi-sb.mpg.de/conferences/FTP-WS/)

http://en.wikipedia.org/wiki/Automated_theorem_proving
(Slightly biased contents)

Theorem Proving at Argonne
http://www-unix.mcs.anl.gov/AR/ (Includes Otter theorem Prover)

http://www.uni-koblenz.de/~beckert/leantap/ (A tableau based prover)

http://www.leancop.de/ (Another tableau based prover)

http://www.cl.cam.ac.uk/research/hvg/Isabelle/

http://alloy.mit.edu/

http://www.cs.miami.edu/~tptp/ (Thousands of problems for Theorem Provers)

http://www.cs.swan.ac.uk/~csetzer/logic-server/software.html (List of Provers)

Summary of Slides 0

0di

1. This course is concerned with the automation of logical reasoning.

2. You are already familiar with one automated reasoner, Prolog. Its evaluation can be viewed in two ways: either as a refutation whereby the conclusion is negated and a contradiction derived from it by resolution using the data, or as a natural deduction process working backwards from the goal.

3.There are other refutation methods, including general resolution and semantic tableaux, which will be covered in the course.

4. Problems can involve propositional data (without quantifiers), or data with quantifiers. Data may include use of the equality predicate, for which special methods can be used (eg Knuth Bendix Procedure), also covered in the course.

5. There are other aspects to automated reasoning, including systems with more user interaction, model checking, modal and temporal logic provers and knowledge representation. This course will not be considering these things.

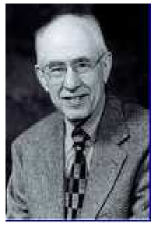
6. You will be able to use several theorem provers in this course.

Gallery 1: Famous Names in Theorem Proving

Alan Robinson
Discovered Resolution
1961

Gerard Huet
Discovered
rewrite systems
1976

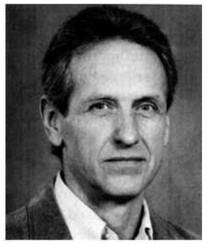
Alan Bundy
Clam Theorem
Prover,
excellent book,
problem solving


Hillary Putnam
Davis Putnam
prover 1957

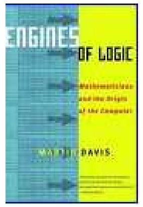
Larry Wos
Wrote Otter
prover
(1980s) and
successor
Prover9

Reiner Hahnle
LeanTap 1997 and
other provers
Key System for
Program Correctness
2005

Gallery 2: Famous Names in Theorem Proving


Robert
Kowalski
(1979) Theory
of Logic
Programming
Connection
Graphs (1976)

Chang and Lee
First book on theorem
Proving (1968)



Donald Knuth
Knuth Bendix
Algorithm
(1971)
Tex

Martin Davis
Davis Putnam Algorithm
(1957)

J S Moore
Boyer MooreProver (1970s)
Structure Sharing (1971)

Gallery 3: Famous Names in Theorem Proving

Donald Loveland
Model
Elimination 1967

Alain
Colmerauer
first Prolog
System

David Plaisted -
Hyper-linking (1992)

Reinhold Letz -
Setheo, Free variable
Tableau proving
(1990s onwards)

Mark Stickel
- PTTP
Theorem
Prover 1982

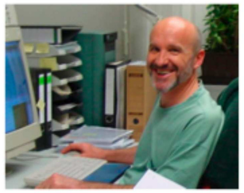
Gallery 4: Famous Names in Theorem Proving

Woody Bledsoe - UT
prover (natural
deduction style)

Ian Horrocks
- Description
Logic
Theorem
Prover

Larry Paulson -
Inventor of Isabelle

Andrei
Voronkov -
Vampire
Theorem
Prover


Norbert Eisinger
- Connection
Graph theory
(1986)