

Automated Reasoning

Ruben Gamboa
Artificial Intelligence II
January 24, 2006
Laramie, WY

Goals of this talk

- * Fundamental research questions in automated reasoning
- * Relationship to other fields
- * Approaches to automated reasoning

Goal of Automated Reasoning

- * Write programs that can prove “mathematical” theorems
- * Ideally, we can answer questions that we could not answer before
 - * 4-color theorem
 - * Robbins algebra == Boolean algebra
- * At least verify existing theorems

Relationship to AI

- * Automated Reasoning was an early concern of AI (e.g., Simon, Newell)
- * It has been used (with mixed results) as a reasoning engine in AI projects
 - * Expert Systems
 - * Knowledge Systems
 - * Logic Programming
 - * Automatic Programming

Relationship to AI

- * AI techniques have been used in automated reasoning
- * Planning
- * Understanding failed proofs
- * Proof by Analogy

Relationship to Logic

- * Leibnitz hoped questions of human affairs could be settled mechanically
- * Godel even pondered the implication of finding proofs in “quadratic” time...
- * ...anticipating Cook’s notion of NP by a few decades
- * Logic is the foundation for automated reasoning

Propositional Logic

- * Study of logical connectives
 - * and, or, not, implies, etc.
- * Truth can be decided by truth tables
- * Tautologies can also be derived using mechanical rules (Post)

Propositional Logic

- * Do NOT underestimate propositional logic!
- * According to CS theory, even deciding whether a propositional formula is satisfiable is much harder than inverting a matrix
- * Many deep mathematical conjectures can be stated in propositional logic

Ramsey's Theorem

- * For each s and t , there is a finite number n , denoted by $R(s, t)$, such that any graph of size n either has a completely connected subgraph of size s or a completely disconnected subgraph of size t .
- * For any s , t , and n , this can be stated as a propositional formula!

Automating Propositional Logic

- * We can construct truth table
- * Or just enough of one to verify/falsify the theorem

Clausal Form

- * We can restrict ourselves to formulas in “clausal form”:
 - * “conjunction of disjuncts”
 - * “ands of ors”
- * This simplifies treatment of the problem (although from a theoretical perspective it's still just as hard)

CNF and DNF

- * There are two notions of "clause"
 - * CNF = "ands of ors"
 - * DNF = "ors of ands"
- * Satisfiability is easy for DNF
- * Provability is easy for CNF

CNF and DNF

- * Given a formula φ it is efficient to find formulas φ_D and φ_C such that
 - * φ_C is in CNF and is satisfiable iff φ is
 - * φ_D is in DNF and is a tautology iff φ is
- * Note: φ_C and φ_D are equivalent to φ in the “wrong” way for an efficient SAT solver or automated theorem prover

Davis-Putnam

- * This can be used to determine if a propositional formula in CNF is satisfiable
- * Rule 1: The 1-literal rule
- * Rule 2: The affirmative-negative rule
- * Rule 3: Eliminating atomic formulas

DP: 1-Literal Rule

- * If there is a unit clause containing p , remove all clauses that contain p , and remove $\neg p$ from all clauses that contain it

p	
$p, \neg q$	$\neg q$
$\neg p, \neg q$	q, r
q, r	

DP: Affirmative-Negative Rule

- * If a literal occurs only positively or negatively, we can remove all clauses that contain it

p	p
$p, \neg q$	$p, \neg q$
$\neg p, \neg q$	$\neg p, \neg q$
q, r	

DP: Eliminating Atomic Formulas

- * Let p be some literal that occurs positively and negatively in the clauses
- * Let p, C_i be all the clauses that contain p
- * Let $\neg p, D_j$ be all the clauses that contain $\neg p$
- * Replace all those clauses with the clauses C_i, D_j (for all i and j)

DP: Eliminating Atomic Formulas

- * This rule is related to resolution (and the clause C_i , D_j is called a resolvent of the clauses p , C_i and $\neg p$, D_j)

p	$\neg q$
$p, \neg q$	$\neg q, \neg q$
$\neg p, \neg q$	q, r
q, r	

Davis-Putnam Redux

- * Apply rules I and II repeatedly
- * If the empty clause is derived, fail
- * If we end up with no clauses, succeed
- * If neither rule I nor rule II applies, try rule III
- * Keep trying!

IF-Trees

- * If-trees are an alternative representation for propositional clauses
- * if x then
 - if y then
 - true
 - else
 - false
 - else
 - false

OBDDs

- * If-trees can be a canonical representation for propositional formulas, as long as:
 - * Along any branch, we test propositions in a fixed order (e.g., $a < b < \dots < z$)
 - * We share identical sub-trees (so we have a graph, not a tree)
 - * We avoid useless tests
 - * if x then true else true

OBDD Properties

- * OBDDs are canonical, so all equivalent formulas have the same representation
- * x and y
- * y and x
- * x and $(y \text{ or } z)$ and $(\text{not } z \text{ or } y)$
- * OBDDs can be used to determine if a formula is a tautology or if it is satisfiable

OBDD Applications

- * OBDDs are used in practice to verify the correctness of hardware algorithms
 - * e.g., adders
- * Also used to support temporal reasoning via model checking
- * Used to verify correctness of protocols, the control logic of a CPU, even software

Predicate Logic

- * Extends propositional logic with quantifiers, variables, and functions
- * There are many versions of predicate logic, but we'll stick with classical first-order logic

Aside: Other Logics

- * Non-monotonic logics can be used for knowledge representation
- * Modal logics work for reasoning about agents
- * Higher-order logics are more natural in certain branches of mathematics (e.g., the reals)

What is true?

- * Remember that propositional formulas had truth tables to determine “truth”, as well as a process to derive true formulas
- * In predicate logic we use Tarski’s definition of truth: true under all interpretations
- * This involves building interpretations (models) for the functions and variables in the formula

What is provable?

- * There is also a notion of “provable” or “derivable”
 - * using “rules of logic” (aka “syllogisms”)
- * The true statements are precisely the provable statements (Godel)
- * There are “true” statements of arithmetic that are neither provable nor disprovable (Godel)

A Surprising Corollary

- * Since proofs are finite, we can enumerate all the proofs to find all the theorems
- * So if a formula (or its negation) is a theorem, we will eventually find a proof
- * Since some formulas are neither theorems nor contradictions, this is not an effective procedure to determine if an arbitrary formula is derivable

Prenex Form

- * As with propositional logic, it helps to start out by normalizing the formulas
- * First of all, all quantifiers can be moved to front:

$$((\forall x)P(x)) \wedge ((\forall x)Q(x)) \Leftrightarrow ((\forall x)(P(x) \wedge Q(x)))$$

Skolemization

- * We can also eliminate existential quantifiers
- * The idea is to create a new function symbol for each existentially quantified variable

$$((\forall x)(\exists y)P(x, y) \Leftrightarrow ((\forall x)P(x, f_y(x))))$$

Equivalences

- * It turns out that if you start with a formula φ , move all quantifiers to the front, and Skolemize it, you end up with a formula φ' that is satisfiable if and only if φ is satisfiable
- * There is a dual theorem for truth (instead of satisfiability) that gets rid of universal quantifiers

Clausal Form

- * Move quantifiers to the front
- * Eliminate existential quantifiers (Skolemization)
- * Remove universal quantifiers, because all variables are universally quantified
- * Propositionally turn the remaining formula into (propositional) clausal form

Axiom Schemas

- * Some theories have an infinite number of axioms, but these axioms follow a predictable pattern (i.e., schema)
- * E.g., induction axioms
- * Theorem provers can not deal with an infinite number of clauses, but they can create the right "clause" as needed
- * E.g., an induction inference rule

Herbrand's Theorem

- * A set of predicate clauses is unsatisfiable if and only if the (countable) set of propositional clauses generated by replacing each variable with all possible ground instances is unsatisfiable
- * Corollary: to determine if a theorem is true of the reals, you only have to consider countably many numbers

Compactness Theorem

- * A (possibly infinite) set of clauses is unsatisfiable if and only if some finite subset of clauses is also unsatisfiable
- * Corollary: there are models of arithmetic with “infinite” integers

Automated Reasoning

- * Simple (and ineffective) automated reasoning technique for predicate logic:
- * We only need to consider the ground instances of the clauses...
- * ...and only finite subsets of the ground clauses
- * Use Davis-Putnam on the finite ground clauses

Resolution

- * A simpler alternative is to derive the ground clauses you need as you go along
- * We use resolution to find the needed ground clauses
- * But it's harder to determine when two literals can be resolved
 - * e.g., $P(a)$ vs. $\neg P(x)$

Unification

- * Unification is a general algorithm that can be used to determine when two literals can be resolved
- * Unification creates a mapping of variables to terms
- * If the mapping is applied to the two terms, they become identical

Unification Examples

$P(x, f(y))$	$P(0, f(0))$	$x=0, y=0$
$P(x, 0)$	$P(0, y)$	$x=0, y=0$
$P(x, y)$	$P(z, z)$	$x=z, y=z$
$P(x, f(0))$	$P(y, f(y))$	$x=0, y=0$
$P(x, y)$	$P(y, 0)$	$x=0, y=0$

Binary Resolution

- * Consider these two clauses:
 - * $P(\dots), Q_1, Q_2, \dots, Q_n$
 - * $\neg P(\dots), R_1, R_2, \dots, R_m$
- * Let Q' (R') be the result of applying the unifying substitution to Q (R)
- * The resolvent is given by
 - * $Q_1', Q_2', \dots, Q_n', R_1', R_2', \dots, R_m'$

Resolution

- * Repeatedly apply the binary resolution rule until either (a) no more resolution steps are possible, or (b) the empty clause is derived
- * If the original set of clauses is unsatisfiable, there is some derivation of the empty clause
- * Almost true—you also need “factoring”

Challenges

- * The problem is efficiency
- * How do we choose the next clauses to resolve?
- * Every time we make a resolution step we generate new clauses
- * Problem gets worse and worse...

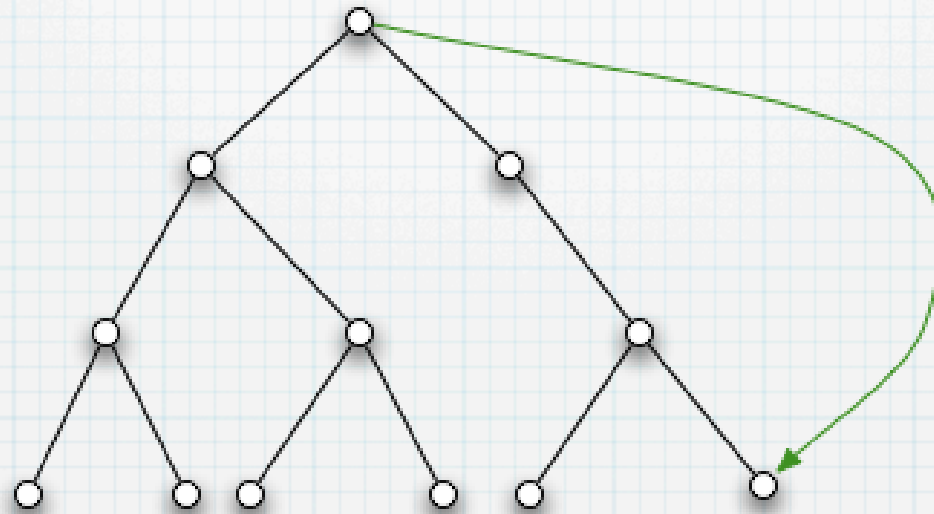
Automated Reasoning Paradox

- * That explains the following paradox
- * The more theorems humans know, the easier it is for them to prove new theorems
- * But adding theorems (i.e., hypotheses) usually slows down theorem provers

Approaches to Speed Up Theorem Proving

- * Make “big” steps instead of “little” steps
 - * We may be able to find a proof before the combinatorial explosion gets us
- * Limit the “legal” resolutions
- * Use heuristics
- * Special-purpose techniques

Big Steps Approach



Hyperresolution

- * A “big” step approach
- * Resolve multiple clauses at once
- * One of the clauses is a “nucleus” with n positive literals (and maybe other literals)
- * Remaining n clauses are “electrons” with only negative literals
- * The (single) resolvent is a new “electron”

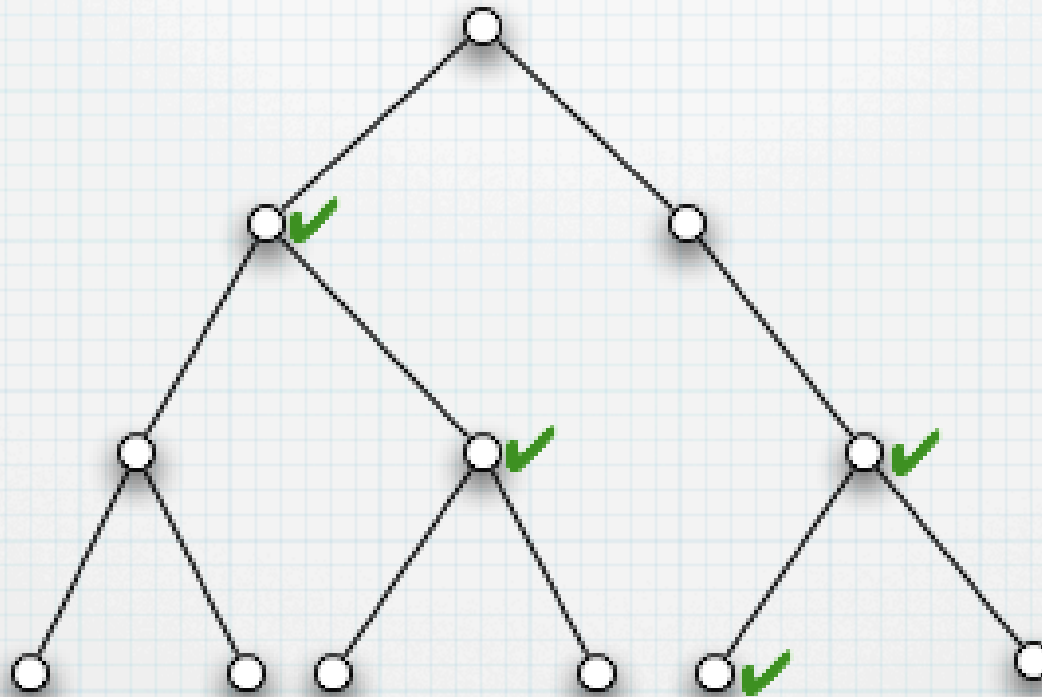
Linear Resolution

- * Only “straight-line” proofs are “legal”
- * Ordinarily, resolution results in a tree of nodes
- * Restrict resolution to “linear” trees:
 - * Every resolution step uses the resolvent of the previous resolution step
- * As long as we can reuse clauses from earlier in the tree, this is complete

Positive Resolution

- * Another “legal” approach:
 - * One of the clauses on each resolution step must consist of only positive literals
- * Amazingly, this is complete!

Heuristics Approach



Set of Support

- * Most theorems consist of a “hypothesis” and a “conclusion”
- * The set-of-support strategy says that all legal resolutions involve at least one clause that is not in the “hypothesis”
- * As long as the hypotheses are consistent, this strategy (heuristic) is complete

Unit Resolution

- * Another strategy (or heuristic) approach
- * The idea is to prefer resolution steps that use unit clauses
- * As a strategy, this is similar to Davis-Putnam's Rule 1
- * Some authors use this as a "legal" approach...in which case it's not complete

Special Techniques

- * Work on specific domains
- * Usually they work very well in these restricted domains
- * They may even solve the problem completely in these domains

Horn Clauses

- * A clause is called a Horn clause if it contains at most one positive literal
- * If it contains exactly one positive literal, it is called definite
- * Horn clauses can be thought of as implications:
 - * $\neg p_1$ or $\neg p_2$ or ... or $\neg p_n$ or q
 - * p_1 and p_2 and ... and $p_n \Rightarrow q$

Resolution on Horn Clauses

- * Horn clauses suggest a natural, goal-directed resolution technique, called backchaining:
- * To prove q : prove p_1 , prove p_2 , ..., and prove p_n
- * This works as long as we're careful about loops
- * e.g., use iterative deepening and backtracking

Infinite Regression

- * Backchaining can easily lead to infinite loops
- * Consider these rules:
 - * $P(s(x)) \Rightarrow P(x)$
 - * $P(0)$
- * Now try to prove $P(0)$...

Prolog

- * Prolog can be described as an incomplete, unsound, but really fast theorem prover
- * Prolog works on Horn clauses, and it uses backchaining as described above
- * with backtracking, but not depth limits or other techniques to prevent infinite loops

Dealing with Equality

- * Equality is an important notion in many theories
- * Every (finite) first-order theory can be extended to a (finite) first-order theory with equality
- * So we do not have to do anything special for equality...just add equality axioms

Higher-Order Unification

- * One way to deal with equality is to add it to unification:
- * C-unification: $x+y$ unifies with $y+x$
- * AC-unification: $x+(y+z)$ and $(x+z)+y$
- * ACl-unification: $\{x,y,x\}$ and $\{x,y\}$
- * This has been used to prove long standing conjectures in algebra

Paramodulation

- * A special technique to deal with equality
- * Allows us to “resolve” an equality with an arbitrary literal that contains a term that unifies with one side of the equality

Clause 1	Clause 2	Resolvent
P	Q	P'
		Q'
$s1 = t$	$R(s2)$	$R(t)'$

Completeness of Paramodulation

- * Paramodulation is important because
 - * it is a complete solution to equality
 - * it is faster than the axiomatic approach
- * But it is still too computationally expensive

Term Rewriting

- * Another approach to equality is term rewriting
- * Each equality axiom is “ordered”, so it is applied in only one direction (to “simplify” terms):
 - * $x + 0 \rightarrow x$
- * Obviously more efficient

Theories of Equality

- * Some theories have equality as their only predicate
- * E.g., group theory
- * Term rewriting can be used by itself (i.e., without resolution) in these cases

General Term Rewriting

- * We can also use term rewriting techniques in general cases
- * Each predicate $P(x,y)$ is converted to a function $p(x,y)$
 - * The predicate holds when $p(x,y)=\text{True}$
- * Now equality is the only predicate!

Canonical Rewriting

- * Sometimes we can guarantee that the rewrite rules eventually rewrite each term into a canonical form
- * So determine if $s=t$ is true, we rewrite s and t into canonical form and check if these are equal

Canonical Rewrite Rules

- * For a rewrite system to be canonical, it must have the following properties:
 - * Well-foundedness: no infinite chains of rewrite equations
 - * Confluence: if s can be rewritten into either s_1 or s_2 , then both s_1 and s_2 eventually rewrite into the same term
- * This means no intelligence is required to apply canonical rewrite rules

Generating Canonical Systems

- * Starting with a set of rewrite rules, it is possible (in some cases) to derive a canonical rewrite system
- * This process is called “completion” of the rewrite system, and it was worked out by Knuth and Bendix
- * This doesn't work for all rewrite rules, but you can tell when it fails

Group Theory

- * It turns out that the axioms of group theory can be completed with the Knuth-Bendix procedure
- * Corollary: group theory is decidable
- * But the axioms of Abelian group theory can not be completed with Knuth-Bendix
- * Abelian group theory is not decidable

Decidable Theories

- * Some first-order theories can be decided by an algorithm
- * Rather than use resolution or rewriting on those theories, you can simply call the algorithm to decide the issue
- * A bigger challenge is to use a special-purpose algorithm to decide a fragment of a large theorem (e.g., linear arithmetic)

Decidable Theories

- * Presburger (or linear) arithmetic:
 - * Integers, addition, subtraction, inequalities
- * Algebraically Closed Fields
 - * Fields where every polynomial (other than constants) has a zero

More Decidable Theories

- * Real Closed Fields

- * Ordered fields with square roots for positive numbers and roots for odd-degree polynomials

- * Euclidean Geometry

- * Wu's method converts geometry into algebra problems

Combining Decidable Theories

- * Some argue that decidable theories are the path to automated reasoning nirvana
- * The hard part is combining the different decidable algorithms in the right places
- * Van Baalen is an expert in this area

Proof Assistants

- * Theorem proving is hard
 - * We have only scratched the surface
- * A useful compromise is the development of proof assistants
 - * Proofs are guaranteed to be correct
 - * The assistants sometimes help

Examples

- * ACL2 is based on recursive function theory, rewriting, and induction
- * HOL is based on higher-order function theory, and introduced the use of tactics
- * NuPerl is based on type theory, and tactics
- * Otter is a pure resolution prover

Research in Automated Reasoning

- * How do we write good theorem provers?
- * How do we use theorem provers (and other programs) to do real math?
- * How do we formalize math?
- * How do we apply theorem provers (and other tools, e.g., model checkers) to software and hardware verification?