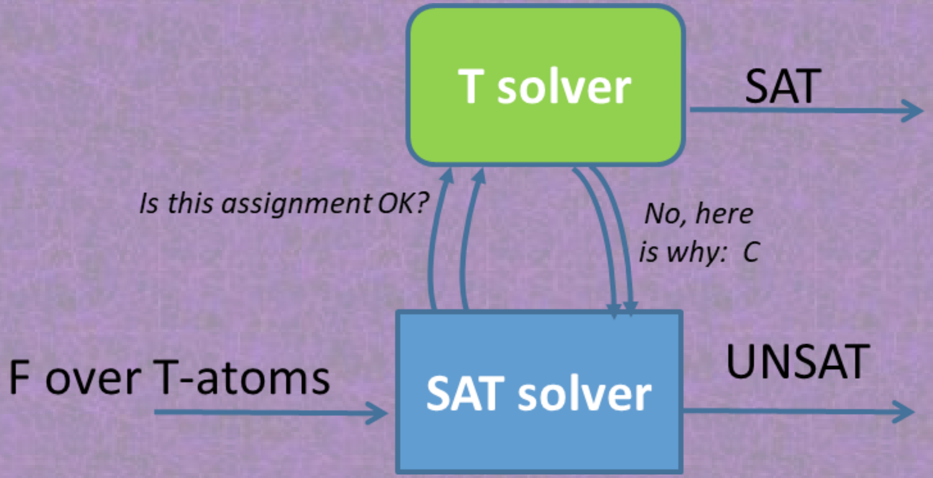
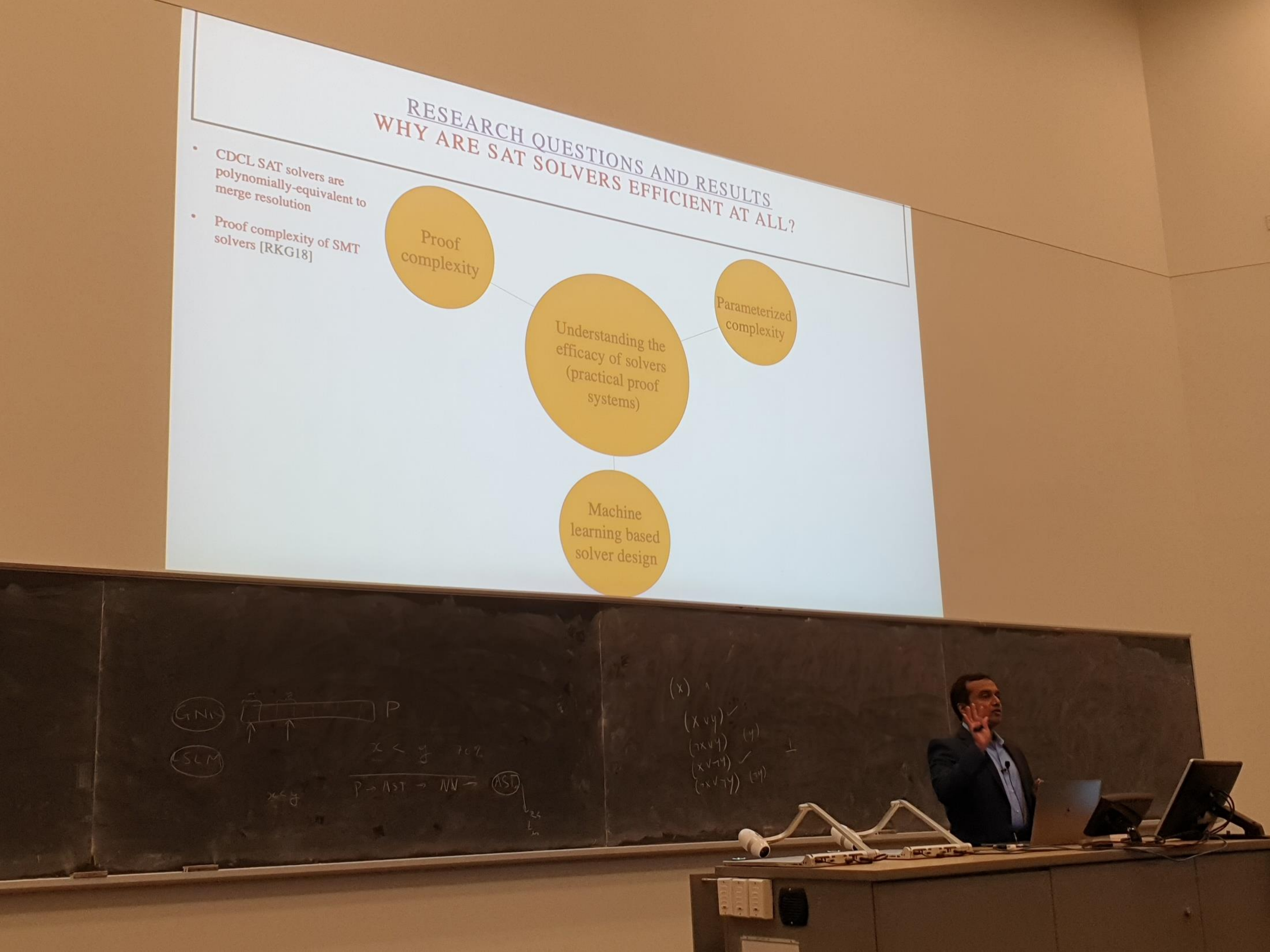
$$\frac{(C \vee x) \quad (D \vee \neg x)}{(C \vee D)}$$

Proof complexity of SAT/SMT solvers



Robert Robere
Antonina Kolokolova
Vijay Ganesh

Waterloo, Aug 29, 2019



RESEARCH QUESTIONS AND RESULTS

WHY ARE SAT SOLVERS EFFICIENT AT ALL?

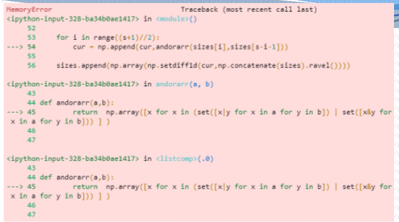
- CDCL SAT solvers are polynomially-equivalent to merge resolution
- Proof complexity of SMT solvers [RKG18]

Understanding the efficacy of solvers (practical proof systems)

Proof complexity

Parameterized complexity

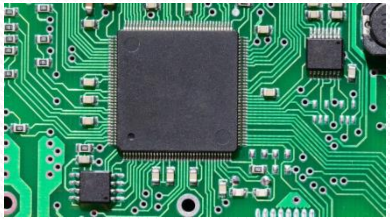
Machine learning based solver design

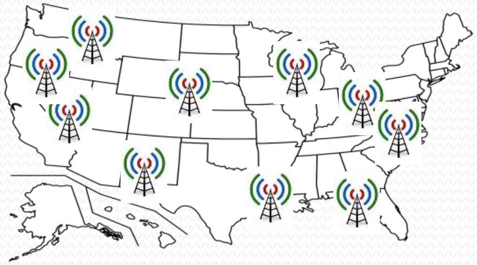


```
MemoryError                               Traceback (most recent call last)
<ipython-input-328-ba34b0ae1417> in <module>()
     52
     53     for i in range((s+1)//2):
---> 54         cur = np.append(cur,andorarr(sizes[i],sizes[s-i-1]))
     55
     56     sizes.append(np.array(np.setdiff1d(cur,np.concatenate(sizes).ravel())))

<ipython-input-328-ba34b0ae1417> in andorarr(a, b)
     43
     44 def andorarr(a,b):
---> 45         return  np.array([x for x in (set([x|y for x in a for y in b]) | set([x&y for
 x in a for y in b])) ] )
     46
     47

<ipython-input-328-ba34b0ae1417> in <listcomp>(.0)
     43
     44 def andorarr(a,b):
---> 45         return  np.array([x for x in (set([x|y for x in a for y in b]) | set([x&y for
 x in a for y in b])) ] )
     46
     47
```

Setting: what kind of problems are we solving?

And what constitutes an acceptable solution?





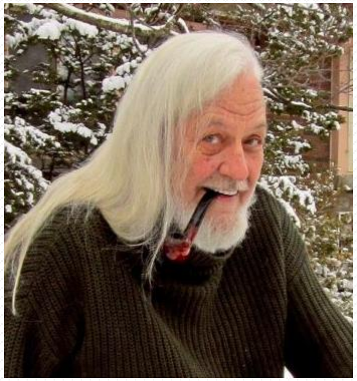
Computational complexity setting: *Given an instance of a problem, decide if this instance is in a specific set or not.*

- Is this number prime? 2147483647 ∈ PRIMES?
- Does this program reach a bad state?
- Is this formula satisfiable?
- Are these two graphs isomorphic?

- But how do we know that the answer is correct?
  - When do we have short and easy to check proofs?
  - What do the words "proof", "short" and "easy to check" mean?

What is “easy” and “short”?

- Jack Edmonds (1960s):
  - “Good algorithm”: runs in polynomial time
    - P: the class of all polynomial-time decidable languages
  - “Good characterization”: “certain information <..> which the supervisor can then use with ease to verify <..> “
    - Short: polynomial length.
    - NP: problems with easy to verify short proofs of all “yes” answers
    - coNP: … of all “no” answers.



Open: $P = NP$? $NP = coNP$? $NP \cap coNP = P$?

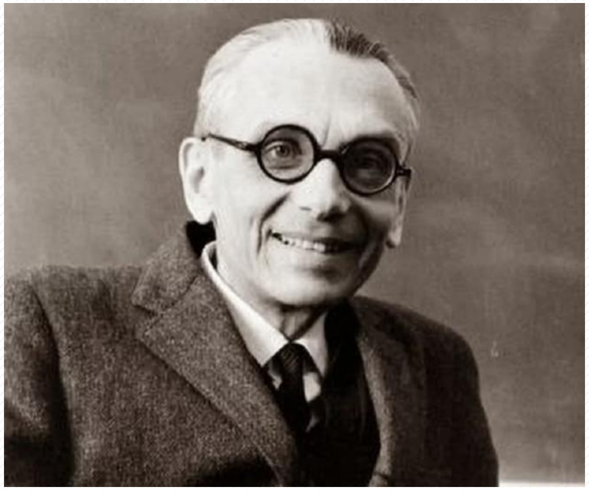
How hard is it to prove theorems?

- Gödel’s 1956 letter to von Neumann:
  - Consider an optimal algorithm checking if a given first-order formula F has a proof of size n.
    - size = number of symbols.
  - How fast can it be in the worst case, as a function of n?
    - $const \cdot n$? $const \cdot n^2$? …

- This letter was only discovered in 1988
  - So complexity theory, including the notion of NP-completeness, and proof complexity started independently from it.

Gödel’s 1956 letter to von Neumann

...If there really were a machine with ... $\sim k \cdot n$ (or even $\sim k \cdot n^2$), this would have consequences of the greatest importance. Namely, it would obviously mean that in spite of the undecidability of the Entscheidungsproblem, the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine. After all, one would simply have to choose the natural number n so large that when the machine does not deliver a result, it makes no sense to think more about the problem...



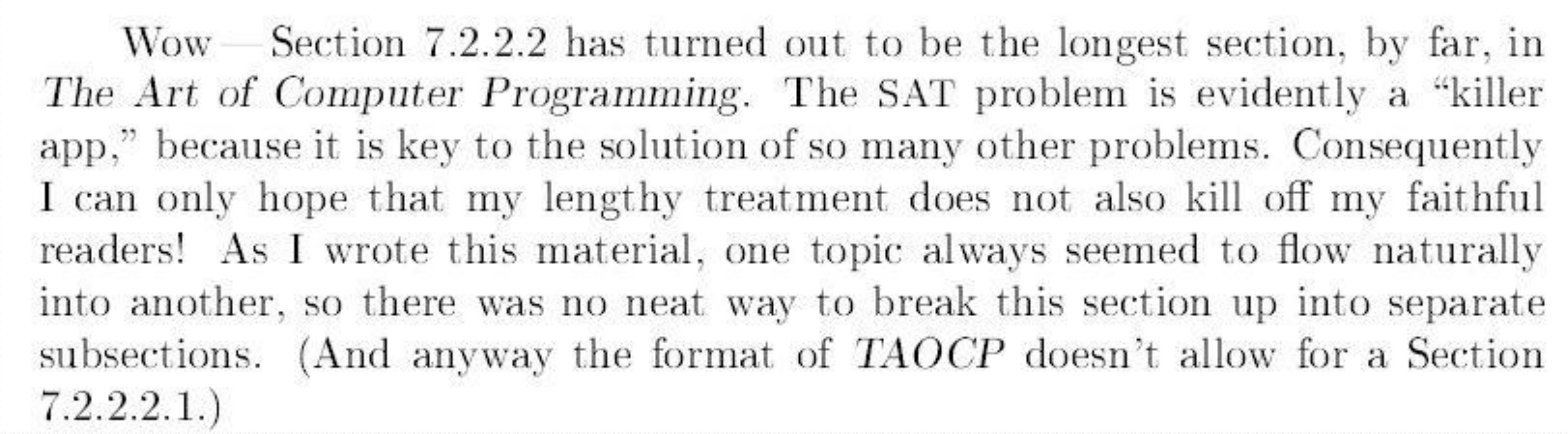
Special case: propositional formulas

- **Satisfiability problem:** well-formed propositional formulas with all variables (implicitly) existentially quantified.
  - Propositional setting: Boolean variables (domain = $\{true, false\}$)
  - **SAT**: set of all satisfiable propositional formulas. **UNSAT**: unsatisfiable.
  - e.g. $\varphi = (x \vee y) \wedge \neg x$ corresponds to F = $\exists x\, \exists y\, (x \vee y) \wedge \neg x$
  - Proof for membership in SAT: satisfying assignment.
  - Proof size = n

- **Tautology problem:** well-formed propositional formulas with all variables implicitly universally quantified.
  - **TAUT**: set of all valid propositional formulas.
  - e.g. $\varphi = (\neg\, x \wedge \neg y) \vee x$ corresponds to F = $\forall x\, \forall y\ \ (\neg\, x \wedge \neg y) \vee x$
  - Proof for $\in TAUT$: ?
  - Proof size bound = ?

- Note: $\varphi \in UNSAT$ iff $\neg\varphi \in TAUT$ iff $\varphi \notin SAT$

From Donald Knuth's "The art of Computer Programming"



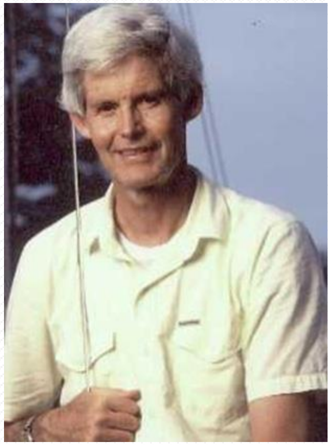
Wow — Section 7.2.2.2 has turned out to be the longest section, by far, in *The Art of Computer Programming*. The SAT problem is evidently a "killer app," because it is key to the solution of so many other problems. Consequently I can only hope that my lengthy treatment does not also kill off my faithful readers! As I wrote this material, one topic always seemed to flow naturally into another, so there was no neat way to break this section up into separate subsections. (And anyway the format of *TAOCP* doesn't allow for a Section 7.2.2.2.1.)

Proof complexity

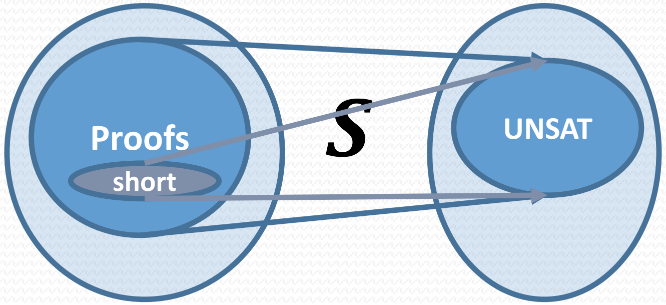
Cook’71, “Complexity of theorem-proving procedures”

Every problem in NP polynomially reduces to SAT

- NP-completeness
  - Easily verifiable short proofs of “yes”: satisfying assignments.
  - What about proofs of “no”?
- Proof complexity [Cook, Reckhow’79]



Propositional Proof Systems



A **propositional proof system** is a polynomial time computable onto function S: $\{0,1\}^* \rightarrow TAUT$

S is **polynomially bounded** if every unsatisfiable formula has a short (polynomial-size) proof.

- Proof size is the number of symbols in a proof.
- NP=coNP iff there exists a polynomially bounded proof system

S **p-simulates** S' if for all tautologies f, f has an S-proof of size at most polynomial of size of shortest S'-proof of f.

Cook's program: prove NP $\neq coNP$ (and so P ≠NP) by proving lower bounds for stronger and stronger proof systems

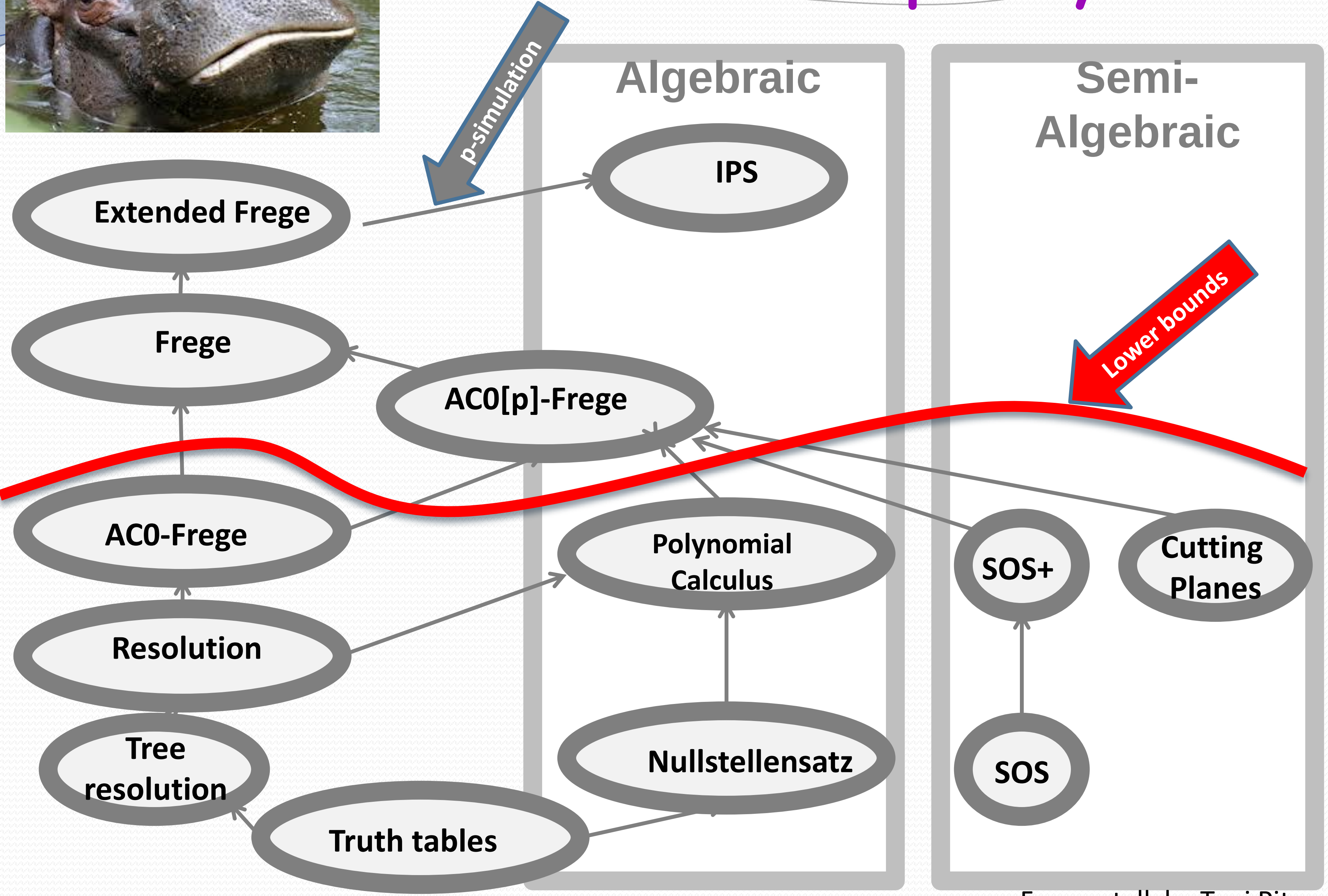
Propositional Proof Systems

These definitions do not say anything about the complexity of **finding** proofs (automatizability):

A system is automatizable if there is an algorithm finding a proof in time polynomial in proof size.

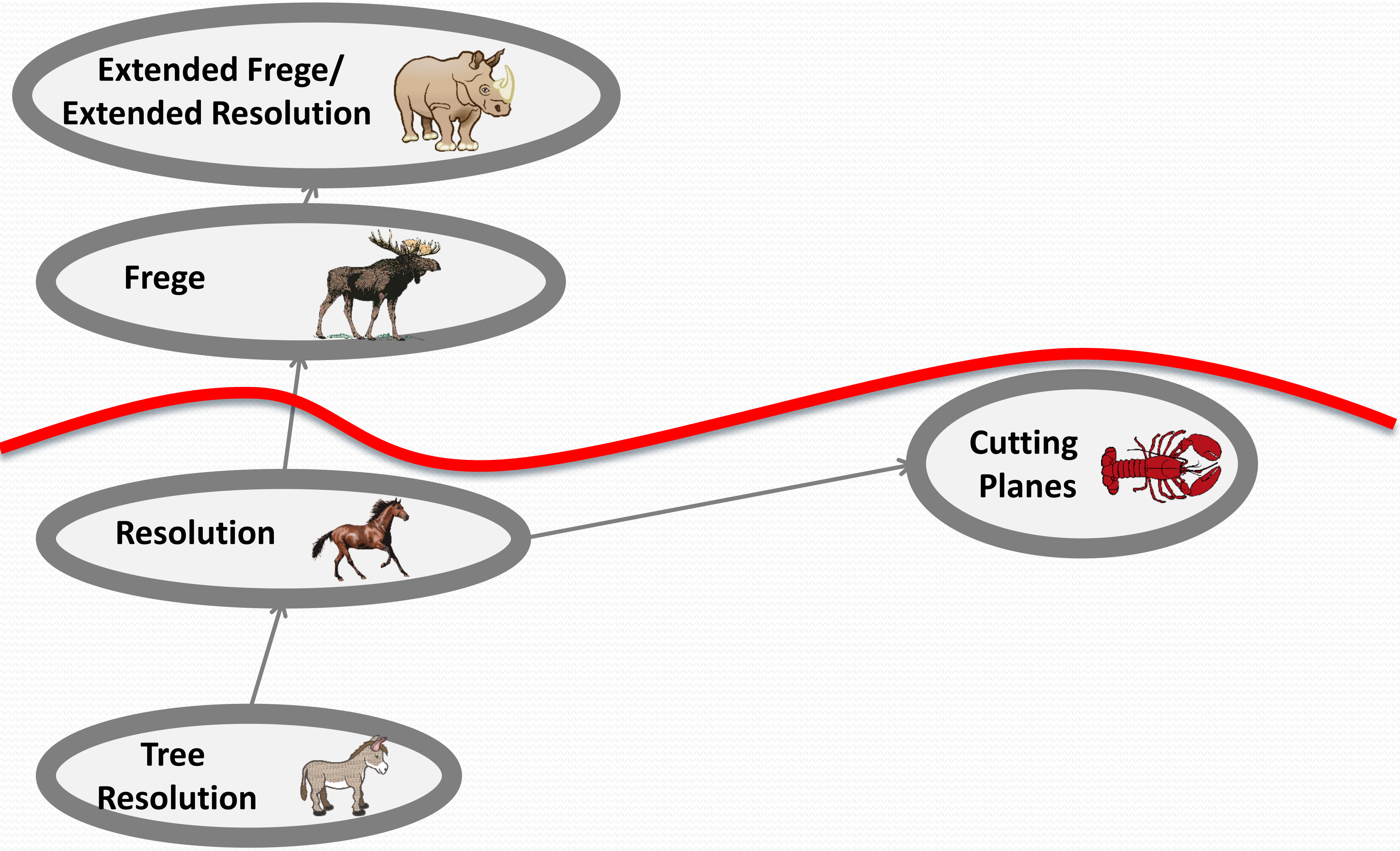
This is where ML could come into play: to help find short proofs when they exist
(but see Atserias/Müller’19 result…)

The Proof Complexity Zoo



From a talk by Toni Pitassi

The Proof Complexity Petting Zoo



Extended Frege/
Extended Resolution

Frege

Cutting
Planes

Resolution

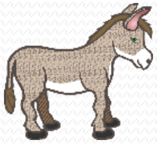
Tree
Resolution

Resolution proof system

- A refutation system: to prove a tautology, refute its negation.
- Start with a CNF formula: $\land$ of clauses ($\lor$ of x, $\neg x$).
- Apply the *resolution rule* (cut on a literal) until an empty clause is derived:

$$\frac{(x \lor C), (\neg x \lor D)}{(C \lor D)}$$

- Tree-like resolution: the graph of the refutation is a tree.

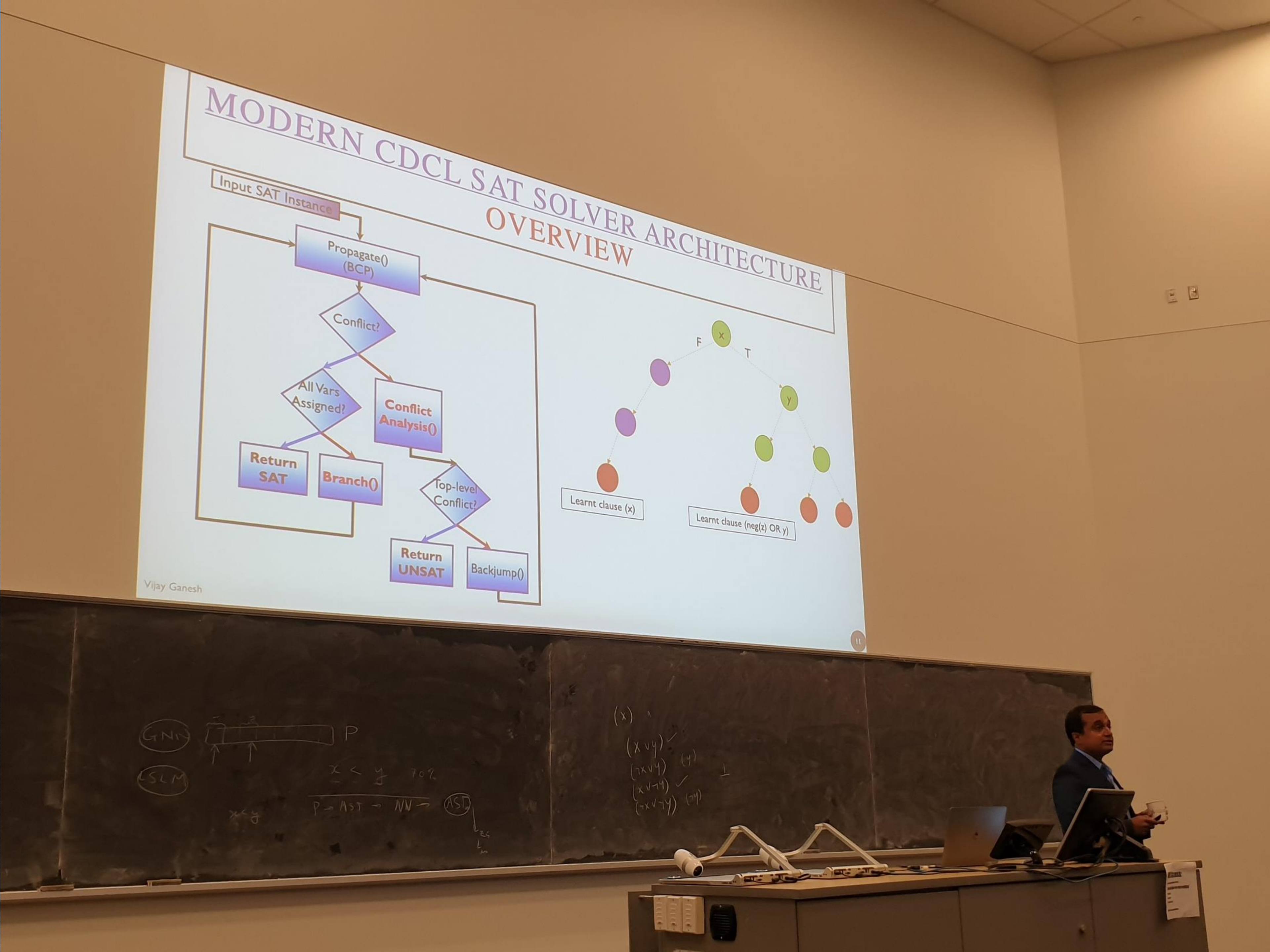


- Resolution proof system is sound and complete.
  - A propositional formula is unsatisfiable if and only if it has a resolution refutation

The proofs by CDCL solvers (with restarts)
are exactly characterized by Resolution!
[Pipatsrisawat/Darwiche'11, Atserias/Fichte/Thurley'11]

When I worked on resolution in 1980s I never thought it would become useful in practice.

Alasdair Urquhart



MODERN CDCL SAT SOLVER ARCHITECTURE OVERVIEW

Input SAT Instance

Propagate() (BCP)

Conflict?

All Vars Assigned?

Conflict Analysis()

Return SAT

Branch()

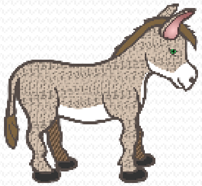
Top-level Conflict?

Return UNSAT

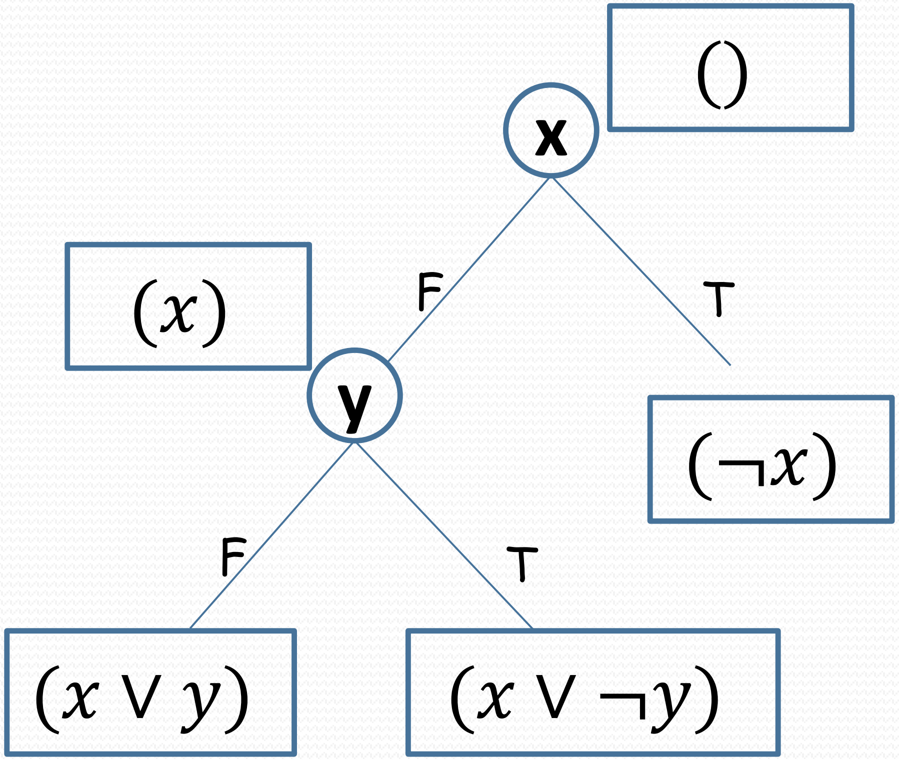
Backjump()

Vijay Ganesh

DPLL vs. tree-like resolution



- $(x \lor \neg\, y) \land (x \lor y) \land (\neg x)$: unsatisfiable
- Label leaves with violated input clauses



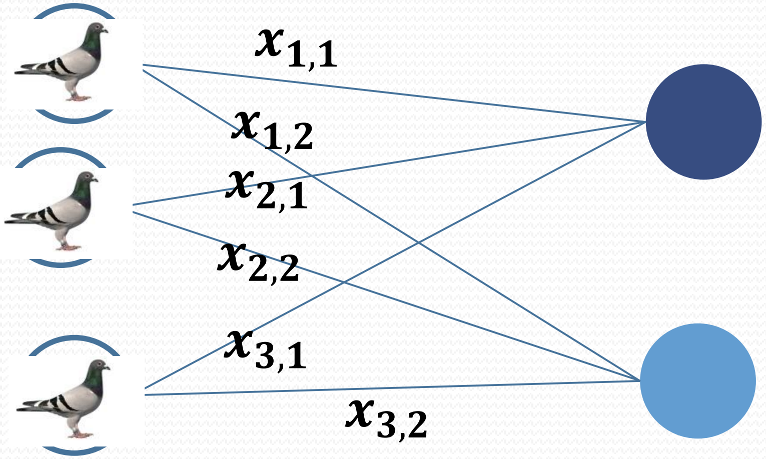
PigeonHole principle

- If n+1 pigeons all fly into n holes, some hole has at least two pigeons.

- There is no injective function from $\{1, \dots, n\}$ to $\{1, \dots, n-1\}$

PigeonHole Principle

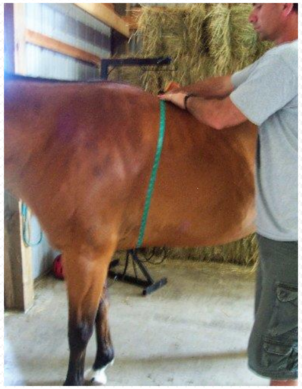
- If n+1 pigeons all fly into n holes, some hole has at least two pigeons.



- PHP: negation of the above
  - $(x_{1,1} \vee x_{1,2}) \wedge (x_{2,1} \vee x_{2,2}) \wedge (x_{3,1} \vee x_{3,2}) \wedge$
  - $(\neg x_{1,1} \vee \neg x_{2,1}) \wedge (\neg x_{1,1} \vee \neg x_{3,1}) \wedge (\neg x_{2,1} \vee \neg x_{3,1}) \wedge$
  - $(\neg x_{1,2} \vee \neg x_{2,2}) \wedge (\neg x_{1,2} \vee \neg x_{3,2}) \wedge (\neg x_{2,2} \vee \neg x_{3,2})$

- Requires exponential-size proofs in Resolution!

Size vs width



- Width of a proof of a formula f:
  - number of literals in a clause with most literals.

Theorem[Ben-Sasson/Wigderson 99]

Every size *s* resolution proof of a formula f can be converted to a resolution proof of f with width $\sqrt{2n\,log\,s} + \mathrm{width}(f)$

- Gives lots of lower bounds!
  - Random k-CNF, pebbling, etc...

SAT solvers that use just CDCL **cannot** solve all instances in polynomial time…

Even the best heuristic will take exponential time to produce an exponential-size proof.

What if there is a small proof? How hard is it to find it?

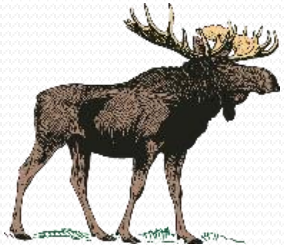
Theorem [Atserias/Müller’2019]

Automating resolution is NP-hard.

- Finding a resolution refutation at most polynomially longer than the shortest one is NP-hard.
- It is even NP-hard to distinguish formulas with polynomial-size proofs from formulas that only have exponential size proofs in resolution.
- In general, resolution is not automatizable in any complexity class C (such as C = subexponential time) unless $NP \subseteq C$

However, tree-like resolution is automatizable in quasi-polynomial time.

Frege proof systems

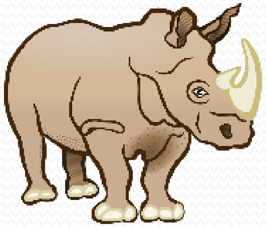


- Textbook-style proof system, natural deduction
  - Modus ponens as an inference rule.
  - A complete set of axiom schemas.
- Cuts on formulas

$$\frac{A, A \to B}{B}$$

- More powerful than Resolution: can prove PigeonHole principle with a polynomial-size proof.
  - For some tautologies, best known proof has size $n^{log\ n}$.
  - Running out of candidate hard tautologies...
- All Frege systems are equivalent [Cook/Reckhow’79]
- Automatizing Frege systems would break cryptography.

The power of extension



- **Extended resolution (extended Frege):**
  - add a rule $z = C$ $(z = F)$ where z is a new variable
  - and C is a clause (F is a subformula).
- Extended resolution and extended Frege are computationally equivalent
  - Correspond to cuts on circuits.
- Every example tautology people tried has a polynomial size extended Frege proof...
  - But no proof that every tautology does.

Metamathematics of P versus NP

- Independence of P versus NP?

  -Baker-Gil-Solovay

  -Razborov-Rudich

- Is P versus NP independent of Extended Frege?

Theorem (Razborov)
"SAT cannot be decided by polynomial-size circuits" requires superpoly-size proofs in resolution (even with k-DNFs instead of clauses)

- if pseudo-random generators exist.

Strange consequences of independence

- 1992: Ben-David/Halevi
  - If P vs. NP is independent of PA$_1$, Peano Arithmetic

    + all true sentences with unbounded only ∀ quantifiers (*)
  - Then SAT can be solved in "almost polynomial time"!

  - Proof idea:
    - Define a function $R(i) = \max_{j<i} \{\min |x| : SAT(x) \neq M_j(x)\}$, where M_j are polynomial-time Turing machines.
    - PA$_1$ does not prove $P \neq NP$, $R^{-1}(n)$ is very slow-growing (e.g., $< \log^* n$)
    - By definition, for every satisfiable formula of size n, one of the first $\log^* n$ TMs will find a satisfying assignment; total runtime $< O(n^{\log^*})$.

(*) Quoting BD/H'92: "All current techniques for proving independence of Peano Arithmetic imply independence of PA$_1$"

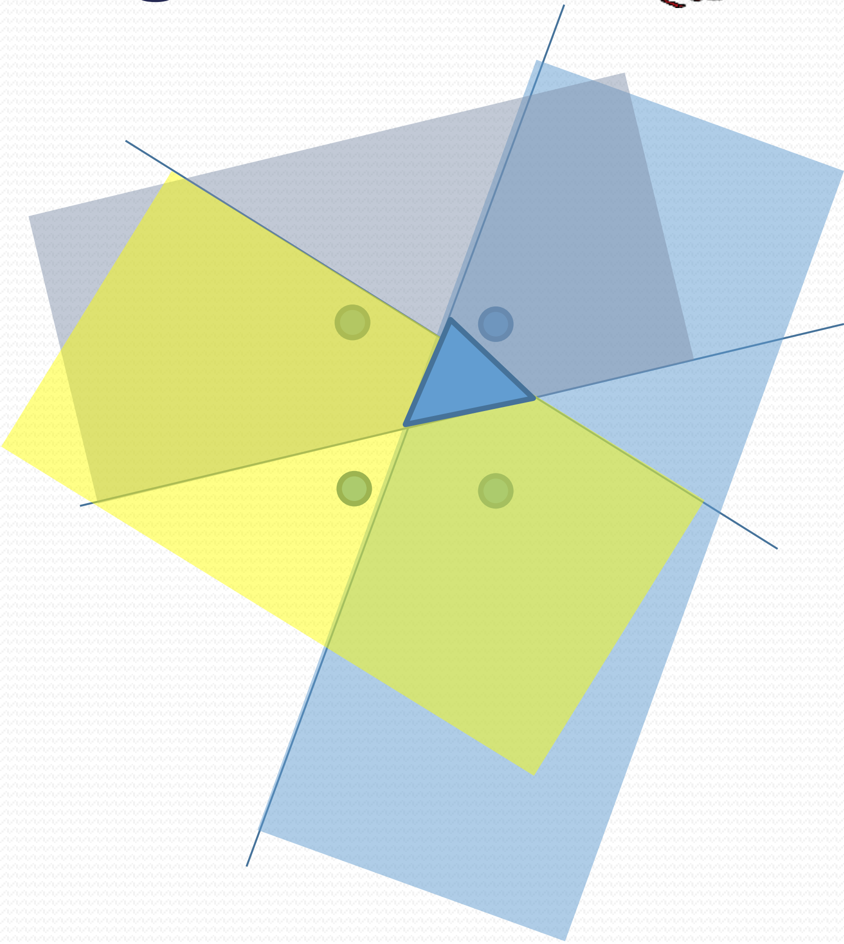
Is the best way to solve constraint satisfaction problems then to encode them as formulas and run SAT solvers?

What about, eg, writing them as systems of equations and using integer linear programming/cutting planes?

Does going to non-Boolean domains help?

Cutting planes: semi-algebraic

- A proof system operating with integer inequalities.
- Rules:
  - addition, multiplication,
  - division by a positive integer with rounding.
- A clause $(x \vee \neg y \vee z)$ becomes inequality $x + (1 - y) + z \geq 1$
- Also have $x \leq 1, x \geq 0$
- Want to derive $0 \geq 1$

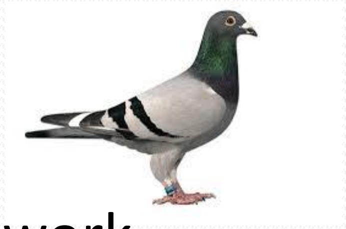


Short proof of the PigeonHolePrinciple.

Exponential lower bound for CliqueColor [Pudlak'97].

Pseudo-Boolean solvers

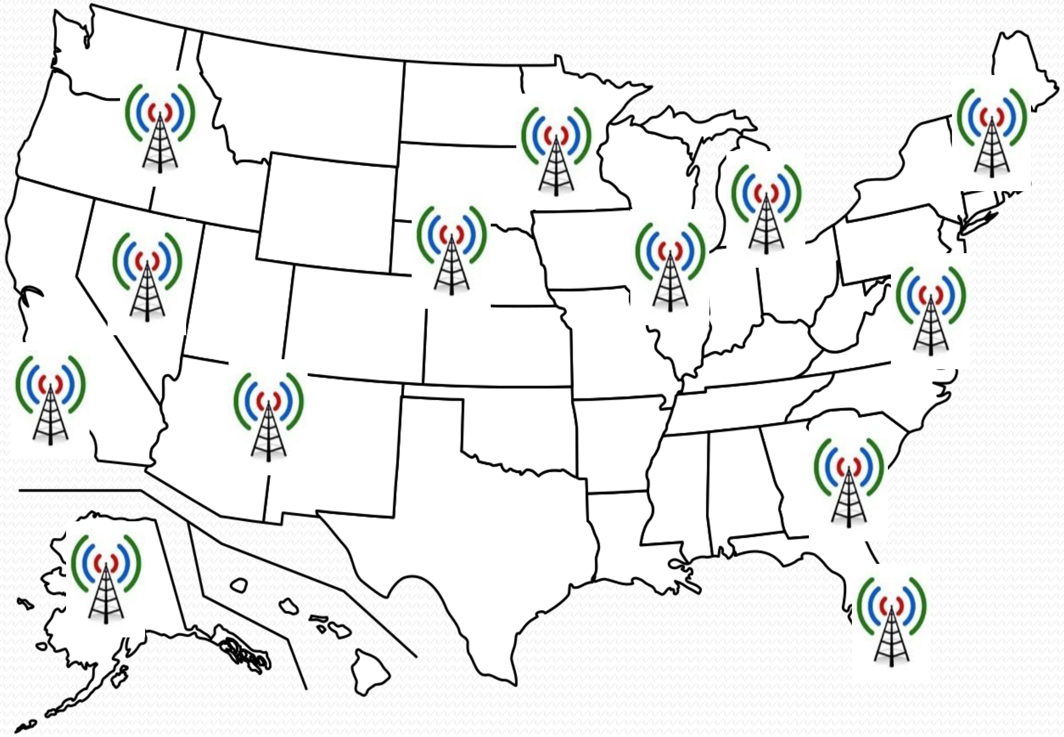
- CDCL-based solvers
- + “cardinality constraints”: $x_2 + x_5 + x_{10} \geq 5$
- Some version of Cutting Planes axioms



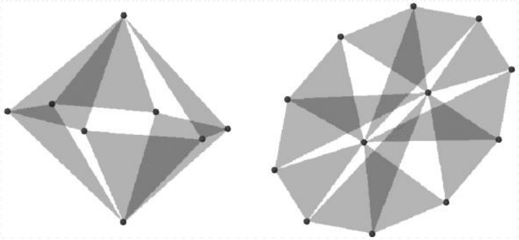
- PHP encoded propositionally:
  - As hard as always: lower bounds for resolution work
- PHP encoded by inequalities/cardinality constraints:
  - Short polynomial-size proof!

SAT vs. Integer linear programming

- FCC spectrum auction:
  - Essentially a colouring problem
  - ILP: poor
  - SAT: good
- TravelingSalesperson:
  - ILP: good
  - SAT: poor

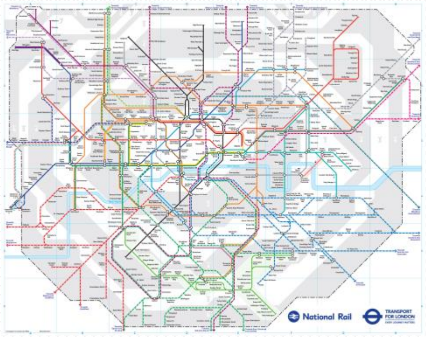


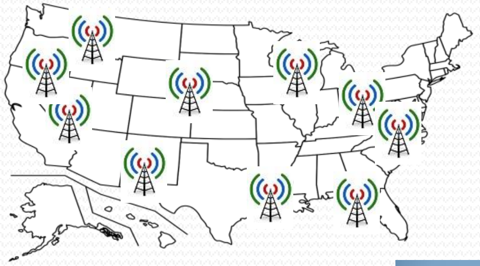
Practical problems rarely start propositional

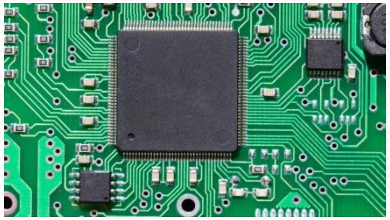


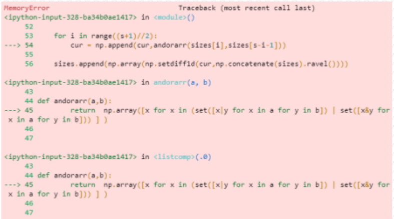
They talk about programs, circuits, graphs, numbers, strings...

Domains with a lot of inherent structure!





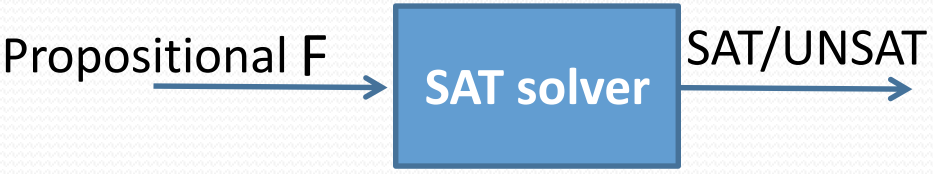




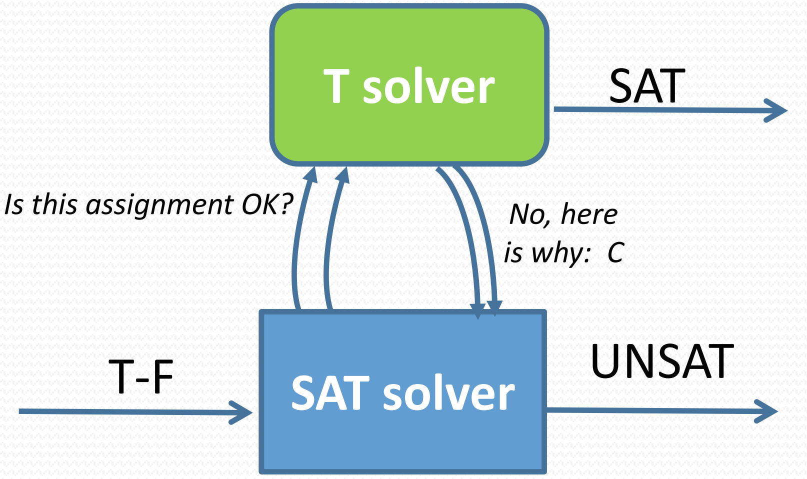
```
MemoryError                               Traceback (most recent call last)
<ipython-input-328-ba34b0ae1417> in <module>()
     52
     53     for i in range((s+1)//2):
---> 54         cur = np.append(cur,andorarr(sizes[i],sizes[s-i-1]))
     55
     56     sizes.append(np.array(np.setdiff1d(cur,np.concatenate(sizes).ravel())))

<ipython-input-328-ba34b0ae1417> in andorarr(a, b)
     43
     44 def andorarr(a,b):
---> 45         return  np.array([x for x in (set([x|y for x in a for y in b]) | set([x&y for
 x in a for y in b])) ] )
     46
     47

<ipython-input-328-ba34b0ae1417> in <listcomp>(.0)
     43
     44 def andorarr(a,b):
---> 45         return  np.array([x for x in (set([x|y for x in a for y in b]) | set([x&y for
 x in a for y in b])) ] )
     46
     47
```

Satisfiability modulo theories (SMT): best of both worlds?

- Use whichever atoms are most convenient:
  - variables
  - equalities
  - inequalities...

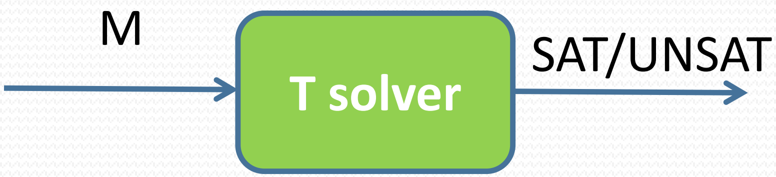


- Alternate between
  - treating atoms as propositional variables
  - checking that an assignment makes sense given atoms' meaning.



Reasoning within a domain: theories

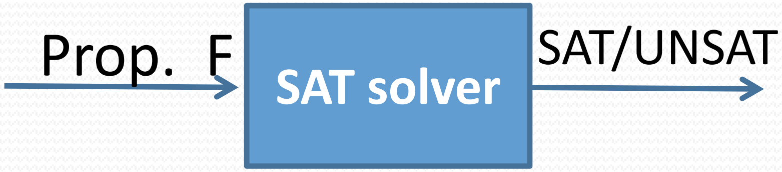
- T is a (first-order) theory over some signature L.
  - Focus on quantifier-free fragment of T
  - Usually have =
- M is a conjunction of atoms of T (and their negations)
- Mainly want (efficiently) decidable theories

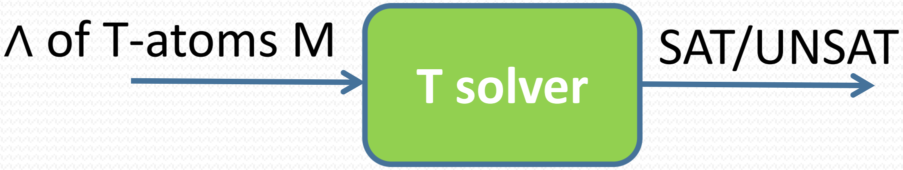


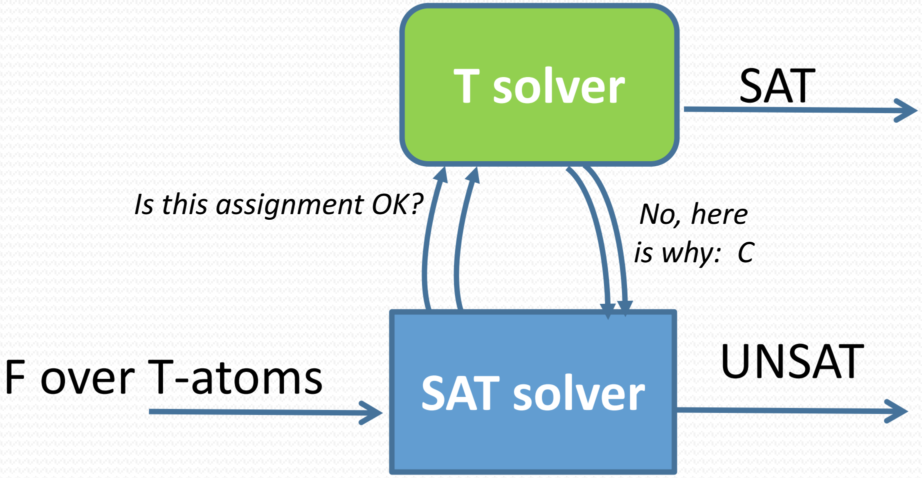
- Linear equations:
  - Atoms: lin. inequalities
  - M: system of lin. ineqs
    - $2x_1 + x_2 \leq 5$
    - $x_1 - 3x_2 \leq 7$
- Theory of equality:
  - Atoms: $a = b, b \neq c$
  - M: $a = b, b = c, a \neq \mathrm{c}$

- *Want to reason about any Boolean combination of theory atoms:* $(x = 0 \lor x = 1) \land x + y > 2 \rightarrow y > 1$

Satisfiability modulo theories (SMT)







- “Lazy” SMT

SAT solving is to resolution as

SMT solving is to... ?

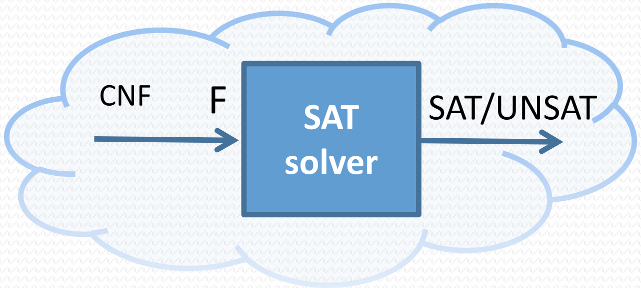
SAT solving is to resolution as

SMT solving is to resolution modulo theories: Res(T)

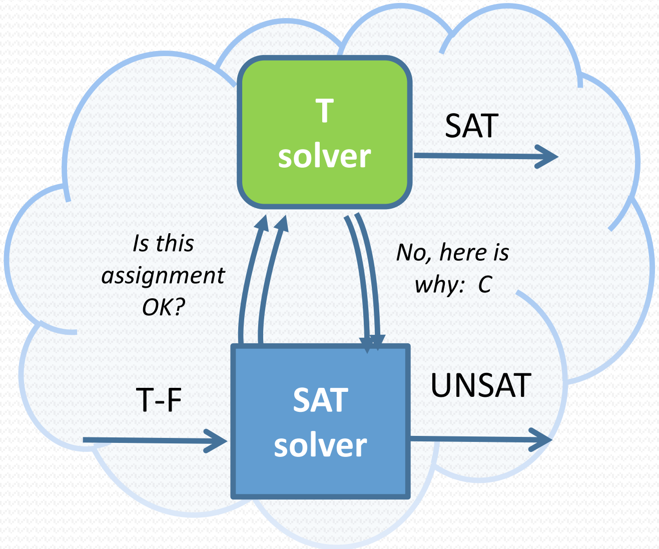
Joint work with Vijay Ganesh and Robert Robere

For which theory T would CDCL(T) correspond to Extended Frege?

Resolution modulo theories



Resolution



Resolution modulo T

Literals are atoms of the theory.
Rules of inference:

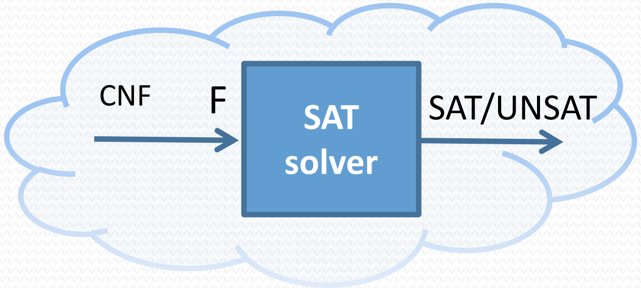
1. Resolution rule

$$\frac{(C \vee x) \quad (D \vee \neg x)}{(C \vee D)}$$

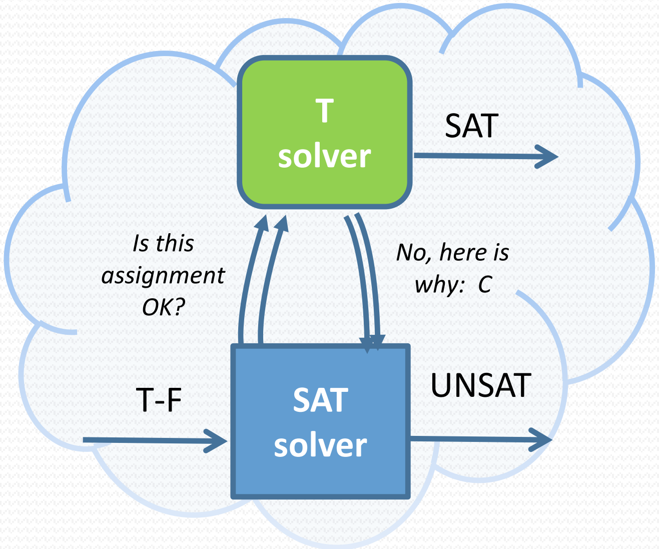
2. Clauses derivable from T

- Eg: T is a theory of equality:
  - $(a \neq b \vee b \neq c \vee a = c)$
- Eg: T is linear arithmetic:
  - $(a \leq c \ \vee \ b \leq d \vee a + b > c + d)$

Resolution modulo theories



Resolution



Resolution modulo T

Literals are atoms of the theory.
Rules of inference:

1. Resolution rule

$$\frac{(C \vee x) \quad (D \vee \neg x)}{(C \vee D)}$$

2. Clauses derivable from T

Res(T):

$$C,\ where\ T \vdash C$$
$$vars(C) \subseteq vars(F)$$

Res*(T):

$$C,\ where\ T \vdash C$$
$$vars(C) \subseteq atoms(T)$$

Main theorem

Let T be a theory, and F an unsat. formula over atoms of T. Then

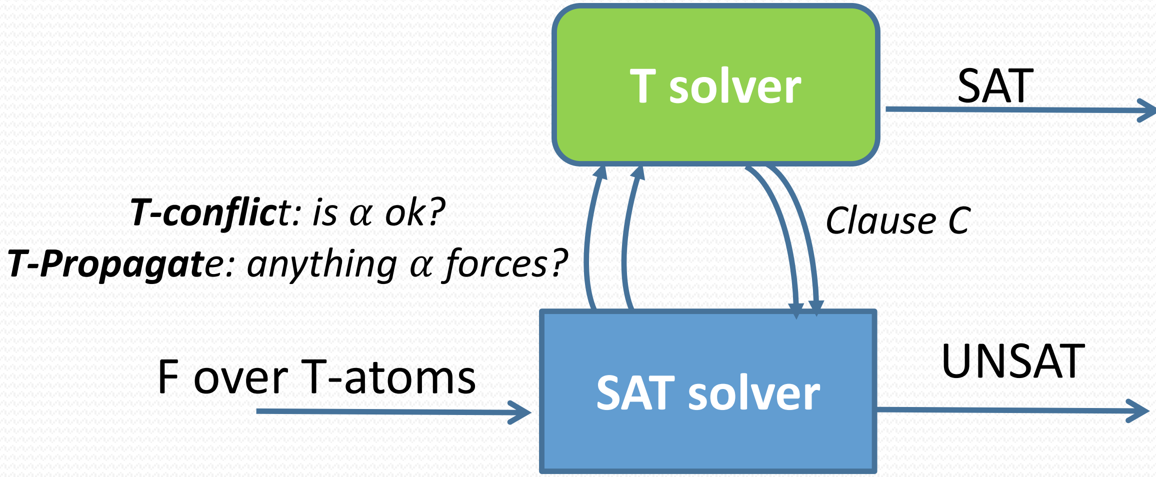
- an SMT solver produces a **Res(T)** refutation of **F.**
- an SMT solver with asserting learning scheme and non-deterministic branching can efficiently simulate **Res(T).**
- When theory solvers can introduce new literals, same statements hold for **Res*(T)** in place of **Res(T).**

CDCL, CDCL(T) and CDCL*(T)

- CDCL:
  - Repeat:
    - **Decision**: set a variable
    - **Propagate** unit clauses
    - If there is a **conflict**, analyse it and learn
    - Maybe **restart**

- CDCL (T):
  - Repeat:
    - **Decision**
    - **Propagation** and **T-Propagation**
    - **Conflict analysis** and **T-conflict**
    - Maybe **restart**

- **T-Conflict:**
  - If a partial assign. α is inconsistent with T,
  - Learn a clause $C \subseteq \neg\alpha$
- **T-propagation:**
  - Partial assign α,
  - $T \vdash \alpha \rightarrow l$
  - Learn a clause $C \subseteq (l \vee \neg\alpha)$


T solver
SAT
__T-conflict__: is α ok?
__T-Propagate__: anything α forces?
Clause C
F over T-atoms
SAT solver
UNSAT

CDCL, CDCL(T) and CDCL*(T)

- CDCL:
  - Repeat:
    - **Decision**: set a variable
    - **Propagate** unit clauses
    - If there is a **conflict**, analyse it and learn
    - Maybe **restart**

- Resolution captures CDCL
  - Pipatsrisawat/Darwiche
  - Atserias/Fichte/Thurley.

- CDCL (T): CDCL plus
  - Repeat:
    - **Decision**
    - **Propagation** and **T-Propagation**
    - **Conflict analysis** and **T-conflict**
    - Maybe **restart**

- CDCL*(T):
  - T-conflict and T-propagation can introduce new literals

- Res(T) captures CDCL(T)
- Res*(T) captures CDCL*(T)
  - Generalizing PD'09, AFT'09

CDCL ≈ Resolution: [PD’09,AFT09]

- Consider a general resolution proof $\Pi = C_1, \dots, C_m$
  - There exists a solver run
    - Sequence of decisions and restarts
  - Such that every clause in Π is (implicitly) learned:
    - A clause $C = (l_1 \vee \cdots \vee l_k)$ is **absorbed** wrt set of clauses S
    - if whenever all l_i except one are set to false, the remaining literal is set by unit-propagation from S.
    - Eg: $(x \vee y \vee z), (\neg z \vee u), (\neg u \vee w)$ absorbs $(x \vee y \vee w)$
  - In polynomial number of steps.

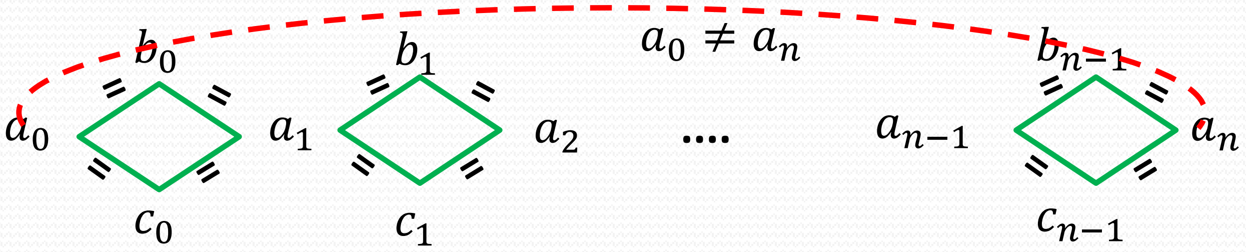
CDCL(T) ≈ Res(T) and CDCL*(T) ≈ Res(T)

- Consider a ~~general resolution~~ Res(T)/Res*(T) proof $\Pi = C_1, \dots, C_m$
  - There exists a solver run
    - Sequence of decisions and restarts and theory clauses
      - Produce theory clauses first
  - Such that every clause in Π is (implicitly) learned:
    - A clause $C = (l_1 \vee \cdots \vee l_k)$ is **absorbed** wrt set of clauses S
    - if whenever all l_i except one are set to false, the remaining literal is set by unit-propagation from S.
    - Eg: $(x \vee y \vee z), (\neg z \vee u), (\neg u \vee w)$ absorbs $(x \vee y \vee w)$
    - Enough to use T-Propagation rule
  - In polynomial number of steps.

New literals

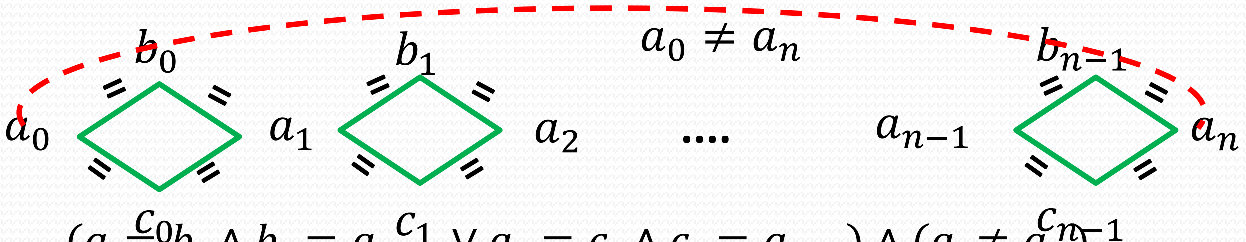
- Theory solver might return a clause with literals not in F:
  - F: $(a = b \vee b = \ \ c) \wedge (b = c \vee c = d) \wedge (a \neq \mathrm{d})$
    - $(x_1 \vee x_2) \ \wedge (x_2 \vee x_3) \wedge (\neg\, x_4)$
  - T returns a clause $(a \neq b \vee b \neq c \vee a = c)$, $a = c$ not in F.

- Diamond equalities [Bjorner, Dutertre, de Moura]
  - Over theory of equality.
  - Hard for resolution if not allowed to introduce new literals.
  - Easy if T introduces literals $a_i \,=\, a_{i+1}$



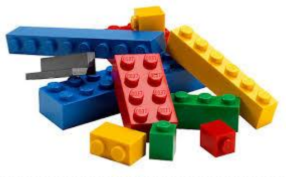
  - $\bigwedge_{i<n} (a_i = b_i \wedge b_i \,=\, a_{i+1} \ \vee a_i \,=\, c_i \wedge c_i \,=\, a_{i+1}) \wedge (a_0 \neq\, a_n)$

New literals

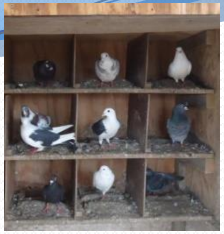
- Diamond equalities [Bjorner, Dutertre, de Moura]
  - 

  - $\bigwedge_{i<n} (a_i = b_i \wedge b_i = a_{i+1} \vee a_i = c_i \wedge c_i = a_{i+1}) \wedge (a_0 \neq a_n)$
- **Theorem** [Hadarean, Horn, King'15]
  - Suppose an unsatisfiable formula F over T has t "critical" assignments: each corresponding to a different theory conflict.
  - Then a CDCL(T) solver needs to learn $\geq t$ theory clauses.
  - **Corollary**: then any Res(T) proof has size $\geq t$
- In diamond equalities, each path from a_0 to a_n is a different critical assignment
  - So need $\geq 2^n$ theory solver calls
  - But if can derive $a_i = a_j$, then polynomial time.

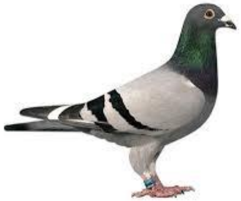
The power of Res*(T)



PigeonHolePrinciple



- PigeonHole Principle: there is no injective function from [n] to [n-1]



- PHP:

$$\wedge_{i\leq n}(\vee_{j<n}\ p_{i,j}) \wedge \bigwedge_{i\neq k,j}(\neg\, p_{i,j} \vee \neg p_{k,j})$$

- =-PHP:

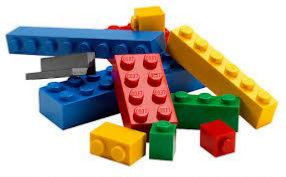
$$\bigwedge_{i\leq \mathrm{n}}(\vee_{j<n}\,(p_i = h_j)\,) \wedge \bigwedge_{i<k\leq \mathrm{n}}(p_i \neq p_k)$$

- EUF-PHP:

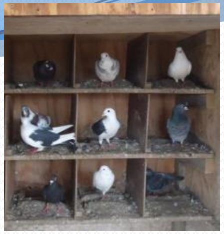
$$\bigwedge_{x\in[n]}(f(x)\neq 0)\ \wedge \bigwedge_{x,y\in[n]}(x\neq y\rightarrow f(x)\neq f(y))$$

- LA-PHP:

$$\bigwedge_{i\leq \mathrm{n}}(\Sigma_{j<n}\, x_{i,j}\geq 1)\ \wedge \bigwedge_{j<\mathrm{n}}(\Sigma_{i\leq n}\, x_{i,j}\leq 1)$$



PigeonHolePrinciple



- PigeonHole Principle: there is no injective function from [n] to [n-1]
- PHP:

$$\bigwedge_{i\leq n}(\bigvee_{j<n}\ p_{i,j}) \wedge \bigwedge_{i\neq k,j}(\neg p_{i,j} \vee \neg p_{k,j})$$

- =-PHP:

$$\bigwedge_{i\leq \mathrm{n}}(\bigvee_{j<n}(p_i = h_j) \wedge \bigwedge_{i<k\leq \mathrm{n}}(p_i \neq p_k)$$

- EUF-PHP:

$$\bigwedge_{x\in[n]}(f(x) \neq 0) \ \wedge \bigwedge_{x,y\in[n]}(x \neq y \to f(x) \neq f(y))$$

- LA-PHP:

$$\bigwedge_{i\leq \mathrm{n}}(\Sigma_{j<n}\, x_{i,j} \geq 1) \ \wedge \bigwedge_{j<\mathrm{n}}(\Sigma_{i\leq n}\, x_{i,j} \leq 1)$$

- Propositional
- Theory of equality:
  - $(a = b \wedge b = c \to a = c)$
- Equality with uninterpreted functions (EUF)
  - equality axioms
  - Ackermann axioms: $(a = b \to f(a) = f(b))$
- Linear arithmetic

Power of Res(T)

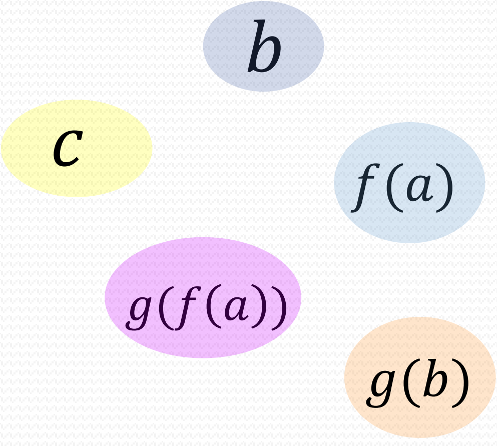
- Res(Theory of Equality) is no more powerful than Resolution
  - Add all n^3 equality axioms to F, then solve.

- Res(Linear Arithmetic) polynomially simulates R(lin)

- Resolution over Equality with Uninterpreted Functions theory, Res(EUF), can effectively p-simulate Frege.
  - Even though conjunctions of EUF atoms are decidable in $O(n \log n)$ time!
    - Using a variant of Union-Find algorithm.
  - E-Res calculus of [Bjorner, de Moura] is already enough
    - Res*(EUF) effectively simulates E-Res

Equality with uninterpreted functions theory (EUF)

- Signature:
  - uninterpreted function symbols of bounded arity
  - constants a, b, c...
- Terms: constants, and inductively $f(\bar{t})$ for functions.
- Atoms: equalities/disequalities over terms: $t_1 = t_2,\ t_1 \neq t_2$
- Formulas: conjunctions of atoms



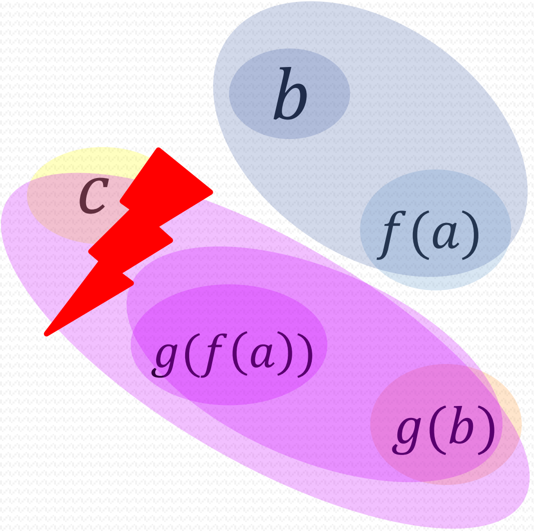
$$(f(a) = b) \wedge (g(b) = c) \wedge (g\big(f(a)\big) \neq c)$$

- Axioms:
  - Equality: $(a = b \wedge b = c \rightarrow a = c)$
  - Ackermann: $\bar{a} = \bar{b} \ \rightarrow \ f(\bar{a}) = f(\bar{b})$

- Can decide very efficiently (time $O(n \log n)$) if a given EUF formula is satisfiable:
  - Downey-Sethi-Tarjan congruence closure (based on Union-Find)

Equality with uninterpreted functions theory (EUF)

- Signature:
  - uninterpreted function symbols of bounded arity
  - constants a, b, c...
- Terms: constants, and inductively $f(\bar{t})$ for functions.
- Atoms: equalities/disequalities over terms: $t_1 = t_2,\ t_1 \neq t_2$
- Formulas: conjunctions of atoms



$$(f(a) = b) \wedge (g(b) = c) \wedge (g(f(a)) \neq c)$$

- Axioms:
  - Equality: $(a = b \wedge b = c \rightarrow a = c)$
  - Ackermann: $\bar{a} = \bar{b} \ \rightarrow \ f(\bar{a}) = f(\bar{b})$

- Can decide very efficiently (time $O(n \log n)$) if a given EUF formula is satisfiable:
  - Downey-Sethi-Tarjan congruence closure (based on Union-Find)

Sequent calculus (LK)

- Equivalent to Frege systems.
  - Natural deduction

- Sequents: $A_1, \dots, A_n \quad \longrightarrow \quad B_1, \dots, B_m$
  - $A_1 \wedge \cdots \wedge A_n \rightarrow B_1 \vee \cdots \vee B_m$
  - Axioms $A \ \rightarrow A, 0 \ \rightarrow S,\ S \ \rightarrow 1.$
  - Rules for ∨,∧, ¬ and cut

$$\frac{F \rightarrow G, A \qquad\qquad A, F \rightarrow G}{F \rightarrow G}$$

$$\frac{F \rightarrow G, A}{\neg A, F \rightarrow G}$$

$$\frac{F \rightarrow G, A \qquad\qquad F \rightarrow G, B}{F \rightarrow G\ , A \wedge B}$$

$$\frac{A, B, F \rightarrow G}{A \wedge B, F \rightarrow G}$$

- Proof size: total number of symbols.

Res(EUF) simulates LK

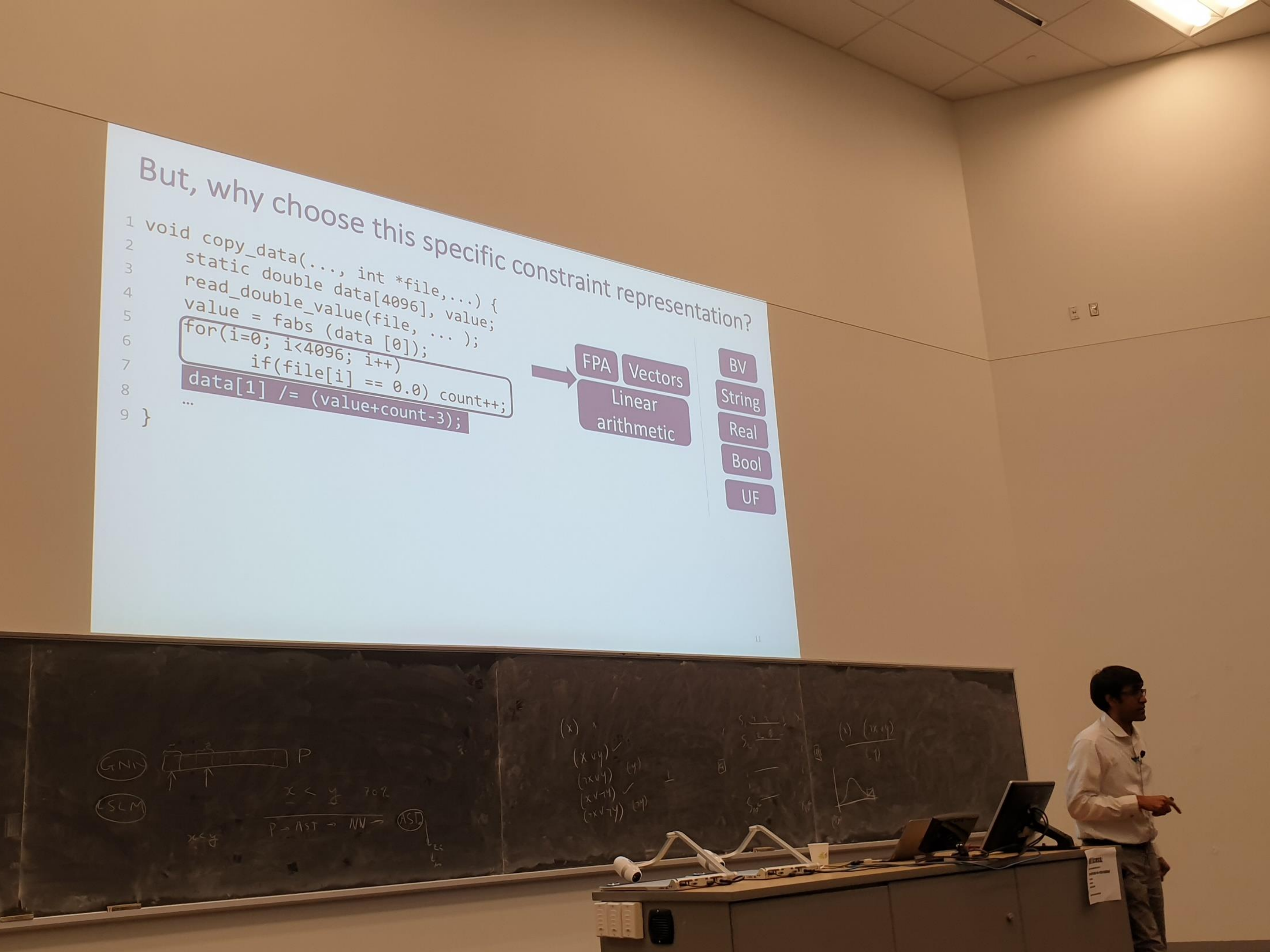
- Suppose there is an LK proof of $F \to 0$
  - An LK-refutation of F
- Add to F:
  - Two constants: $e_0 \neq e_1$
  - Definitions of N, O, A (and, or, not):
    - $N(e_0) = e_1, N(e_1) = e_0, O(e_1, e_0) = e_1$,….
  - Bounded variable range: $\wedge(x_i = e_0 \vee x_i = e_1)$

- Now simulate an LK proof by constructing terms for all formulas in the proof inductively
  - Prove that at each step of LK proof: $A_1 \ldots A_k \to B_1 \ldots B_\ell$
  - Either one of the A terms is e_0 or one of the B terms is e_1
    - Also for each subformula in proof so far, its term $= e_0$ or $= e_1$

For which theory T would Res*(T) effectively p-simulate Extended Frege?

Didn’t you say EUF instances are usually flattened?

- Arnold Beckmann’s observation: just add a flattening rule to Res*(EUF)!
  - Flattening: every time a new term is introduced, add a new variable for this term.
    - Do not need to decide when to add extension variables!



But, why choose this specific constraint representation?

```
void copy_data(..., int *file,...) {
    static double data[4096], value;
    read_double_value(file, ... );
    value = fabs (data [0]);
    for(i=0; i<4096; i++)
        if(file[i] == 0.0) count++;
    data[1] /= (value+count-3);
    …
}
```

FPA

Vectors

Linear arithmetic

BV

String

Real

Bool

UF

11

Eager vs. Lazy SMT

Question. Is it better to use a theory solver as an oracle, or just bit-blast all the way to propositional SAT instance?

- Ackermann Reduction maps EUF formulas of size m to SAT instances of size $O(m^2)$

Theorem. Assuming the **Exponential Time Hypothesis** (SAT requires $2^{\Omega(n)}$ time), **any** reduction from EUF to SAT requires a blow-up of Ω(m log m)**.**

Remark. This is tight!

Lots of open problems

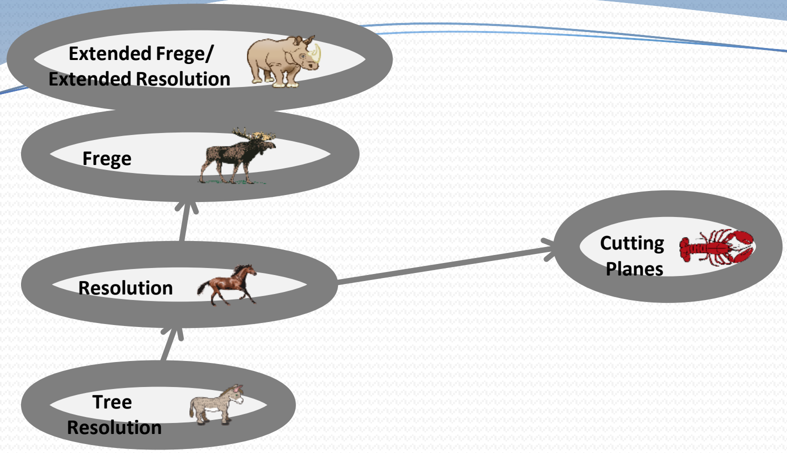
- Upper/lower bounds on Res(T)/ Res*(T) for a variety of theories.
- Proof complexity of model checking and first-order provers?
- Knowledge compilation:
  - How to choose T given a problem and class of instances?
  - In particular, when to choose Eager SMT and when Lazy?
  - And how to choose T-representation?

Given an instance of a problem, what is the best way to state it to make it easier to solve?

- How to even compare representations of the same problem in different underlying languages?
  - Is Marc’s and Jakob’s PseudoBoolean encoding of PHP the “same PHP” as the classic CNF encoding?
    - Work in progress

ML for choosing the right representation and heuristics?

- Kevin Leyton-Brown:
  - Empirical hardness models
    - Predict runtime for a specific instance from this instance’s features.
    - Uses random forests/regression trees
  - Application: automatic algorithm configuration
    - SATzilla, Hydra…
- Can something like this be done for representations?



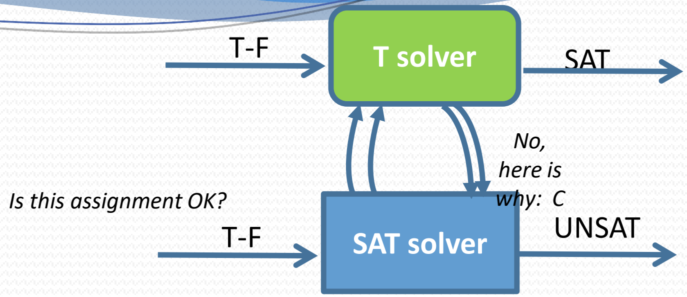
Extended Frege/
Extended Resolution

Frege

Resolution

Cutting
Planes

Tree
Resolution



T-F

T solver

SAT

No,
here is
why: *C*

Is this assignment OK?

T-F

SAT solver

UNSAT

Thank you!