Automated Reasoning

Peter Baumgartner

NICTA, Canberra

and

RSISE, ANU

Contact
Office: NICTA Bldg, Tower A, 7 London Circuit
Email: Peter.Baumgartner@nicta.com.au
Web: http://users.rsise.anu.edu.au/~baumgart/

Many slides based on material from Scott Sanner

Schedule

- Introduction to Logic (John Slaney)
  - Starting week of March 3
- **Automated Reasoning (Peter Baumgartner)**
  - **Starting week of March 17**
- SAT solving (Anbulagan)
  - Starting week of March 31
- Knowledge Compilation (Jinbo Huang)
  - Starting week of April 28
- Temporal Logic (Michael Norrish)
  - Starting week of May 12
- Higher-Order Logic (Jeremy Dawson)
  - Starting week of May 26

Automated Reasoning

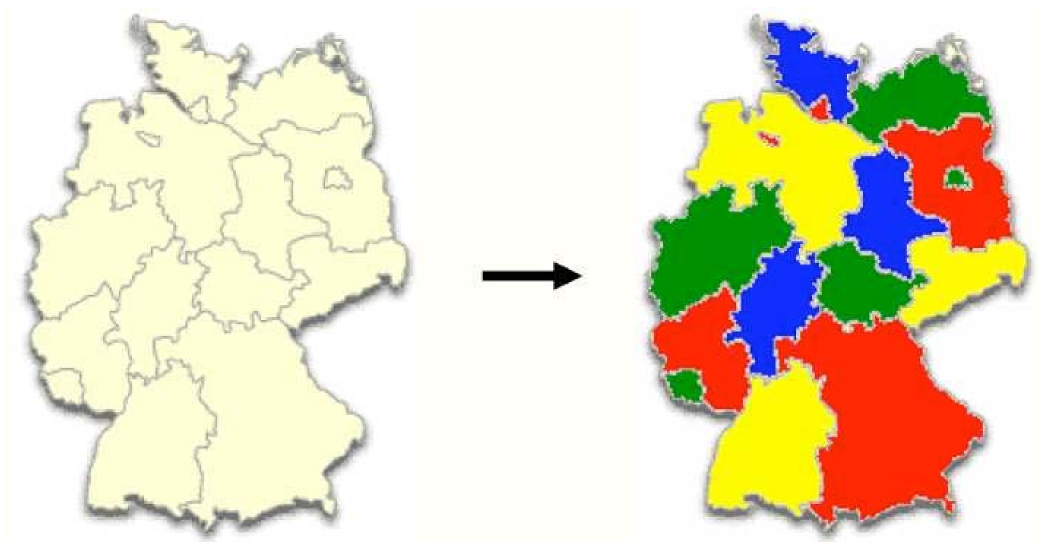
- ... vs. **calculation:**
  - Problem: $2^2 = ?$ $3^2 = ?$ $4^2 = ?$
  - "Easy", often polynomial

- ... vs. **constraint solving**:
  - Problem spec: $x^2 = a$ where $x \in [1 .. b]$
  - Problem instance: fix parameter values a and b: $a = 16$, $b = 10$
  - Find satisfying values then for variable x (from finite domain)
  - "Difficult", often exponential (NP-complete problems)

- ... is, among others, about (first-order logic) **theorem proving**:
  - Problem: $\exists x\, (x^2 = a \land x \in [1 .. b])$
  - Is is satisfiable? valid?
  - "Very difficult" (often undecidable)

Logical Analysis of Systems

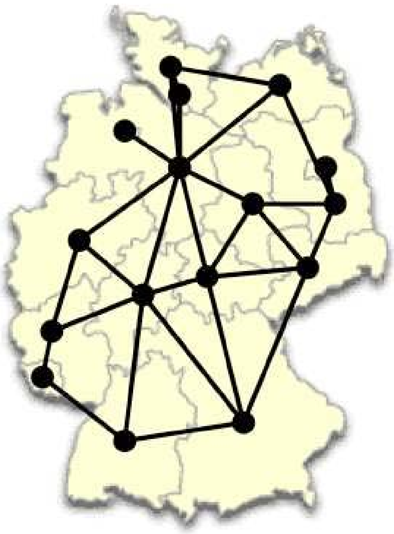
Logical Analysis Example: Three-Coloring Problem



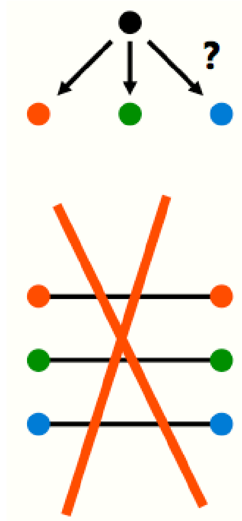
Problem: Given a map
Can it be colored with only three colors?

Three-Coloring Problem - Graph Theory Abstraction

Problem Instance



Problem Specification



Three-Coloring Problem - Formalization

- **Every node has at least one color**

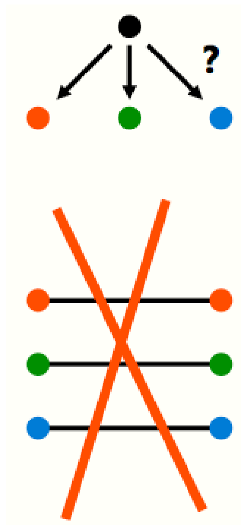
$$\forall N\ (\mathrm{red}(N) \lor \mathrm{green}(N) \lor \mathrm{blue}(N))$$

- **Every node has at most one color**

$$\begin{aligned}\forall N\ ((&\mathrm{red}(N) \rightarrow \neg\mathrm{green}(N)) \land \\ (&\mathrm{red}(N) \rightarrow \neg\mathrm{blue}(N)) \land \\ (&\mathrm{blue}(N) \rightarrow \neg\mathrm{green}(N)))\end{aligned}$$

- **Adjacent nodes have different color**

$$\begin{aligned}\forall M, N\ (\mathrm{edge}(M, N) \rightarrow (&\neg(\mathrm{red}(M) \land \mathrm{red}(N)) \land \\ &\neg(\mathrm{green}(M) \land \mathrm{green}(N)) \land \\ &\neg(\mathrm{blue}(M) \land \mathrm{blue}(N))))\end{aligned}$$



Three-Coloring Problem - Solving Problem Instances ...

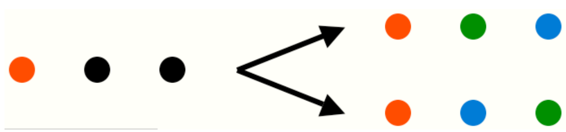
- ... with a **constraint solver**
  - Let constraint solver find values for variables such that spec is satisfied.
    - Variables: colors of nodes in the graph
    - Values: red, green or blue
- ... with a **first-order logic theorem prover**
  - Let the theorm prover prove that the three-colouring formula (see previous slide) + specific graph (as a formula) is satisfiable
- To solve problem instances, a constraint solver is usually much more efficient than a first-order theorem prover (e.g. use a propositional SAT solver)
  - Theorem provers are not even guaranteed to terminate on such problems!

What is the role of theorem proving then?

Three-Coloring Problem: The Role of Theorem Proving

- Functional Dependencies
  - The blue coloring functionally depends on the red and green coloring

  - The blue coloring does not functionally depend on the red coloring



- Theorem proving tasks: are the following valid (expressed as formulas)?
  - The blue coloring functionally depends on the red and green coloring
  - The blue coloring functionally depends on the red coloring
- (Learning about functional dep. might be instructive for modeller and solver)
- These are "proper" theorem proving tasks: analysis wrt **all** instances
- Demo now, files can be downloaded from my web page

Abstracting from the Example

- **AR systems functionality**
  - **Input**: a set of formulas in a specific **logical language**
  - **Run**: analyze these formulas by **logical inference** for a specific **task**
  - **Output**: the result of the analysis (proof, counterexample, solution...)
- **Rationale** - deduction: the "ultimate declarative paradigm"
  - Formulas describe possible worlds
  - Draw conclusions by (sound) logical inference
  - Learn something about the "real world"
- **Logical language and semantics**
  - Propositional, first-order, higher-order, modal, description logic, ...
  - monotonic/non-monotonic, probabilistic, ressource-bounded, ...
- **Logical Inference and task**
  - Calculus (Resolution, ...) -> Proof procedure -> Implementation
  - Prove theorem, disprove conjecture, plausible explanation, find a model,...

History

- **Pre-computer era**
  - Early: Aristotle, Leibniz
  - 19th century: Boole, DeMorgan, Peano, Cantor and others
  - 20th century: Hilbert, Skolem, Herbrand, Gödel, Gentzen, Church
- **Computer era** (among many others)
  - 1960s Calculi
    - Davis-Putnam-Logemann-Loveland (DPLL), Resolution, Model Elimination
  - 1970s Logic programming
    - Prolog
  - 1980s Knowledge representation
    - Description Logics
  - 1990s Modern theory of resolution
  - 2000s (Serious) applications

Applications

- **Proofs of Mathematical Conjectures**
  - Graph theory: Four color theorem
  - Boolean algebra: Robbins conjecture
- **Verification**
  - Hardware: arithmetic units correctness
  - Software: functional correctness, safety properties, static checking
- **Query Answering**
  - Build domain-specific knowledge bases,
    use theorem proving to answer queries
- **Key to Success**
  - Chose your logic and calculus carefully (e.g. avoid undecidable logic if possible)
  - Need domain-specific optimizations (e.g. avoid successor-arithmetic)
  - Domain-independent optimization (e.g. "subsumption", good data structures)
- **Next**: preview of some logics and calculi

Example of Propositional Logic Sequent Proof

- **Given**:
  - **Axioms**:
    None
  - **Conjecture**:
    $A \vee \neg A$?
- **Inference**:
  - Gentzen Sequent Calculus

- **Direct Proof**:

(I)	$A \vdash A$
($\neg$R)	$\vdash \neg A, A$
($\vee$R2)	$\vdash A \vee \neg A, A$
(PR)	$\vdash A, A \vee \neg A$
($\vee$R1)	$\vdash A \vee \neg A, A \vee \neg A$
(CR)	$\vdash A \vee \neg A$

- **Problem**:
  - the Sequent Calculus is deduction-complete - it can derive **every** tautology
  - Calculi for ATP used nowadays are only refutation-complete
    (they can only derive a contradiction for a given theorem)

Example of First-order Logic Resolution Proof

- **Given**:
  - **Axioms**:
    $\forall x\ Man(x) \Rightarrow Mortal(x)$

    Man(Socrates)
  - **Conjecture**:
    $\exists y\ Mortal(y)$?
- **Inference**:
  Resolution calculus

CNF:
$\neg Man(x) \lor Mortal(x)$

Man(Socrates)
$\neg Mortal(y)$ *(Neg. conj.)*

Proof:
1. $\neg Mortal(y)$ *(Neg. conj.)*
2. $\neg Man(x) \lor Mortal(x)$ *(Given)*
3. Man(Socrates) *(Given)*
4. Mortal(Socrates) *(Res. 2,3)*
5. $\bot$ *(Res. 1,4)*

Contradiction $\Rightarrow$ Conj. is true

Example of Description Logic Tableaux Proof

- **Given**:
  - **Axioms**:
    None
  - **Conjecture**:
    $\exists$ Child.$\neg$Male $\Rightarrow$

    $\neg\forall$ Child.Male ?
- **Inference**:
  Tableaux

Proof:
Check unsatisfiability of
$\exists$Child.$\neg$Male $\sqcap$ $\forall$ Child.Male

x: $\exists$Child.$\neg$Male $\sqcap$ $\forall$ Child.Male

x: $\forall$ Child.Male *($\sqcap$-rule)*

x: $\exists$Child.$\neg$Male *($\sqcap$-rule)*

x: Child y *($\exists$-rule)*

y: $\neg$Male *($\exists$-rule)*

y: Male *($\forall$-rule)*

<CLASH>

Contradiction $\Rightarrow$ Conj. is true

Calculi and Properties

- **For each calculus one has to specify**:
  - Syntax and semantics of its logic
  - Foundational axioms (if any)
  - Inference rules
  - How to combine inference rules applications into derivations

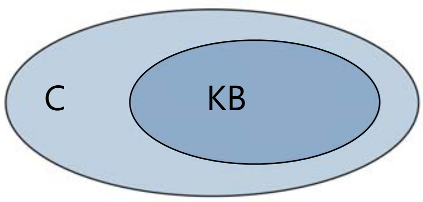
- **Derivability and Entailment**
  - Let KB be the conjunction of axioms
  - Let F be a formula (possibly a conjecture)
  - We say KB $\vdash$ F (read: KB derives F)
    if F can be derived from KB through rules of inference
  - We say KB $\models$ F (read: KB entails F, or KB models F)
    if (model-theoretic) semantics hold that F is true whenever KB is true

Model-Theoretic Semantics

- **Model-theoretic semantics for (propositional) logics**
  - An interpretation is a truth assignment to atomic elements of a KB:
    $$I\ \langle C,D \rangle \in \{ \langle F,F \rangle, \langle F,T \rangle, \langle T,F \rangle, \langle T,T \rangle \}$$
  - A model of a formula is an interpretation where it is true:
    $I\ \langle C,D \rangle = \langle F,T \rangle$ models $C \vee D$, $C \Rightarrow D$, but not $C \wedge D$
  - Two important properties of a formula C w.r.t. axioms of KB:
    - Entailment, written as KB $\models$ C: C is true in all models of KB
    - Consistency: C is true in ≥ 1 model of KB

- **Think of truth in a set-theoretic manner**

KB $\models$ C



Models of KB
$\subseteq$ Models of C

Calculi and Properties

- Important properties of calculi:
  - Soundness: If KB $\vdash$ C then KB $\vDash$ C
  - Completeness: If KB $\vDash$ C then KB $\vdash$ C
  - Refutational completeness: KB $\cup$ { $\neg$C } $\vdash$ $\bot$)
  - Termination: halts on any KB and C after finite time
- These properties may be incompatible, depending on the logic
  - **Decidable logics**: all three
    - Example: propositional logic
  - **Semi-decidable logics**: can have sound and (refutationally) complete calculus
    - Thus terminates if KB $\vDash$ C . Example: first-order logic
  - **Non-r.e. logics**: can't have sound and (refutationally) complete calculus
    - Example: second-order logic

Propositional Logic Syntax

- Propositional variables: p, rain, sunny
- Connectives: $\Rightarrow$ $\Leftrightarrow$ $\neg$ $\wedge$ $\vee$
- Inductive definition of well-formed formula (wff):
  - Base: All propositional vars are wffs
  - Inductive 1: If A is a wff then $\neg$A is a wff
  - Inductive 2: If A and B are wffs then
    A $\wedge$ B, A $\vee$ B, A $\Rightarrow$ B, A $\Leftrightarrow$ B are wffs
- Examples:
  - rain, rain $\Rightarrow$ $\neg$ sunny
  - (rain $\Rightarrow$ $\neg$ sunny) $\Leftrightarrow$ (sunny $\Rightarrow$ $\neg$rain)

Prop. Logic Semantics

- For a formula F, the truth I(F) under interpretation I is recursively defined:
  - Base:
    - F is prop var A then I(F)=true iff I(A)=true
  - Recursive:
    - F is $\neg$C then I(F)=true iff I(C)=false
    - F is C $\wedge$ D then I(F)=true iff I(C)=true and I(D)=true
    - F is C $\vee$ D then I(F)=true iff I(C)=true or I(D)=true
    - F is C $\Rightarrow$ D then I(F)=true iff I($\neg$C $\vee$ D)=true
    - F is C $\Leftrightarrow$ D then I(F)=true iff I(C $\Rightarrow$ D)=true and I(D $\Rightarrow$ C)=true

- Truth defined recursively from ground up
  - Modal logics don't have this property!

CNF Normalization

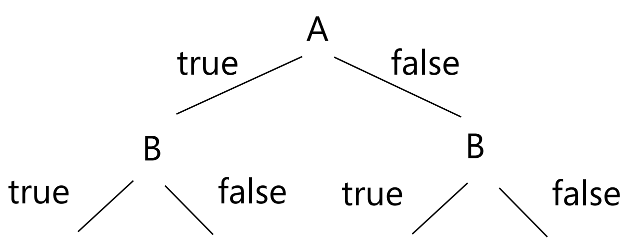
- Many theorem proving techniques req. KB to be in clausal normal form (CNF):
  - Rewrite all C $\Leftrightarrow$ D as C $\Rightarrow$ D $\wedge$ D $\Rightarrow$ C
  - Rewrite all C $\Rightarrow$ D as $\neg$C $\vee$ D
  - Push negation through connectives:
    - Rewrite $\neg$(C $\wedge$ D) as $\neg$C $\vee$ $\neg$D
    - Rewrite $\neg$(C $\vee$ D) as $\neg$C $\wedge$ $\neg$D
  - Rewrite double negation $\neg\neg$ C as C
  - Now NNF, to get CNF, distribute $\vee$ over $\wedge$:
    - Rewrite (C $\wedge$ D) $\vee$ E as (C $\vee$ E) $\wedge$ (D $\vee$ E)
- A clause is a disjunction of literals (pos/neg propositional variables)
- Can express KB, a set of clauses, as the conjunction of its clauses

CNF Normalization Example

- Given KB with single formula:
  - ¬ (rain ⇒ wet) ⇒ (inside ∧ warm)
- Rewrite all C ⇒ D as ¬C ∨ D
  - ¬ ¬ (¬ rain ∨ wet) ∨ (inside ∧ warm)
- Push negation through connectives:
  - (¬ ¬ ¬ rain ∨ ¬ ¬ wet) ∨ (inside ∧ warm)
- Rewrite double negation ¬ ¬ C as C
  - (¬ rain ∨ wet) ∨ (inside ∧ warm)
- Distribute ∨ over ∧:
  - (¬rain ∨ wet ∨ inside) ∧ (¬rain ∨ wet ∨ warm)
- CNF KB: {¬rain ∨ wet ∨ inside, ¬rain ∨ wet ∨ warm}

Prop. Theorem Proving

- A ⇒ B iff A ∧ ¬B is unsatisfiable
- Propositional logic is decidable, but NP-complete (reduction to 3-SAT)
- State-of-the-art prop. unsatisfiability methods are DPLL-based
- Many optimizations, more in lecture on SAT solving by Anbulagan



Instantiate prop vars until all clauses falsified, backtrack and do for all instantiations ⇒ unsat!

Prop. Tableaux Methods

- Given negated query F (in NNF), use rules to recursively break down:
  - α-Rule: Given $A \land B$ add A and B
  - β-Rule: Given $A \lor B$ branch on A and B
  - Clash: If A and $\neg A$ occur on same branch
- Clash on all branches indicates unsat!

$A \land \neg A \lor \neg B \land B$

$A \land \neg A$	(β-Rule)	$B \land B$	(β-Rule)
A	(α-Rule)	$\neg B$	(α-Rule)
$\neg A$	(α-Rule)	B	(α-Rule)
(Clash)		(Clash)	

Propositional Resolution

- One (!) inference rule

Resolution:

$$\frac{A \lor B \quad \neg B \lor C}{A \lor C}$$

Example application:

$$\frac{\neg precip \lor \neg freezing \lor snow \qquad \neg snow \lor slippery}{\neg precip \lor \neg freezing \lor slippery}$$

- The resolution calculus is sound and (refutationally) complete:

  $KB \models C$ if and only if $KB \cup \{\neg C\} \vdash \bot$

  – Simple strategy for completeness is to close clause set under Resolution
- NB: "One inference rule" calculus treats clauses as sets, otherwise need factoring:

Factoring: $$\frac{A \lor A \lor B}{A \lor B}$$

Soundness and completeness proof: see blackboard

Resolution Strategies

- Need strategies to restrict search:
  - **Unit resolution**
    - Only resolve with unit clauses
    - Complete for Horn KB (gives a "bottom-up flavour")
    - Intuition: Decrease clause size
  - **Set of support** (see also next two slides)
    - SOS starts with query clauses
    - Only resolve SOS clauses with non-SOS clauses and put resolvents in SOS
    - Intuition: KB should be satisfiable so refutation should derive from query
  - **Linear resolution**
    - Only resolve query clause with KB clauses, resolvent is new query clause
    - Complete for Horn KB (gives a "top-down flavour"), basis for Prolog
    - Together with ancestor resolution ⇒ complete for non-Horn, too
  - **Ordered resolution** resolve on maximal literals in clause only

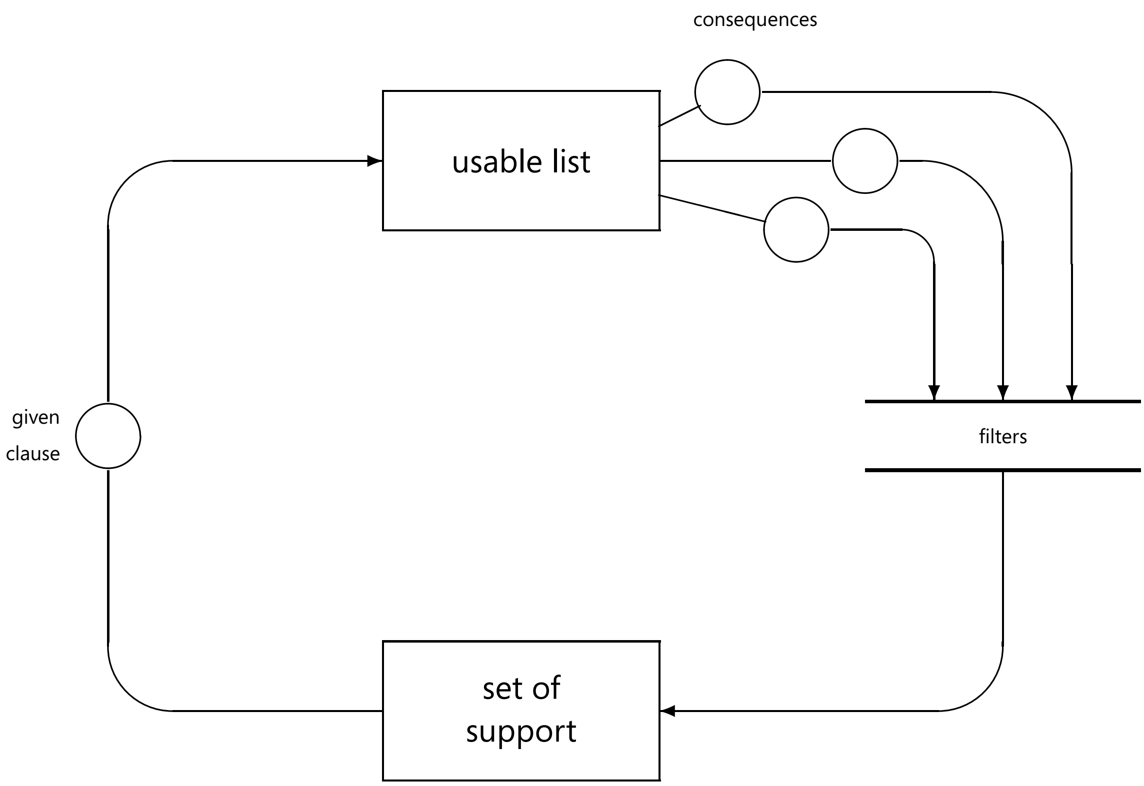
The "Given Clause Loop"

- In Otter theorem prover `http://en.wikipedia.org/wiki/Otter_(theorem_prover)`
- Lists of clauses maintained by the algorithm: USABLE and SOS.
- Initialize SOS with the input clauses, USABLE empty.
- Algorithm

```
While (SOS is not empty and no refutation has been found)
  1. Let given_clause be the `lightest' clause in SOS;
  2. Move given_clause from SOS to usable;
  3. Infer and process new clauses using the inference rules in
     effect; each new clause must have the given_clause as
     one of its parents and members of usable as its other
     parents;  new clauses that pass the retention tests
     are appended to SOS;
End of while loop.
```

- Fairness here: define clause weight e.g. as ``depth + length'' of clause
  - Important property: no clause is delayed infinitely long in (1)

The Given Clause Loop - Graphically



First-order logic

- Refer to objects and relations between them
- Propositional logic requires all relations to be propositionalized
  - Peter-at-home, Peter-at-work,
    Jim-at-subway, etc…
- Really want a compact relational form:
  - at(Peter, home), at(Peter, work),
    at(Jim, subway), etc…
- Then can use variables and quantify over all objects:
  - $\forall x\ (\mathrm{person}(x) \Rightarrow \exists y\ \mathrm{at}(x,y) \wedge \mathrm{place}(y))$

From Propositional Logic to First-Order Logic

- Generalize Syntax
- Generalize Semantics
  - Work with **Herbrand interpretations**
- Clause normal form generation
  - Involves **Skolemization** now
- Calculi for first-order clause logic
  - Involve **substitutions** and **unification** now

First-order Logic Syntax

- **Terms** (technical definition is inductive because of function symbols)
  - Variables: w, x, y, z
  - Constants: a, b, c, d
  - Functions over terms: f(a), f(x,y), f(x,c,f(f(z)))
- **Atoms**: P(x), Q(f(x,y)), R(x, f(x,f(c,z),c))
- **Connectives**: $\Rightarrow$ $\Leftrightarrow$ $\neg$ $\wedge$ $\vee$
- **Quantifiers**: $\forall$ $\exists$
- Inductive definition of wff:
  - Same as propositional logic but with following modifications
  - Base: All atoms over terms are wffs
  - Inductive: If A is a wff and x is a variable term
    then $\forall$x A and $\exists$x A are wffs

First-order Logic Semantics

- Interpretation I = (ΔI,•I)
  - ΔI is a non-empty domain
  - •I maps each function symbol f of arity n to a total function $(\Delta I)^n \mapsto \Delta I$
  - •I maps each predicate symbol P of arity n into a set of n-tuples over ΔI
- **Herbrand** interpretations (Th: KB is satisfiable iff it has a Herbrand model)
  - ΔI is set of ground terms { Peter, Jim, loc(Peter), loc(Jim), loc(loc(Peter)), ... }
  - •I maps each f to the identity function. Thus, I(loc(Peter)) = loc(Peter)
  - •I maps each predicate symbol P of arity n into a set of n-tuples over ΔI
  - Logical connectives interpreted as in propositional logic; new:
    $I \models \forall x\, A$ iff $I \models A[x/t]$, for all ground terms $t \in \Delta I$
- Example
  - •I may map at(•,•) into { ⟨Peter, loc(Peter)⟩ , ⟨Jim, loc(Jim)⟩ }
  - All other ground predicates are false in I, e.g. at(Jim, Jim)

Skolemization

- Skolemization is the process of getting rid of all ∃ quantifiers from a formula while preserving (un)satisfiability:
  - If ∃x quantifier is the outermost quantifier, remove the ∃ quantifier and substitute a new constant for x
  - If ∃x quantifier occurs inside of ∀ quantifiers, remove the ∃ quantifier and substitute a new function of all ∀ quantified variables for x
- Examples:
  - Skolemize($\exists w\, \exists x\, \forall y\, \forall z\, P(w,x,y,z)$) =

    $$\forall y\, \forall z\, P(c,d,y,z)$$

  - Skolemize($\forall w\, \exists x\, \forall y\, \exists z\, P(w,x,y,z)$) =

    $$\forall w\, \forall y\, P(w,f(w),y,g(w,y))$$

CNF Conversion

- CNF conversion is the same as the propositional case up to NNF, then do:
  - Standardize apart variables (all quantified variables get different names)
    - e.g. $(\forall x\ A(x)) \land (\exists x\ \neg A(x))$ becomes $(\forall x\ A(x)) \land (\exists y\ \neg A(y))$
  - Shift all quantifiers in front of formula (obtain, ultimately, prenex normal form)
    - $(\forall x\ A(x)) \land (\exists y\ \neg A(y))$ becomes $\exists y\ \forall x\ (A(x) \land \neg A(y))$
  - Skolemize formula
    - e.g. $\exists y\ \forall x\ (A(x) \land \neg A(y))$ becomes $\forall x\ (A(x) \land \neg A(c))$
  - Drop universals
    - e.g. $\forall x\ (A(x) \land \neg A(c))$ becomes $A(x) \land \neg A(c)$
  - Distribute $\lor$ over $\land$
  - Write result as a clause set (trivial)
    - e.g. $A(x) \land \neg A(c)$ becomes $\{ A(x), \neg A(c) \}$

Herbrand's Theorem

- Refutational theorem proving calculi are based on the following chain of reasoning
  - Any FO-formula is unsatisfiable iff its clause form is unsatisfiable
    (Non-trivial part is Skolemization)
  - A clause set is unsatisfiable iff it has no satisfying Herbrand interpretation
    (i.e. no Herbrand model)
  - A clause set has no Herbrand model iff
    some finite set of ground instances of its clauses is unsatisfiable
    (Herbrand's theorem)
- A naive application of the chain leads to **Gilmore's method**
  - It searches for this unsatisfiable set of ground instances in a direct way

Gilmore's Method

Preprocessing:

Given Formula

$\forall x\, \exists y\, P(y,x)$
$\land\, \forall z\, \neg P(z,a)$

Clause Form

$P(f(x),x)$
$\neg P(z,a)$

Outer loop:
Grounding

Inner loop:
Propositional
Method

Gilmore's Method

Preprocessing:

Given Formula

$\forall x\, \exists y\, P(y,x)$
$\land\, \forall z\, \neg P(z,a)$

Clause Form

$P(f(x),x)$
$\neg P(z,a)$

Outer loop:
Grounding

$P(f(a),a)$
$\neg P(a,a)$

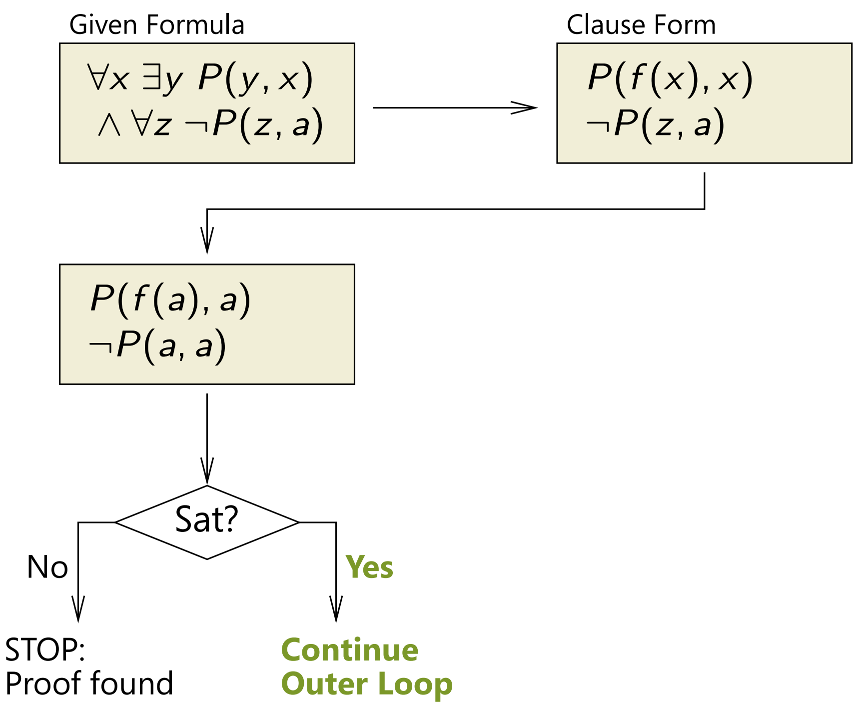
Inner loop:
Propositional
Method

Gilmore's Method

Preprocessing:

Outer loop:
Grounding

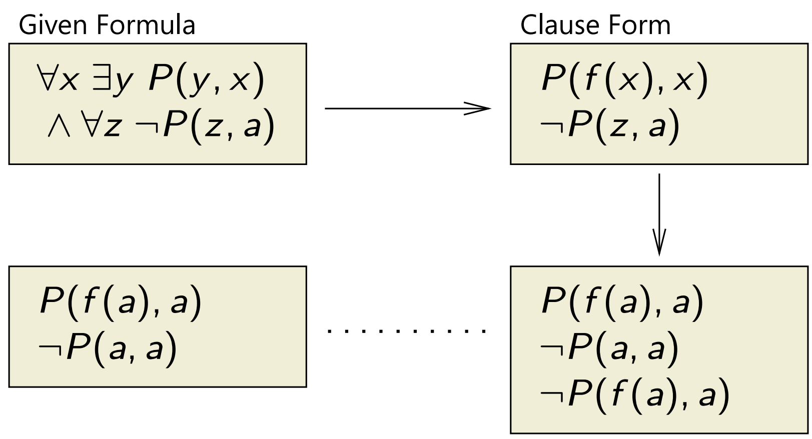
Inner loop:
Propositional
Method



Gilmore's Method

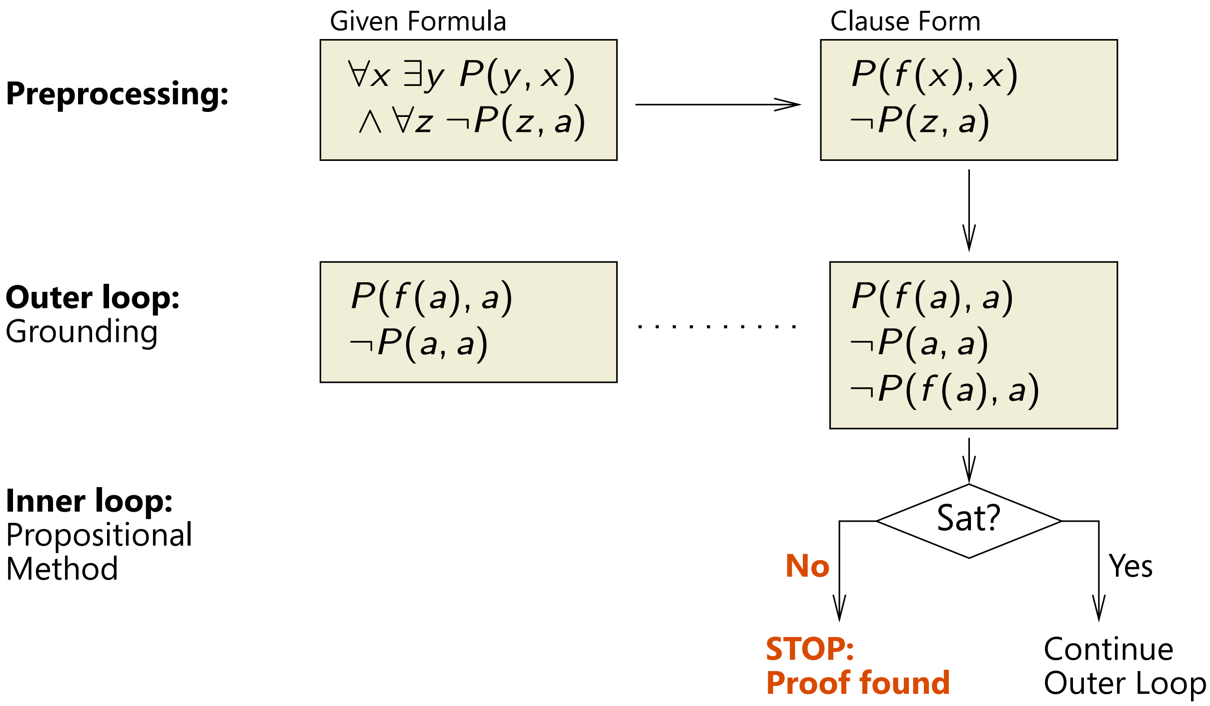
Preprocessing:

Outer loop:
Grounding



Inner loop:
Propositional
Method

Gilmore's Method



Problems with Gilmore's Method

- Gilmore's method reduces proof search in first-order logic to propositional logic unsatisfiability problems
- Main problem is the unguided generation of (very many) ground clauses
- All modern calculi address this problem in one way or another. e.g.
  - **Guidance**
    Instance-Based Methods are similar to Gilmore's method but generate ground instances in a guided way
  - **Avoidance**
    Resolution calculi need not generate the ground instances at all,
    they work directly on first-order clauses, not on their ground instances.
    This way, infinitely many ground resolution steps can be represented compactly with one first-order resolution step (sometimes)
- They use the **unification** operation to enable this

Better Methods for First-order Theorem Proving

- **Tableaux methods**
  - Highly successful for description and modal logics, which conform to certain (syntactically restricted) fragments of FOL
  - Not treated here
- **Resolution Methods**
  - Most successful technique for a variety of KBs
  - But... search space grows very quickly
  - Need a variety of optimizations in practice
    - strategies, ordering, redundancy elimination
- **Instance Based Methods**
  - Reduce proof search in FOL to proof search in propositional logic
  - Comparably new and interesting paradigm
- All methods are based on Herbrand interpretations
  - which justifies the use of **unification**

Substitution and Unification

- **Substitution**
  - A substitution list θ is a list of variable-term pairs
    - e.g., θ={x/3, y/f(z)}
  - When θ is applied to an FOL formula, every free occurrence of a variable in the list is replaced with the given term
    - e.g. $(P(x,y) \land \exists x\ P(x,y))\theta = P(3,f(z)) \land \exists x\ P(x,f(z))$
- **Unification / Most General Unifier**
  - The unifier UNIF(x,y) of two atoms/terms is a substitution that makes both arguments identical
    - e.g. UNIF(P(x,f(x)), P(y, f(f(z)))) = {x/f(1), y/f(1), z/1}
  - The most general unifier MGU(x,y) is just that... all other unifiers can be obtained from the MGU by additional substitution (MGU exists for unifiable args)
    - e.g. MGU(P(x,f(x)), P(y, f(f(z)))) = {x/f(z), y/f(z)}

An Instance-Based Method ("InstGen")

Current clauses

$$\mathsf{P}(\mathsf{f}(x), x) \lor \mathsf{Q}(x) \qquad \neg\mathsf{P}(z, \mathsf{a}) \lor \neg\mathsf{Q}(z)$$

ground

x,z → $

$$\underline{\mathsf{P}(\mathsf{f}(\$), \$)} \lor \mathsf{Q}(\$) \qquad \underline{\neg\mathsf{P}(\$, \mathsf{a})} \lor \neg\mathsf{Q}(\$)$$

Model: $\{\mathsf{P}(\mathsf{f}(\$), \$), \neg\mathsf{P}(\$, \mathsf{a})\}$

Model determines literals selection in current clauses for InstGen inference:

$$\text{InstGen}\ \frac{\underline{\mathsf{P}(\mathsf{f}(x), x)} \lor \mathsf{Q}(x) \qquad \underline{\neg\mathsf{P}(z, \mathsf{a})} \lor \neg\mathsf{Q}(z)}{\mathsf{P}(\mathsf{f}(\mathsf{a}), \mathsf{a}) \lor \mathsf{Q}(\mathsf{a}) \qquad \neg\mathsf{P}(\mathsf{f}(\mathsf{a}), \mathsf{a}) \lor \neg\mathsf{Q}(\mathsf{f}(\mathsf{a}))}$$

Conclusions are obtained by unifying selected literals
Add conclusions to "current clauses" and start over

Lifting Propositional Resolution to First-Order Resolution

- **Propositional Resolution**

Clauses	Ground instances
$P(f(x), y)$	$\{P(f(a), a), \ldots, P(f(f(a)), f(f(a))), \ldots\}$
$\neg P(z, z)$	$\{\neg P(a), \ldots, \neg P(f(f(a)), f(f(a))), \ldots\}$

Only common instances of $P(f(x), y)$ and $P(z, z)$ give rise to inference:

$$\frac{P(f(f(a)), f(f(a))) \qquad \neg P(f(f(a)), f(f(a)))}{\bot}$$

- **Observation (leading to "lifting lemma of resolution inferences")**

All common instances of $P(f(x), y)$ and $P(z, z)$ are instances of $P(f(x), f(x))$
$P(f(x), f(x))$ is computed deterministically by *unification*

- **First-Order Resolution**

$$\frac{P(f(x), y) \qquad \neg P(z, z)}{\bot}$$

Justified by existence of $P(f(x), f(x))$ via unification;
observation above tells us that these are the only inferences neccessary

Resolution for First-Order Clauses

- **Inference rules**

$$\frac{C \vee A \qquad D \vee B}{(C \vee D)\sigma} \quad \text{if } \sigma = \mathrm{MGU}(A, B) \qquad \text{[resolution]}$$

$$\frac{C \vee A \vee B}{(C \vee A)\sigma} \quad \text{if } \sigma = \mathrm{MGU}(A, B) \quad \text{[factorization]}$$

In both cases, A and B have to be renamed apart (made variable disjoint).

- **Example**

$$\frac{Q(z) \vee P(z, z) \quad \neg P(x, y)}{Q(x)} \quad \text{where } \sigma = [z/x, y/x] \qquad \text{[resolution]}$$

$$\frac{Q(z) \vee P(z, a) \vee P(a, y)}{Q(a) \vee P(a, a)} \quad \text{where } \sigma = [z/a, y/a] \quad \text{[factorization]}$$

Example of First-Order Resolution Proof

Given:
Axioms:
∀x Man(x) ⇒ Mortal(x)

Man(Socrates)

Conjecture:
∃y Mortal(y) ?

Inference:
Refutation
Resolution

CNF:
¬Man(x) ∨ Mortal(x)

Man(Socrates)
¬Mortal(y) [Neg. conj.]

Proof:
1. ¬Mortal(y) [Neg. conj.]
2. ¬Man(x) ∨ Mortal(x) [Given]
3. Man(Socrates) [Given]
4. Mortal(Socrates) [Res. 2,3]
5. ⊥ [Res. 1,4]

Contradiction ⇒ Conj. is true

Importance of Factoring

- Without the factoring rule, resolution is incomplete
- For example, take the following refutable clause set:
  - { A(w) v A(z), ~A(y) v ~A(z) }
- All binary resolutions yield clauses of the same form
- Clause set is only refutable if one of the clauses is first factored

Search Control

- Goal-directed / bottom-up search, as in propositional logic
  - SLD Resolution
    - KB of definite clauses (i.e. Horn rules), e.g.
      Uncle(x,y) :- Father(x,z) ∧ Brother(z,y)
    - Resolution backward chains from goal of rules
    - With negation-as-failure semantics, SLD- resolution is logic programming, i.e. Prolog
  - Negative and Positive Hyperresolution
    - All negative (positive) literals in nucleus clause are simultaneously resolved with completely positive (negative) satellite clauses
    - Positive Hyperresolution yields backward chaining
    - Negative Hyperresolution yields forward chaining
- Such search strategies prevent the generation of resolvents,
  they don't explain when clauses can be deleted (redundancy control)

Redundancy Control

- Redundancy of clauses is a huge problem in FOL resolution
  - For clauses C & D, C is redundant if $\exists\theta$ s.t. $C\theta \subseteq D$ as a multiset, a.k.a. θ-subsumption
  - If true, D is redundant and can be removed
    - Intuition: If D used in a refutation, $C\theta$ could be substituted leading to even shorter refutation

- Two types of subsumption where N is a new resolvent and A is a current clause:
  - Forward subsumption: A θ-subsumes N, delete N
  - Backward subsumption: N θ-subsumes A, delete A

- Forward / backward subsumption expensive but saves many redundant inferences
- Leads to saturation-based theorem proving (with orderings, in general)

Saturation Theorem Proving

- Given a set of clauses S:
  - S is saturated if all possible inferences from clauses in S generate forward subsumed clauses
  - All new inferences are "redundant" then and need not be carried out, without sacrificing completeness
  - If S does not contain the empty clause then S is satisfiable

- Saturation without deriving the empty clause implies no proof possible!
  And the clause set is satisfiable then.

- Usually need ordering restrictions to reach finite saturation.

Term Indexing

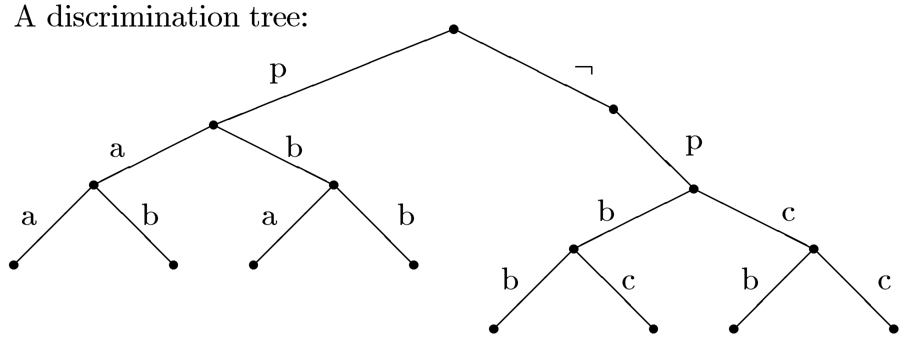
- Term indexing is an implementation technique for fast retrieval of sets of terms / clauses matching criteria

- Common uses in modern theorem provers:
  - Term q (query) is unifiable with term t (in index), i.e., ∃θ s.t. qθ = tθ
  - Term t is an instance of q, i.e., ∃θ s.t. qθ = t
  - Term t is a generalization of q, i.e., ∃θ s.t. q = tθ
  - Clause q subsumes clause t, i.e., ∃θ s.t. qθ ⊆ t
  - Clause q is subsumed by clause t, i.e., ∃θ s.t. tθ ⊆ q

- Techniques: (Google for "term indexing")
  - Path indexing
  - Substitution tree indexing, **discrimination trees**

Discrimination Tree Indexing

A discrimination tree:



Stores P(a,a), P(a,b), P(b,a), ... -P(c,c)

- Tree structure to look up terms or literals from a (large) database
  - Branches store terms as written down (from left to right)
- Doesn't distinguish different variables P(x,y) becomes P(?,?) (Can overretrieve)
- More efficient for common uses than linear search
- Can be combined with hashing of symbols, if branching is high

Equality

- A predicate w/ special interpretation
- Could axiomatize:
  - $x=x$ (reflexive)
  - $x=y \Rightarrow y=x$ (symmetric)
  - $x=y \wedge y=z \Rightarrow x=z$ (transitive)
  - For each function symbol f:
    - $x_1=y_1 \wedge \dots \wedge x_n=y_n \Rightarrow f(x_1,\dots,x_n)=f(y_1,\dots,y_n)$ (congruence)
  - For each predicate symbol P:
    - $x_1=y_1 \wedge \dots \wedge x_n=y_n \wedge P(x_1,\dots,x_n) \Rightarrow P(y_1,\dots,y_n)$ (congruence)
- Lead to bad search space
- Better to use dedicated inference rules (Paramodulation)

Inference Rules for Equality

- Demodulation (incomplete, based on matching)

Literal containing t

$$\frac{l=r \quad L[t] \vee D}{L[r\theta] \vee D} \quad l\theta = t$$

Example application:

$$\frac{f(x)=x \quad P(f(a)) \vee Q}{P(a) \vee Q} \quad \theta=\{x/a\}$$

- Paramodulation (complete, based on unification)

Literal containing t

$$\frac{l=r \vee C \quad L[t] \vee D}{(L[r] \vee C \vee D)\theta} \quad \theta=MGU(l,t)$$

Example application:

$$\frac{f(x,a)=x \vee C(x) \quad P(f(b,y)) \vee Q(y)}{P(b) \vee C(b) \vee Q(a)} \quad \theta=\{x/b,y/a\}$$

Equality Reasoning: Conclusions

- The inference rule of paramodulation together with the resolution and factoring inference rules constitute a sound a complete calculus for first-order logic with equality, i.e. can semi-decide the question whether

  $$E \models \phi$$

  holds, where E is the theory of equality (Ref, Sym, Trans, Congruence) and ϕ is an (arbitrary) formula.
- Caution: some search strategies no longer work (are incomplete), e.g. SOS
  - Unless "paramodulation into and below variables is permitted" (inpractical)
  - The practically most successful theorem provers are saturation-based, heavily use term orderings ("Replace bigger terms by smaller ones"), and the main inference rule is called "superposition"
- Natural question: can one "build-in" other/richer theories than E?
  - Answer: yes, the keyword is "Theory Reasoning"

Theory Reasoning

Let T be a first-order theory of signature Σ

Let L be a class of Σ-formulas

The T-validity Problem

Given ϕ in L, is it the case that $T \models \phi$?

More accurately:

Given ϕ in L, is it the case that $T \models \forall \phi$?

Examples

- "0/0, s/1, +/2, =/2, ≤/2" $\models \exists y.\ y > x$
- "The theory of equality E" $\models \phi$ (ϕ arbitrary formula, as above)
- "An equational theory" $\models \exists\ s_1=t_1 \wedge \ldots \wedge s_n=t_n$ (E-Unification problem)
- "Some group theory" $\models s = t$ (Word problem)

The T-validity problem is decidably only for restricted L and T

Theory Reasoning

The T-validity Problem

Is it the case that $T \models \phi$?

More accurately:

Is it the case that $T \models \forall \phi$?

I.e., Free vars are constants

The Dual Problem: T-satisfiability

Is it the case that ϕ is T-satisfiable?

More accurately:

Is it the case that $\exists \phi$ is T-satisfiable?

I.e., Free vars are constants

Prop: $T \models \Phi$ iff $\neg\Phi$ is T-unsatisfiable

Approaches to Theory Reasoning

- **Theory-Reasoning in Automated First-Order Theorem Proving:**
  - Semi-decide the T-validity problem, $T \models \phi$?
  - ϕ arbitrary first-order formula, T universal theory
  - Generality is strength and weakness at the same time
  - Really successful only for specific instance:
    - T = equality and equality inference rules like paramodulation

Approaches to Hybrid Reasoning

- **Satisfiability Modulo Theories (SMT)**
  - Decide the T-validity problem, $T \models \phi$?
  - Usual restrictions:
    - ϕ quantifier-free, i.e. all variables implicitly universally quantified
    - The T-satisfiability of conjunctions of literals must be decidable
  - SMT is the perhaps most advanced approach among those mentioned
  - Applications in particular to Formal verification

Checking Satisfiability Modulo Theories

Usual Formulation

Given:

A decision procedure for T-satisfiability of sets of literals

A quantifier-free formula ϕ (implicitly existentially quantified)

Task: Decide whether ϕ is T-satisfiable?

Approaches:

- **Eager translation into SAT**
  - Encode problem and theory into an equisatisfiable propositional formula
  - Feed formula to a SAT-solver
- **Lazy translation into SAT**
  - Couple a SAT solver with a decision procedure for T-satisfiability of ground literals
  - For instance if T is "equality" then the Nelson-Oppen congruence closure method can be used

Lazy Translation Into SAT

$$g(a) = c \quad \wedge \quad f(g(a)) \neq f(c) \ \vee \ g(a) = d \quad \wedge \quad c \neq d$$

Theory: Equality

Lazy Translation Into SAT

$$\underbrace{g(a) = c}_{1} \quad \wedge \quad \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \ \vee \ \underbrace{g(a) = d}_{3} \quad \wedge \quad \underbrace{c \neq d}_{\overline{4}}$$

- Send $\{1,\ \overline{2} \vee 3,\ \overline{4}\}$ to SAT solver.
- SAT solver returns model $\{1,\ \overline{2},\ \overline{4}\}$.
  Theory solver finds $\{1,\ \overline{2}\}$ E-unsatisfiable.
- Send $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2\}$ to SAT solver.
- SAT solver returns model $\{1,\ 2,\ 3,\ \overline{4}\}$.
  Theory solver finds $\{1,\ 3,\ \overline{4}\}$ E-unsatisfiable.
- Send $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4\}$ to SAT solver.
  SAT solver finds $\{1,\ \overline{2} \vee 3,\ \overline{4},\ \overline{1} \vee 2,\ \overline{1} \vee \overline{3} \vee 4\}$ unsatisfiable.

Lazy Translation Into SAT: Summary

- Mapping atoms to propositions is abstraction
- SAT solver computes a **solution**,
  i.e. **boolean assignment for atoms in literal set**
- Solution from SAT solver may not be true solution,
  i.e. the literal set is T-unsatisfiable
- Refine (strengthen) propositional formula by incorporating reason for false solution
- Reason provided by theory decision procedure, typically in form of subset of given literal set

More Optimizations

- **Theory Consequences**
  - The theory solver may return consequences (typically literals) to guide the SAT solver
- **Online SAT solving**
  - The SAT solver continues its search after accepting additional clauses (rather than restring from scratch)
- **Backjumping**
  - Instead of chronological backtracking
- **Preprocessing atoms**
  - Atoms are rewritten into normal form, using theory-specific atoms (e.g. associativity, commutativity)
- **Several layers of decision procedures**
  - „Cheaper" ones are applied first

Some SMT Systems

• Argo-lib,	University of Belgrade
• DPLL(T),	Technical University of Catalonia, U Iowa
• CVC Lite,	Stanford
• haRVey,	Loria
• ICS,	SRI
• Math-SAT,	ITC
• Tsat++,	University of Genova
• UCLID,	CMU

Combining Theories

Theories:

- $\mathcal{R}$: theory of rationals
  $\Sigma_{\mathcal{R}} = \{\leq, +, -, 0, 1\}$
- $\mathcal{L}$: theory of lists
  $\Sigma_{\mathcal{L}} = \{=, \mathrm{hd}, \mathrm{tl}, \mathrm{nil}, \mathrm{cons}\}$
- $\mathcal{E}$: theory of equality
  Σ: free function and predicate symbols

Problem: Is

$$x \leq y \wedge y \leq x + \mathrm{hd}(\mathrm{cons}(0, \mathrm{nil})) \wedge P(h(x) - h(y)) \wedge \neg P(0)$$

satisfiable in $\mathcal{R} \cup \mathcal{L} \cup \mathcal{E}$?

Nelson-Oppen Combination Method

G. Nelson and D.C. Oppen: *Simplification by cooperating decision procedures*, ACM Trans. on Programming Languages and Systems, 1(2):245-257, 1979.

Given:

- $\mathcal{T}_1$, $\mathcal{T}_2$ first-order theories with signatures Σ_1, Σ_2
- $\Sigma_1 \cap \Sigma_2 = \emptyset$
- ϕ quantifier-free formula over $\Sigma_1 \cup \Sigma_2$

Obtain a decision procedure for satisfiability in $\mathcal{T}_1 \cup \mathcal{T}_2$ from decision procedures for satisfiability in $\mathcal{T}_1$ and $\mathcal{T}_2$.

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \land y \leq x + \mathrm{hd}(\mathrm{cons}(0, \mathrm{nil})) \land P(h(x) - h(y)) \land \neg P(0)$$

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \wedge y \leq x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \wedge P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \wedge \neg P(\underbrace{0}_{v_5})$$

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \wedge y \leq x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \wedge P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \wedge \neg P(\underbrace{0}_{v_5})$$

$\mathcal{R}$	$\mathcal{L}$	$\mathcal{E}$
$x \leq y$		$P(v_2)$
$y \leq x + v_1$		$\neg P(v_5)$

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \wedge y \leq x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \wedge P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \wedge \neg P(\underbrace{0}_{v_5})$$

$\mathcal{R}$	$\mathcal{L}$	$\mathcal{E}$
$x \leq y$		$P(v_2)$
$y \leq x + v_1$		$\neg P(v_5)$
$v_2 = v_3 - v_4$	$v_1 = \mathrm{hd}(\mathrm{cons}(v_5, \mathrm{nil}))$	$v_3 = h(x)$
$v_5 = 0$		$v_4 = h(y)$

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \wedge y \leq x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \wedge P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \wedge \neg P(\underbrace{0}_{v_5})$$

$\mathcal{R}$	$\mathcal{L}$	$\mathcal{E}$
$x \leq y$		$P(v_2)$
$y \leq x + v_1$		$\neg P(v_5)$
$v_2 = v_3 - v_4$	$v_1 = \mathrm{hd}(\mathrm{cons}(v_5, \mathrm{nil}))$	$v_3 = h(x)$
$v_5 = 0$		$v_4 = h(y)$
	$v_1 = v_5$	

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \wedge y \leq x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \wedge P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \wedge \neg P(\underbrace{0}_{v_5})$$

$\mathcal{R}$	$\mathcal{L}$	$\mathcal{E}$
$x \leq y$		$P(v_2)$
$y \leq x + v_1$		$\neg P(v_5)$
$v_2 = v_3 - v_4$	$v_1 = \mathrm{hd}(\mathrm{cons}(v_5, \mathrm{nil}))$	$v_3 = h(x)$
$v_5 = 0$		$v_4 = h(y)$
$x = y$	$v_1 = v_5$	

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \wedge y \leq x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \wedge P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \wedge \neg P(\underbrace{0}_{v_5})$$

$\mathcal{R}$	$\mathcal{L}$	$\mathcal{E}$
$x \leq y$		$P(v_2)$
$y \leq x + v_1$		$\neg P(v_5)$
$v_2 = v_3 - v_4$	$v_1 = \mathrm{hd}(\mathrm{cons}(v_5, \mathrm{nil}))$	$v_3 = h(x)$
$v_5 = 0$		$v_4 = h(y)$
$x = y$	$v_1 = v_5$	$v_3 = v_4$

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \land y \leq x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \land P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \land \neg P(\underbrace{0}_{v_5})$$

$\mathcal{R}$	$\mathcal{L}$	$\mathcal{E}$
$x \leq y$		$P(v_2)$
$y \leq x + v_1$		$\neg P(v_5)$
$v_2 = v_3 - v_4$	$v_1 = \mathrm{hd}(\mathrm{cons}(v_5, \mathrm{nil}))$	$v_3 = h(x)$
$v_5 = 0$		$v_4 = h(y)$
$x = y$	$v_1 = v_5$	$v_3 = v_4$
$v_2 = v_5$		

Nelson-Oppen Combination Method

Variable abstraction + *equality propagation*:

$$x \leq y \land y \leq x + \underbrace{\mathrm{hd}(\mathrm{cons}(0, \mathrm{nil}))}_{v_1} \land P(\underbrace{\underbrace{h(x)}_{v_3} - \underbrace{h(y)}_{v_4}}_{v_2}) \land \neg P(\underbrace{0}_{v_5})$$

$\mathcal{R}$	$\mathcal{L}$	$\mathcal{E}$
$x \leq y$		$P(v_2)$
$y \leq x + v_1$		$\neg P(v_5)$
$v_2 = v_3 - v_4$	$v_1 = \mathrm{hd}(\mathrm{cons}(v_5, \mathrm{nil}))$	$v_3 = h(x)$
$v_5 = 0$		$v_4 = h(y)$
$x = y$	$v_1 = v_5$	$v_3 = v_4$
$v_2 = v_5$		$\perp$