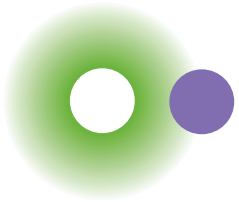

Automated Theorem Proving

Peter Baumgartner



NICTA

`Peter.Baumgartner@nicta.com.au`

`http://users.rsise.anu.edu.au/~baumgart/`

Slides partially based on material by Alexander Fuchs, Harald Ganzinger, John Slaney, Viorica Sofronie-Stockermans and Uwe Waldmann

Purpose of This Lecture

Overview of Automated Theorem Proving (ATP)

- Emphasis on automated proof methods for **first-order logic**
- More “breadth” than “depth”

Standard techniques covered

- Normal forms of formulas
- Herbrand interpretations
- Resolution calculus, unification
- Instance-based methods
- Model computation
- Theory reasoning: Satisfiability Modulo Theories

Part 1: What is Automated Theorem Proving?

First-Order Theorem Proving in Relation to ...

... Calculation: Compute function value at given point:

$$\text{Problem: } 2^2 = ? \quad 3^2 = ? \quad 4^2 = ?$$

“Easy” (often polynomial)

... Constraint Solving: Given:

● Problem: $x^2 = a$ where $x \in [1 \dots b]$

(x variable, a , b parameters)

● Instance: $a = 16$, $b = 10$

Find values for variables such that problem instance is satisfied

“Difficult” (often exponential, but restriction to **finite** domains)

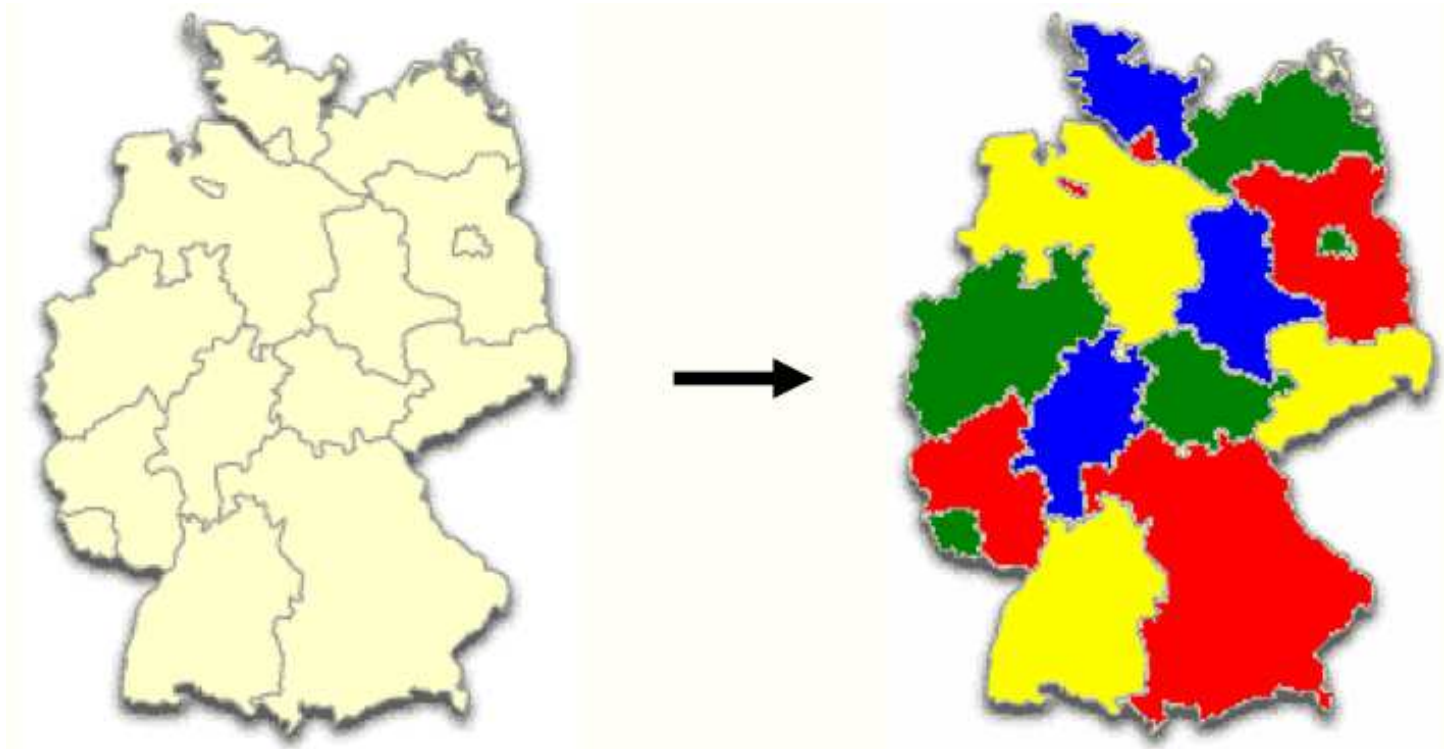
First-Order Theorem Proving: Given:

$$\text{Problem: } \exists x (x^2 = a \wedge x \in [1 \dots b])$$

Is it satisfiable? unsatisfiable? valid?

“Very difficult” (often undecidable)

Logical Analysis Example: Three Coloring Problem



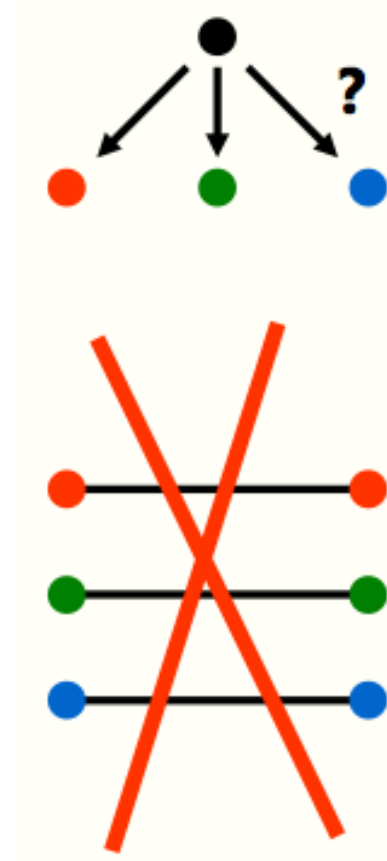
Problem: Given a map. Can it be colored using only three colors, where neighbouring countries are colored differently?

Three Coloring Problem - Graph Theory Abstraction

Problem Instance



Problem Specification



The Rôle of Theorem Proving?

Three Coloring Problem - Formalization

Every node has at least one color

$$\forall N (\text{red}(N) \vee \text{green}(N) \vee \text{blue}(N))$$

Every node has at most one color

$$\begin{aligned} \forall N ((\text{red}(N) \rightarrow \neg \text{green}(N)) \wedge \\ (\text{red}(N) \rightarrow \neg \text{blue}(N)) \wedge \\ (\text{blue}(N) \rightarrow \neg \text{green}(N))) \end{aligned}$$

Adjacent nodes have different color

$$\begin{aligned} \forall M, N (\text{edge}(M, N) \rightarrow (\neg(\text{red}(M) \wedge \text{red}(N)) \wedge \\ \neg(\text{green}(M) \wedge \text{green}(N)) \wedge \\ \neg(\text{blue}(M) \wedge \text{blue}(N)))) \end{aligned}$$

Three Coloring Problem - Solving Problem Instances ...

... with a constraint solver:

Let constraint solver find value(s) for variable(s) such that problem instance is satisfied

Here: Variables: Colors of nodes in graph
Values: Red, green or blue
Problem instance: Specific graph to be colored

... with a theorem prover

Let the theorem prover prove that the three coloring 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 theorem prover (e.g. use a SAT solver)
- Theorem provers are not even guaranteed to terminate, in general

Other tasks where theorem proving is more appropriate?

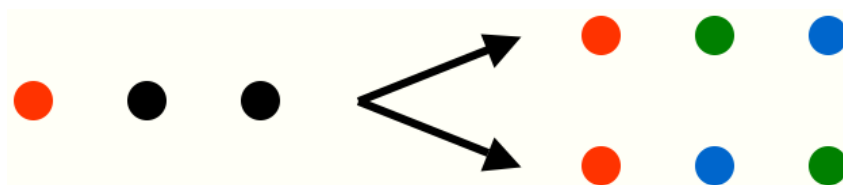
Three Coloring Problem: The Rôle of Theorem Proving

Functional dependency

- Blue coloring depends functionally on the red and green coloring



- Blue coloring does not functionally depend on the red coloring



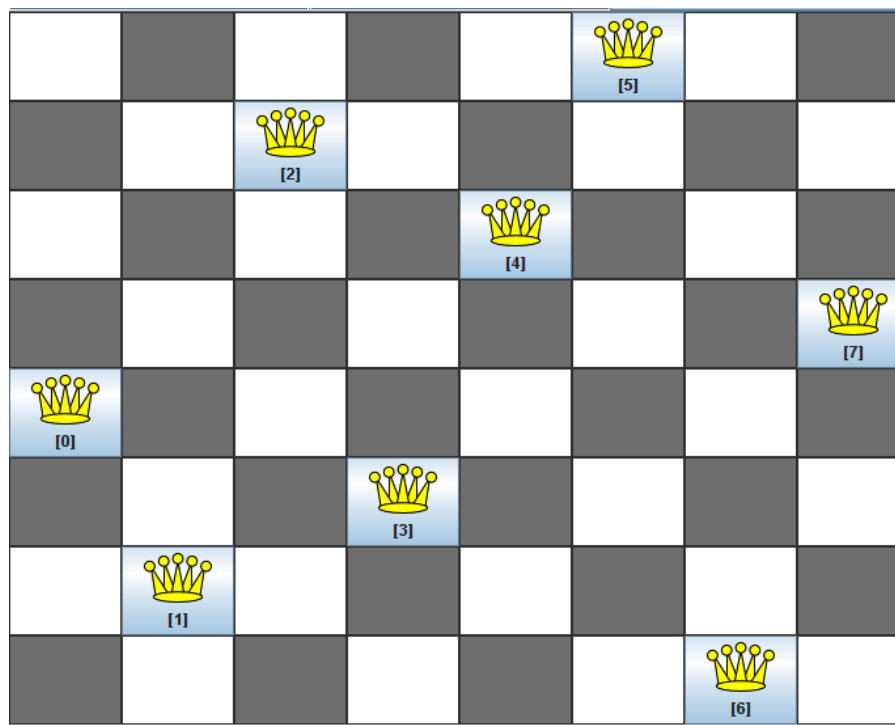
Theorem proving: Prove a formula is valid. Here:

Is “the blue coloring is functionally dependent on the red/green and green coloring” (as a formula) valid, i.e. holds for all possible graphs?

I.e. analysis wrt. **all** instances \Rightarrow theorem proving is adequate

Theorem Prover Demo

Another Example: Proving Symmetries



The n-queens problem has “variable symmetries”: mirroring preserves solutions.

Can add a constraint “queen on first row must be left of queen on last row”.

Theorem Prover Demo

Part 2: Methods in Automated Theorem Proving

How to Build a (First-Order) Theorem Prover

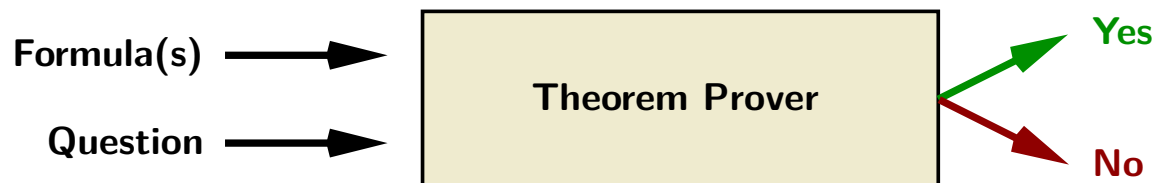
1. Fix an **input language** for formulas
2. Fix a **semantics** to define what the formulas mean
Will be always “classical” here
3. Determine the desired **services** from the theorem prover
(The questions we would like the prover be able to answer)
4. Design a **calculus** for the logic and the services
Calculus: high-level description of the “logical analysis” algorithm
This includes redundancy criteria for formulas and inferences
5. Prove the calculus is **correct** (sound and complete) wrt. the logic and the services, if possible
6. Design a **proof procedure** for the calculus
7. Implement the proof procedure (research topic of its own)

Go through the red issues in the rest of this talk

How to Build a (First-Order) Theorem Prover

1. Fix an **input language** for formulas
2. Fix a **semantics** to define what the formulas mean
Will be always “classical” here
3. Determine the desired **services** from the theorem prover
(The questions we would like the prover be able to answer)
4. Design a **calculus** for the logic and the services
Calculus: high-level description of the “logical analysis” algorithm
This includes redundancy criteria for formulas and inferences
5. Prove the calculus is **correct** (sound and complete) wrt. the logic and the services, if possible
6. Design a **proof procedure** for the calculus
7. Implement the proof procedure (research topic of its own)

Languages and Services — Propositional SAT



Formula: Propositional logic formula ϕ

Question: Is ϕ satisfiable?

(Minimal model? Maximal consistent subsets?)

Theorem Prover: Based on BDD, **DPLL**, or stochastic local search

Issue: the formula ϕ can be **BIG**

DPLL as a Semantic Tree Method

$$(1) A \vee B$$

$$(2) C \vee \neg A$$

$$(3) D \vee \neg C \vee \neg A$$

$$(4) \neg D \vee \neg B$$

\langle empty tree \rangle

$$\{\} \not\models A \vee B$$

$$\{\} \models C \vee \neg A$$

$$\{\} \models D \vee \neg C \vee \neg A$$

$$\{\} \models \neg D \vee \neg B$$

- A Branch stands for an interpretation
- **Purpose of splitting:** satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it (\star)

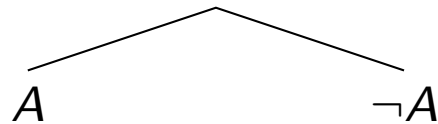
DPLL as a Semantic Tree Method

$$(1) A \vee B$$

$$(2) C \vee \neg A$$

$$(3) D \vee \neg C \vee \neg A$$

$$(4) \neg D \vee \neg B$$



$$\{A\} \models A \vee B$$

$$\{A\} \not\models C \vee \neg A$$

$$\{A\} \models D \vee \neg C \vee \neg A$$

$$\{A\} \models \neg D \vee \neg B$$

- A Branch stands for an interpretation
- **Purpose of splitting:** satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it (★)

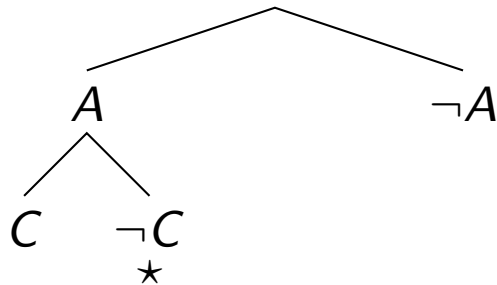
DPLL as a Semantic Tree Method

(1) $A \vee B$

(2) $C \vee \neg A$

(3) $D \vee \neg C \vee \neg A$

(4) $\neg D \vee \neg B$



$\{A, C\} \models A \vee B$

$\{A, C\} \models C \vee \neg A$

$\{A, C\} \not\models D \vee \neg C \vee \neg A$

$\{A, C\} \models \neg D \vee \neg B$

- A Branch stands for an interpretation
- **Purpose of splitting:** satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it (★)

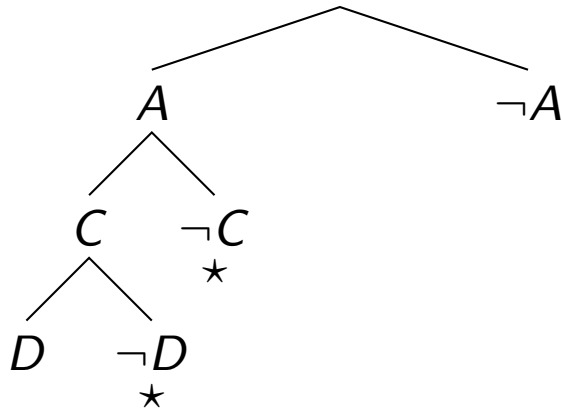
DPLL as a Semantic Tree Method

(1) $A \vee B$

(2) $C \vee \neg A$

(3) $D \vee \neg C \vee \neg A$

(4) $\neg D \vee \neg B$



$\{A, C, D\} \models A \vee B$

$\{A, C, D\} \models C \vee \neg A$

$\{A, C, D\} \models D \vee \neg C \vee \neg A$

$\{A, C, D\} \models \neg D \vee \neg B$

Model $\{A, C, D\}$ found.

- A Branch stands for an interpretation
- **Purpose of splitting:** satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it (*)

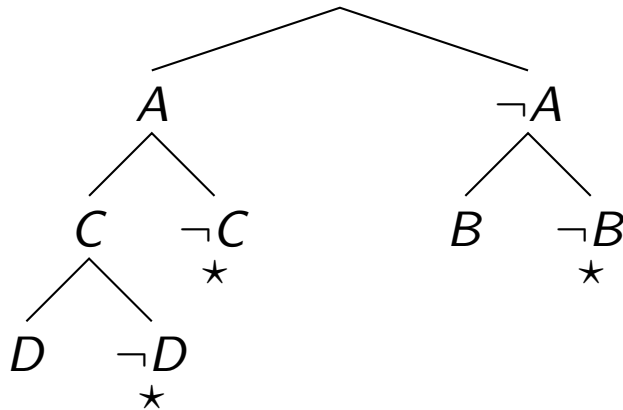
DPLL as a Semantic Tree Method

(1) $A \vee B$

(2) $C \vee \neg A$

(3) $D \vee \neg C \vee \neg A$

(4) $\neg D \vee \neg B$



$\{B\} \models A \vee B$

$\{B\} \models C \vee \neg A$

$\{B\} \models D \vee \neg C \vee \neg A$

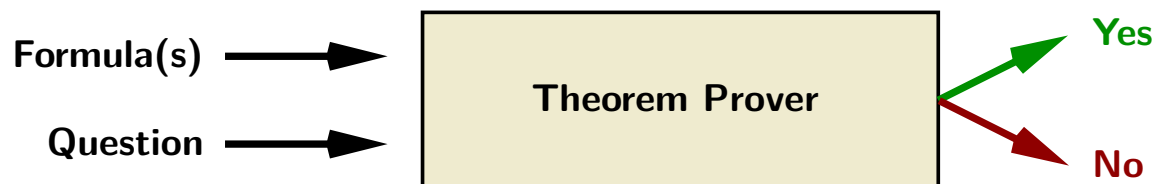
$\{B\} \models \neg D \vee \neg B$

Model $\{B\}$ found.

- A Branch stands for an interpretation
- **Purpose of splitting:** satisfy a clause that is currently falsified
- Close branch if some clause is plainly falsified by it (★)

DPLL is the basis of most efficient SAT solvers today

Languages and Services — Description Logics



Formula: Description Logic TBox + ABox (restricted FOL)

TBox: Terminology

ABox: Assertions

$\text{Professor} \sqcap \exists \text{supervises} . \text{Student} \sqsubseteq \text{BusyPerson}$ $p : \text{Professor} \quad (p, s) : \text{supervises}$

Question: Is TBox + ABox satisfiable?

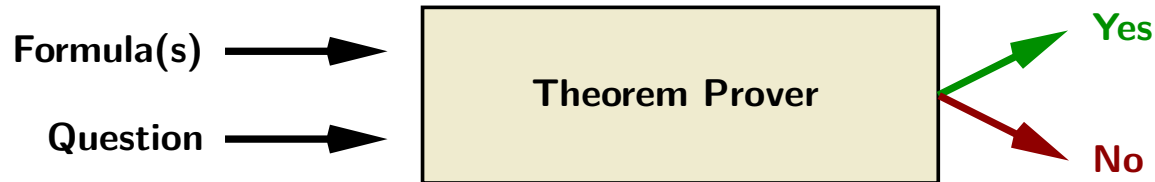
(Does C subsume D ?, Concept hierarchy?)

Theorem Prover: Tableaux algorithms (predominantly)

Issue: Push expressivity of DLs while preserving decidability

See overview lecture by Maurice Pagnucco on “Knowledge Representation and Reasoning”

Languages and Services — Satisfiability Modulo Theories (SM)



Formula: Usually **variable-free** first-order logic formula ϕ
Equality \doteq , combination of theories, free symbols

Question: Is ϕ valid? (satisfiable? entailed by another formula?)

$$\models_{\text{NUL}} \forall l (c = 5 \rightarrow \text{car}(\text{cons}(3 + c, l)) \doteq 8)$$

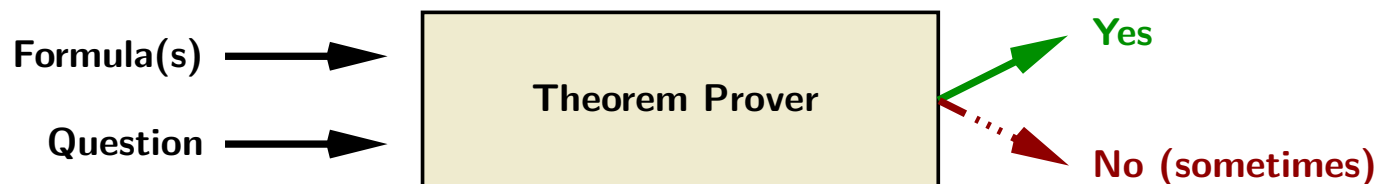
Theorem Prover: DPLL(T), translation into SAT, first-order provers

Issue: essentially undecidable for non-variable free fragment

$$P(0) \wedge (\forall x P(x) \rightarrow P(x + 1)) \models_{\mathbb{N}} \forall x P(x)$$

Design a “good” prover anyways (ongoing research)

Languages and Services — “Full” First-Order Logic



Formula: First-order logic formula ϕ (e.g. the three-coloring spec above)

Usually with equality \doteq

Question: Is ϕ formula valid? (satisfiable?, entailed by another formula?)

Theorem Prover: Superposition (Resolution), Instance-based methods

Issues

- Efficient treatment of equality
- Decision procedure for sub-languages or useful reductions?
Can do e.g. DL reasoning? Model checking? Logic programming?
- Built-in inference rules for arrays, lists, arithmetics (still open research)

How to Build a (First-Order) Theorem Prover

1. Fix an **input language** for formulas
2. Fix a **semantics** to define what the formulas mean
Will be always “classical” here
3. Determine the desired **services** from the theorem prover
(The questions we would like the prover be able to answer)
4. Design a **calculus** for the logic and the services
Calculus: high-level description of the “logical analysis” algorithm
This includes redundancy criteria for formulas and inferences
5. Prove the calculus is **correct** (sound and complete) wrt. the logic and the services, if possible
6. Design a **proof procedure** for the calculus
7. Implement the proof procedure (research topic of its own)

Semantics

“The function f is continuous”, expressed in (first-order) predicate logic:

$$\forall \varepsilon (0 < \varepsilon \rightarrow \forall a \exists \delta (0 < \delta \wedge \forall x (|x - a| < \delta \rightarrow |f(x) - f(a)| < \varepsilon)))$$

Underlying Language

Variables $\varepsilon, a, \delta, x$

Function symbols $0, | - |, - - -, f(-)$

Terms are well-formed expressions over variables and function symbols

Predicate symbols $- < -, - = -$

Atoms are applications of predicate symbols to terms

Boolean connectives $\wedge, \vee, \rightarrow, \neg$

Quantifiers \forall, \exists

The function symbols and predicate symbols comprise a signature Σ

Semantics

“The function f is continuous”, expressed in (first-order) predicate logic:

$$\forall \varepsilon (0 < \varepsilon \rightarrow \forall a \exists \delta (0 < \delta \wedge \forall x (|x - a| < \delta \rightarrow |f(x) - f(a)| < \varepsilon)))$$

“Meaning” of Language Elements – Σ -Algebras

Universe (aka Domain): Set U

Variables \mapsto values in U (mapping is called “assignment”)

Function symbols \mapsto (total) functions over U

Predicate symbols \mapsto relations over U

Boolean connectives \mapsto the usual boolean functions

Quantifiers \mapsto “for all ... holds”, “there is a ..., such that”

Terms \mapsto values in U

Formulas \mapsto Boolean (Truth-) values

Semantics - Σ -Algebra Example

Let Σ_{PA} be the standard signature of Peano Arithmetic

The standard interpretation \mathbb{N} for Peano Arithmetic then is:

$$U_{\mathbb{N}} = \{0, 1, 2, \dots\}$$

$$0_{\mathbb{N}} = 0$$

$$s_{\mathbb{N}} : n \mapsto n + 1$$

$$+_{\mathbb{N}} : (n, m) \mapsto n + m$$

$$*_{\mathbb{N}} : (n, m) \mapsto n * m$$

$$\leq_{\mathbb{N}} = \{(n, m) \mid n \text{ less than or equal to } m\}$$

$$<_{\mathbb{N}} = \{(n, m) \mid n \text{ less than } m\}$$

Note that \mathbb{N} is just one out of **many possible** Σ_{PA} -interpretations

Semantics - Σ -Algebra Example

Evaluation of terms and formulas

Under the interpretation \mathbb{N} and the assignment $\beta : x \mapsto 1, y \mapsto 3$ we obtain

$$(\mathbb{N}, \beta)(s(x) + s(0)) = 3$$

$$(\mathbb{N}, \beta)(x + y \doteq s(y)) = \textit{True}$$

$$(\mathbb{N}, \beta)(\forall z z \leq y) = \textit{False}$$

$$(\mathbb{N}, \beta)(\forall x \exists y x < y) = \textit{True}$$

$$\mathbb{N}(\forall x \exists y x < y) = \textit{True} \quad (\text{Short notation when } \beta \text{ irrelevant})$$

Important Basic Notion: Model

If ϕ is a closed formula, then, instead of $I(\phi) = \textit{True}$ one writes

$$I \models \phi \quad (\text{"}I \text{ is a model of } \phi\text{"})$$

E.g. $\mathbb{N} \models \forall x \exists y x < y$

Standard reasoning services can now be expressed semantically

Services Semantically

E.g. “entailment”:

Axioms over $\mathbb{R} \wedge \text{continuous}(f) \wedge \text{continuous}(g) \models \text{continuous}(f + g) ?$

Services

Model(I, ϕ): $I \models \phi ?$ (Is I a model for ϕ ?)

Validity(ϕ): $\models \phi ?$ ($I \models \phi$ for every interpretation?)

Satisfiability(ϕ): ϕ satisfiable? ($I \models \phi$ for some interpretation?)

Entailment(ϕ, ψ): $\phi \models \psi ?$ (does ϕ entail ψ ?, i.e.
for every interpretation I : if $I \models \phi$ then $I \models \psi$?)

Solve(I, ϕ): find an assignment β such that $I, \beta \models \phi$

Solve(ϕ): find an interpretation and assignment β such that $I, \beta \models \phi$

Additional complication: fix interpretation of some symbols (as in \mathbb{N} above)

What if theorem prover’s native service is only “Is ϕ unsatisfiable?” ?

Semantics - Reduction to Unsatisfiability

- Suppose we want to prove an entailment $\phi \models \psi$
- Equivalently, prove $\models \phi \rightarrow \psi$, i.e. that $\phi \rightarrow \psi$ is valid
- Equivalently, prove that $\neg(\phi \rightarrow \psi)$ is not satisfiable (unsatisfiable)
- Equivalently, prove that $\phi \wedge \neg\psi$ is unsatisfiable

Basis for (predominant) refutational theorem proving

Dual problem, much harder: to disprove an entailment $\phi \models \psi$ find a model of $\phi \wedge \neg\psi$

One motivation for (finite) model generation procedures

How to Build a (First-Order) Theorem Prover

1. Fix an **input language** for formulas
2. Fix a **semantics** to define what the formulas mean
Will be always “classical” here
3. Determine the desired **services** from the theorem prover
(The questions we would like the prover be able to answer)
4. Design a **calculus** for the logic and the services
Calculus: high-level description of the “logical analysis” algorithm
This includes redundancy criteria for formulas and inferences
5. Prove the calculus is **correct** (sound and complete) wrt. the logic and the services, if possible
6. Design a **proof procedure** for the calculus
7. Implement the proof procedure (research topic of its own)

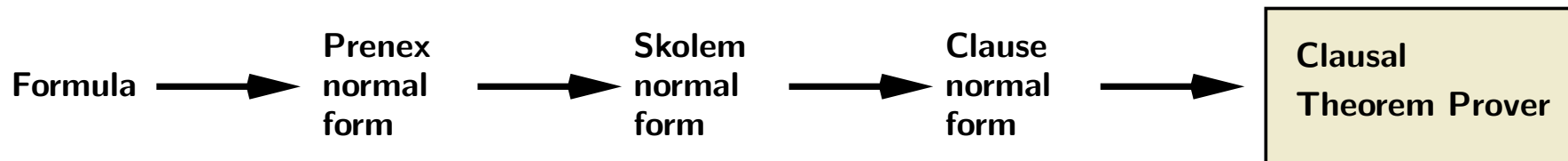
Calculus - Normal Forms

Most first-order theorem provers take formulas in **clause normal form**

Why Normal Forms?

- Reduction of logical concepts (operators, quantifiers)
- Reduction of syntactical structure (nesting of subformulas)
- Can be exploited for efficient data structures and control

Translation into Clause Normal Form



Prop: the given formula and its clause normal form are equi-satisfiable

Prenex Normal Form

Prenex formulas have the form

$$Q_1x_1 \dots Q_nx_n F,$$

where F is quantifier-free and $Q_i \in \{\forall, \exists\}$

Computing prenex normal form by the rewrite relation \Rightarrow_P :

$$(F \leftrightarrow G) \Rightarrow_P (F \rightarrow G) \wedge (G \rightarrow F)$$

$$\neg Qx F \Rightarrow_P \overline{Q}x \neg F \quad (\neg Q)$$

$$(Qx F \rho G) \Rightarrow_P Qy(F[y/x] \rho G), \quad y \text{ fresh}, \quad \rho \in \{\wedge, \vee\}$$

$$(Qx F \rightarrow G) \Rightarrow_P \overline{Q}y(F[y/x] \rightarrow G), \quad y \text{ fresh}$$

$$(F \rho Qx G) \Rightarrow_P Qy(F \rho G[y/x]), \quad y \text{ fresh}, \quad \rho \in \{\wedge, \vee, \rightarrow\}$$

Here \overline{Q} denotes the quantifier **dual** to Q , i.e., $\overline{\forall} = \exists$ and $\overline{\exists} = \forall$.

In the Example

$$\forall \varepsilon (0 < \varepsilon \rightarrow \forall a \exists \delta (0 < \delta \wedge \forall x (|x - a| < \delta \rightarrow |f(x) - f(a)| < \varepsilon)))$$

$\Rightarrow P$

$$\forall \varepsilon \forall a (0 < \varepsilon \rightarrow \exists \delta (0 < \delta \wedge \forall x (|x - a| < \delta \rightarrow |f(x) - f(a)| < \varepsilon)))$$

$\Rightarrow P$

$$\forall \varepsilon \forall a \exists \delta (0 < \varepsilon \rightarrow 0 < \delta \wedge \forall x (|x - a| < \delta \rightarrow |f(x) - f(a)| < \varepsilon))$$

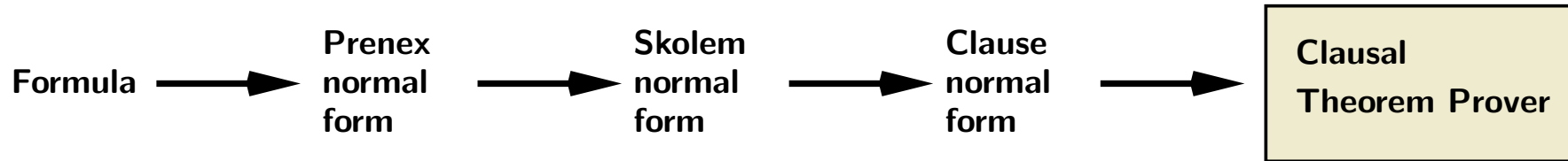
$\Rightarrow P$

$$\forall \varepsilon \forall a \exists \delta (0 < \varepsilon \rightarrow \forall x (0 < \delta \wedge |x - a| < \delta \rightarrow |f(x) - f(a)| < \varepsilon))$$

$\Rightarrow P$

$$\forall \varepsilon \forall a \exists \delta \forall x (0 < \varepsilon \rightarrow (0 < \delta \wedge (|x - a| < \delta \rightarrow |f(x) - f(a)| < \varepsilon)))$$

Skolem Normal Form



Intuition: replacement of $\exists y$ by a concrete choice function computing y from all the arguments y depends on.

Transformation \Rightarrow_S

$$\forall x_1, \dots, x_n \exists y F \Rightarrow_S \forall x_1, \dots, x_n F[f(x_1, \dots, x_n)/y]$$

where f/n is a new function symbol (**Skolem function**).

In the Example

$$\forall \epsilon \forall a \exists \delta \forall x (0 < \epsilon \rightarrow 0 < \delta \wedge (|x - a| < \delta \rightarrow |f(x) - f(a)| < \epsilon))$$

\Rightarrow_S

$$\forall \epsilon \forall a \forall x (0 < \epsilon \rightarrow 0 < d(\epsilon, a) \wedge (|x - a| < d(\epsilon, a) \rightarrow |f(x) - f(a)| < \epsilon))$$

Clausal Normal Form (Conjunctive Normal Form)

Rules to convert the matrix of the formula in Skolem normal form into a conjunction of disjunctions:

$$\begin{aligned}(F \leftrightarrow G) &\Rightarrow_K (F \rightarrow G) \wedge (G \rightarrow F) \\(F \rightarrow G) &\Rightarrow_K (\neg F \vee G) \\ \neg(F \vee G) &\Rightarrow_K (\neg F \wedge \neg G) \\ \neg(F \wedge G) &\Rightarrow_K (\neg F \vee \neg G) \\ \neg\neg F &\Rightarrow_K F \\(F \wedge G) \vee H &\Rightarrow_K (F \vee H) \wedge (G \vee H) \\(F \wedge \top) &\Rightarrow_K F \\(F \wedge \perp) &\Rightarrow_K \perp \\(F \vee \top) &\Rightarrow_K \top \\(F \vee \perp) &\Rightarrow_K F\end{aligned}$$

They are to be applied modulo associativity and commutativity of \wedge and \vee

In the Example

$$\forall \varepsilon \forall a \forall x (0 < \varepsilon \rightarrow 0 < d(\varepsilon, a) \wedge (|x - a| < d(\varepsilon, a) \rightarrow |f(x) - f(a)| < \varepsilon))$$

$$\Rightarrow_K$$

$$0 < d(\varepsilon, a) \vee \neg(0 < \varepsilon)$$

$$\neg(|x - a| < d(\varepsilon, a)) \vee |f(x) - f(a)| < \varepsilon \vee \neg(0 < \varepsilon)$$

Note: The universal quantifiers for the variables ε , a and x , as well as the conjunction symbol \wedge between the clauses are not written, for convenience

The Complete Picture

$$F \Rightarrow_P^* Q_1 y_1 \dots Q_n y_n G \quad (G \text{ quantifier-free})$$

$$\Rightarrow_S^* \forall x_1, \dots, x_m H \quad (m \leq n, H \text{ quantifier-free})$$

$$\Rightarrow_K^* \underbrace{\underbrace{\forall x_1, \dots, x_m}_{\text{leave out}} \bigwedge_{i=1}^k \underbrace{\bigvee_{j=1}^{n_i} L_{ij}}_{\text{clauses } C_i}}_{F'}$$

$N = \{C_1, \dots, C_k\}$ is called the **clausal (normal) form** (CNF) of F

Note: the variables in the clauses are implicitly universally quantified

Instead of showing that F is unsatisfiable, the proof problem from now is to show that N is unsatisfiable

Can do better than “searching through all interpretations”

Theorem: N is satisfiable iff it has a Herbrand model

Herbrand Interpretations

A **Herbrand interpretation** (over a given signature Σ) is a Σ -algebra \mathcal{A} such that

- The universe is the set T_Σ of ground terms over Σ (a **ground term** is a term without any variables):

$$U_{\mathcal{A}} = T_\Sigma$$

- Every function symbol from Σ is “mapped to itself”:

$$f_{\mathcal{A}} : (s_1, \dots, s_n) \mapsto f(s_1, \dots, s_n), \text{ where } f \text{ is } n\text{-ary function symbol in } \Sigma$$

Example

- $\Sigma_{Pres} = (\{0/0, s/1, +/2\}, \{</2, \leq/2\})$



$$U_{\mathcal{A}} = \{0, s(0), s(s(0)), \dots, 0 + 0, s(0) + 0, \dots, s(0 + 0), s(s(0) + 0), \dots\}$$

- $0 \mapsto 0, s(0) \mapsto s(0), s(s(0)) \mapsto s(s(0)), \dots, 0 + 0 \mapsto 0 + 0, \dots$

Herbrand Interpretations

Only interpretations $p_{\mathcal{A}}$ of predicate symbols $p \in \Sigma$ is undetermined in a Herbrand interpretation

- $p_{\mathcal{A}}$ represented as the set of ground atoms

$$\{p(s_1, \dots, s_n) \mid (s_1, \dots, s_n) \in p_{\mathcal{A}} \text{ where } p \in \Sigma \text{ is } n\text{-ary predicate symbol}\}$$

- Whole interpretation represented as $\bigcup_{p \in \Sigma} p_{\mathcal{A}}$

Example

- $\Sigma_{Pres} = (\{0/0, s/1, +/2\}, \{</2, \leq/2\})$ (from above)

- \mathbb{N} as Herbrand interpretation over Σ_{Pres}

$$I = \left\{ \begin{array}{l} 0 \leq 0, 0 \leq s(0), 0 \leq s(s(0)), \dots, \\ 0 + 0 \leq 0, 0 + 0 \leq s(0), \dots, \\ \dots, (s(0) + 0) + s(0) \leq s(0) + (s(0) + s(0)), \dots \end{array} \right\}$$

Herbrand's Theorem

Proposition

A Skolem normal form $\forall\phi$ is unsatisfiable iff it has no Herbrand model

Theorem (Skolem-Herbrand-Theorem)

$\forall\phi$ has no Herbrand model iff some finite set of ground instances $\{\phi\gamma_1, \dots, \phi\gamma_n\}$ is unsatisfiable

Applied to clause logic:

Theorem (Skolem-Herbrand-Theorem)

A set N of Σ -clauses is unsatisfiable iff some finite set of ground instances of clauses from N is unsatisfiable

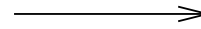
Leads immediately to theorem prover “Gilmore’s Method”

Gilmore's Method - Based on Herbrand's Theorem

Preprocessing:

Given Formula

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



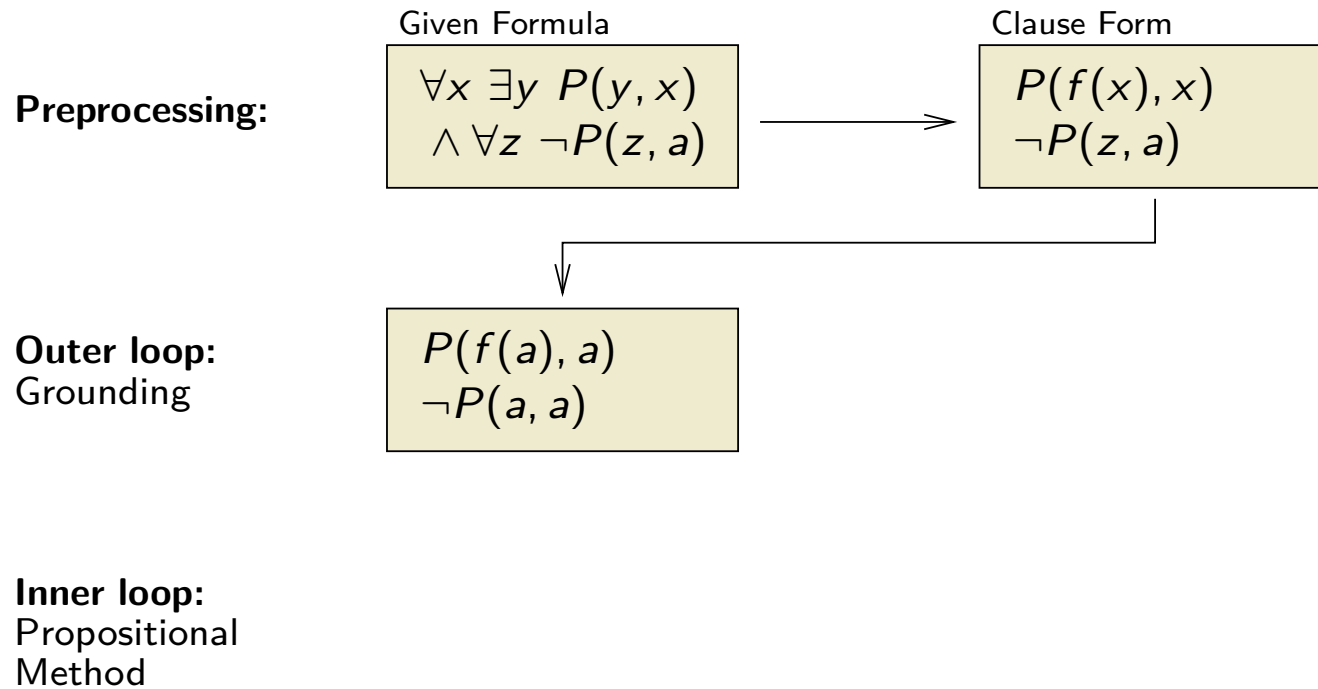
Clause Form

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

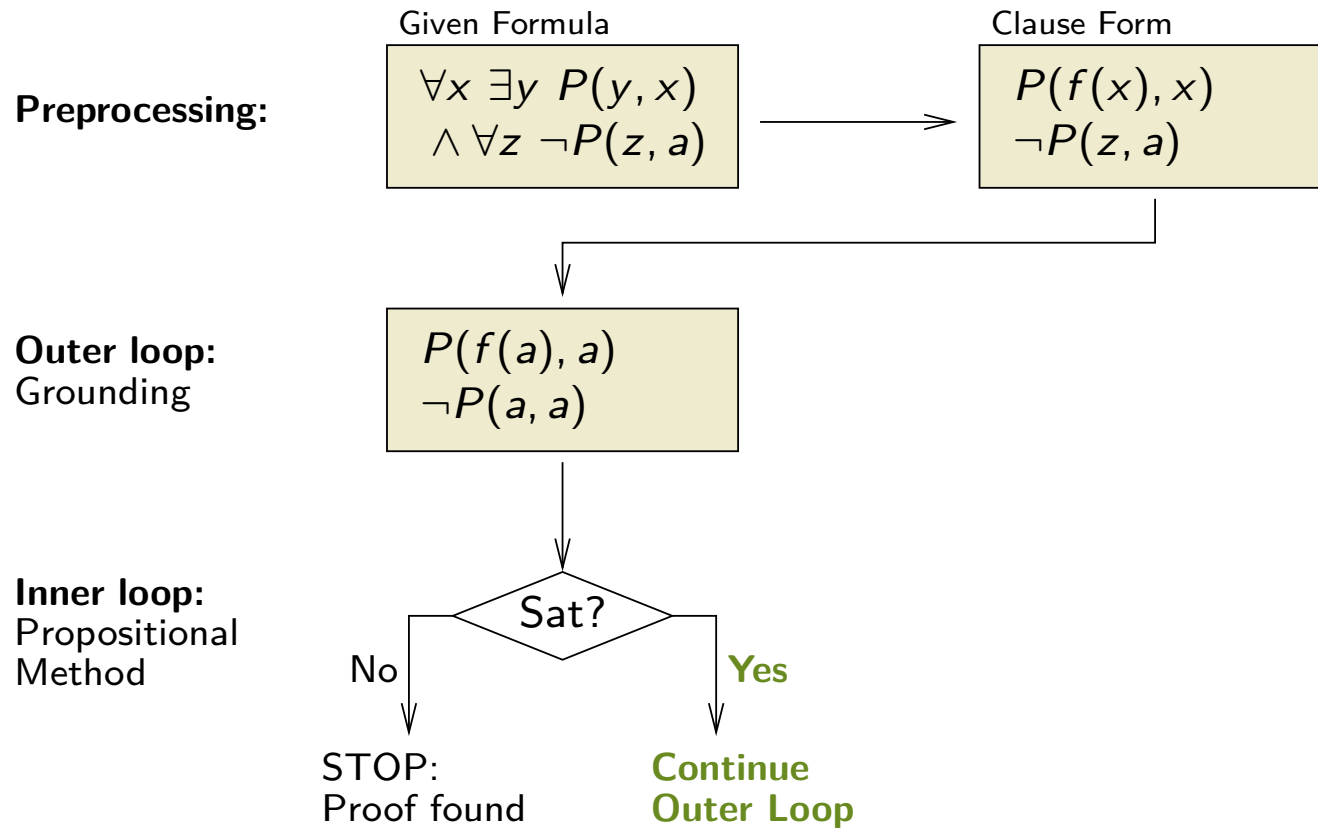
Outer loop:
Grounding

Inner loop:
Propositional
Method

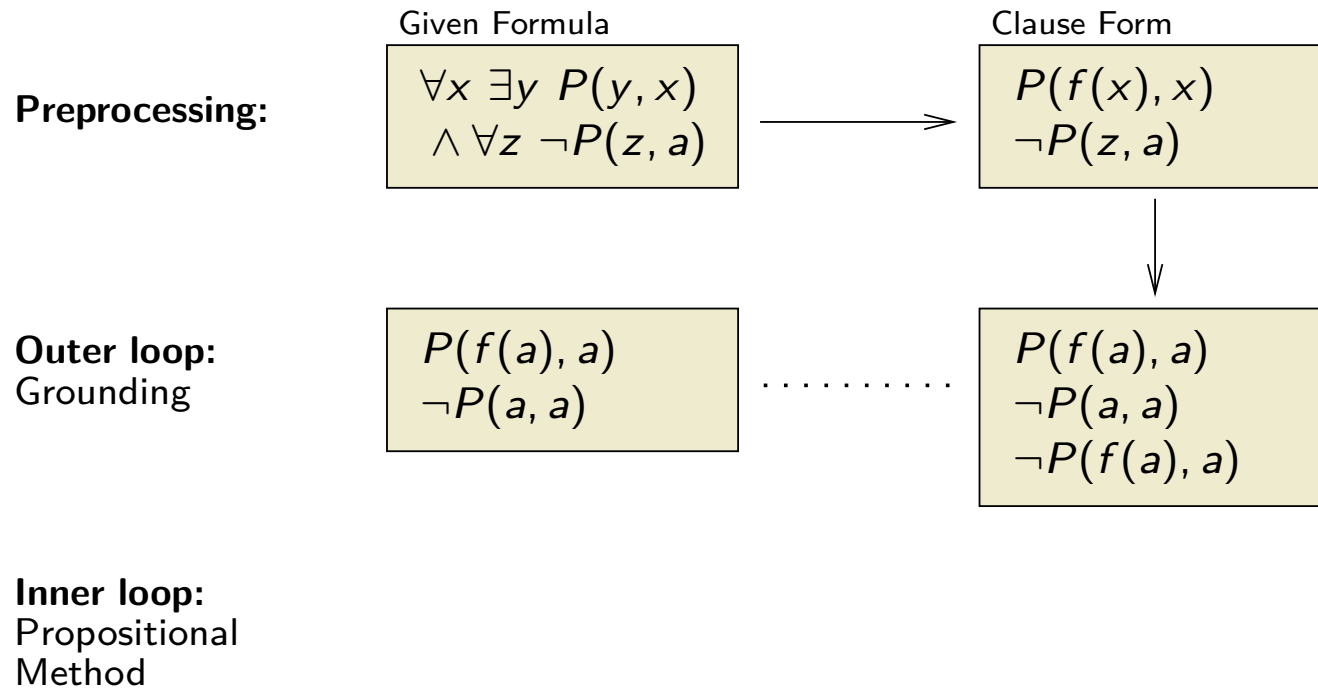
Gilmore's Method - Based on Herbrand's Theorem



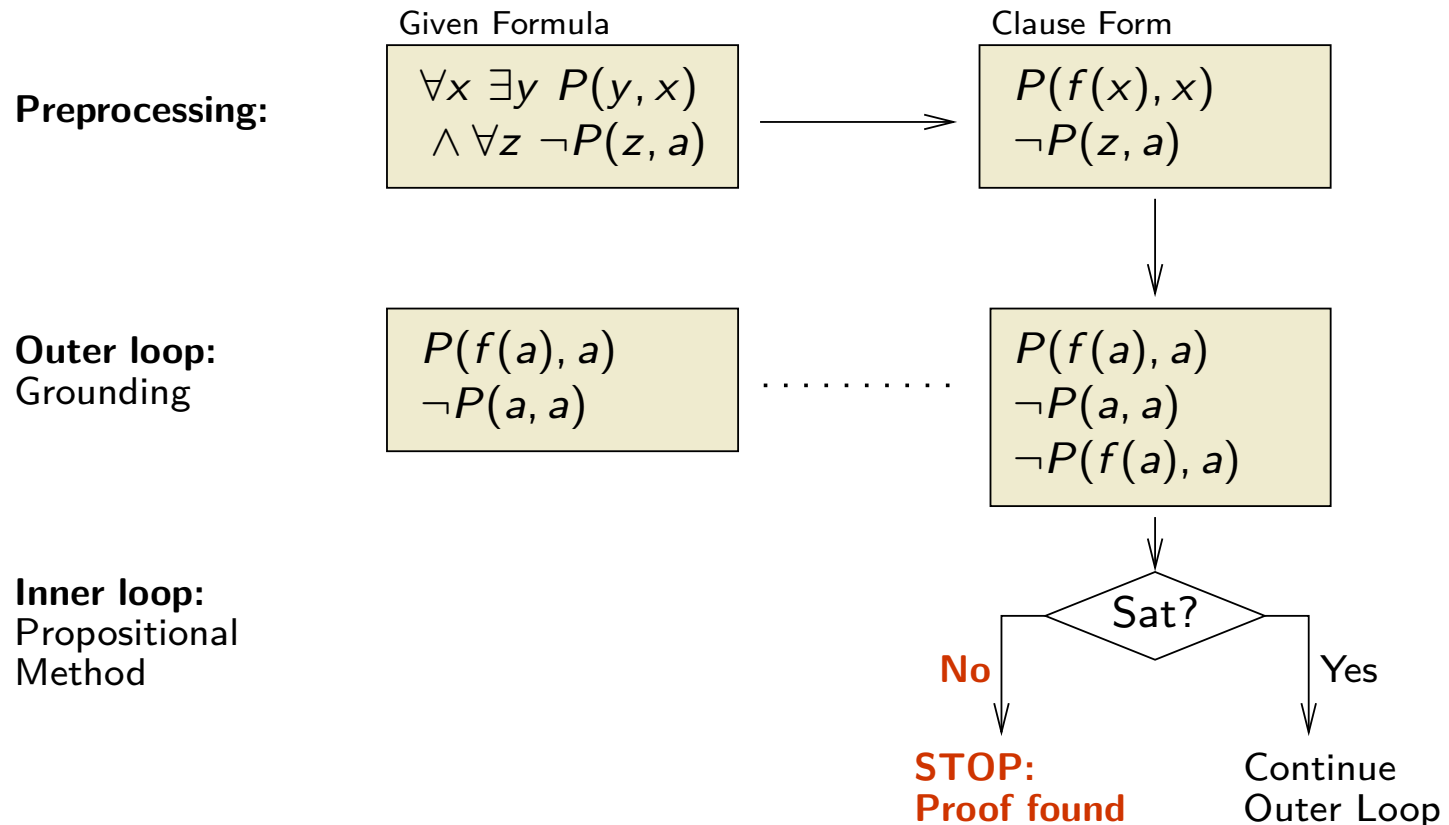
Gilmore's Method - Based on Herbrand's Theorem



Gilmore's Method - Based on Herbrand's Theorem



Gilmore's Method - Based on Herbrand's Theorem



Calculi for First-Order Logic Theorem Proving

- 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
- Resolution inferences operate directly on clauses, not on their ground instances

Next: propositional Resolution, lifting, first-order Resolution

The Propositional Resolution Calculus *Res*

Modern versions of the first-order version of the resolution calculus [Robinson 1965] are (still) the most important calculi for FOTP today.

Propositional resolution inference rule:

$$\frac{C \vee A \quad \neg A \vee D}{C \vee D}$$

Terminology: $C \vee D$: **resolvent**; A : **resolved atom**

Propositional (positive) factorisation inference rule:

$$\frac{C \vee A \vee A}{C \vee A}$$

These are **schematic inference rules**:

C and D – propositional clauses

A – propositional atom

“ \vee ” is considered associative and commutative

Sample Proof

1. $\neg A \vee \neg A \vee B$ (given)
2. $A \vee B$ (given)
3. $\neg C \vee \neg B$ (given)
4. C (given)
5. $\neg A \vee B \vee B$ (Res. 2. into 1.)
6. $\neg A \vee B$ (Fact. 5.)
7. $B \vee B$ (Res. 2. into 6.)
8. B (Fact. 7.)
9. $\neg C$ (Res. 8. into 3.)
10. \perp (Res. 4. into 9.)

Soundness of Propositional Resolution

Proposition

Propositional resolution is sound

Proof:

Let $I \in \Sigma\text{-Alg}$. To be shown:

1. for resolution: $I \models C \vee A, I \models D \vee \neg A \Rightarrow I \models C \vee D$
2. for factorization: $I \models C \vee A \vee A \Rightarrow I \models C \vee A$

Ad (i): Assume premises are valid in I . Two cases need to be considered:

(a) A is valid in I , or (b) $\neg A$ is valid in I .

a) $I \models A \Rightarrow I \models D \Rightarrow I \models C \vee D$

b) $I \models \neg A \Rightarrow I \models C \Rightarrow I \models C \vee D$

Ad (ii): even simpler

Completeness of Propositional Resolution

Theorem:

Propositional Resolution is refutationally complete

- That is, if a propositional clause set is unsatisfiable, then Resolution will derive the empty clause \perp eventually
- More precisely: If a clause set is unsatisfiable and closed under the application of the Resolution and Factorization inference rules, then it contains the empty clause \perp
- Perhaps easiest proof: semantic tree proof technique (see blackboard)
- This result can be considerably strengthened, some strengthenings come for free from the proof

Propositional resolution is not suitable for first-order clause sets

Lifting Propositional Resolution to First-Order Resolution

Propositional resolution

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

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

$$\frac{P(f(f(a)), f(f(a))) \quad \neg P(f(f(a)), f(f(a)))}{\perp}$$

Unification

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

$$\frac{P(f(x), y) \quad \neg P(z, z)}{\perp}$$

Justified by existence of $P(f(x), f(x))$

Can represent infinitely many propositional resolution inferences

Substitutions and Unifiers

- A **substitution** σ is a mapping from variables to terms which is the identity almost everywhere

Example: $\sigma = [y \mapsto f(x), z \mapsto f(x)]$

- A substitution can be **applied** to a term or atom t , written as $t\sigma$

Example, where σ is from above: $P(f(x), y)\sigma = P(f(x), f(x))$

- A substitution γ is a **unifier** of s and t iff $s\gamma = t\gamma$

Example: $\gamma = [x \mapsto a, y \mapsto f(a), z \mapsto f(a)]$ is a unifier of $P(f(x), y)$ and $P(z, z)$

- A unifier σ of s is **most general** iff for every unifier γ of s and t there is a substitution δ such that $\gamma = \sigma \circ \delta$; notation: $\sigma = \text{mgu}(s, t)$

Example: $\sigma = [y \mapsto f(x), z \mapsto f(x)] = \text{mgu}(P(f(x), y), P(z, z))$

There are (linear) algorithms to compute mgu's or return "fail"

Resolution for First-Order Clauses

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

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

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

Example

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

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

Completeness of First-Order Resolution

Theorem: Resolution is **refutationally complete**

- That is, if a clause set is unsatisfiable, then Resolution will derive the empty clause \perp eventually
- More precisely: If a clause set is unsatisfiable and closed under the application of the Resolution and Factorization inference rules, then it contains the empty clause \perp
- Perhaps easiest proof: Herbrand Theorem + completeness of propositional resolution + Lifting Theorem (see blackboard)

Lifting Theorem: the conclusion of any propositional inference on ground instances of first-order clauses can be obtained by instantiating the conclusion of a first-order inference on the first-order clauses

- Closure can be achieved by the “Given Clause Loop”

The “Given Clause Loop”

As used in the Otter theorem prover:

Lists of clauses maintained by the algorithm: usable and sos.

Initialize sos with the input clauses, usable empty.

Algorithm (straight from the Otter manual):

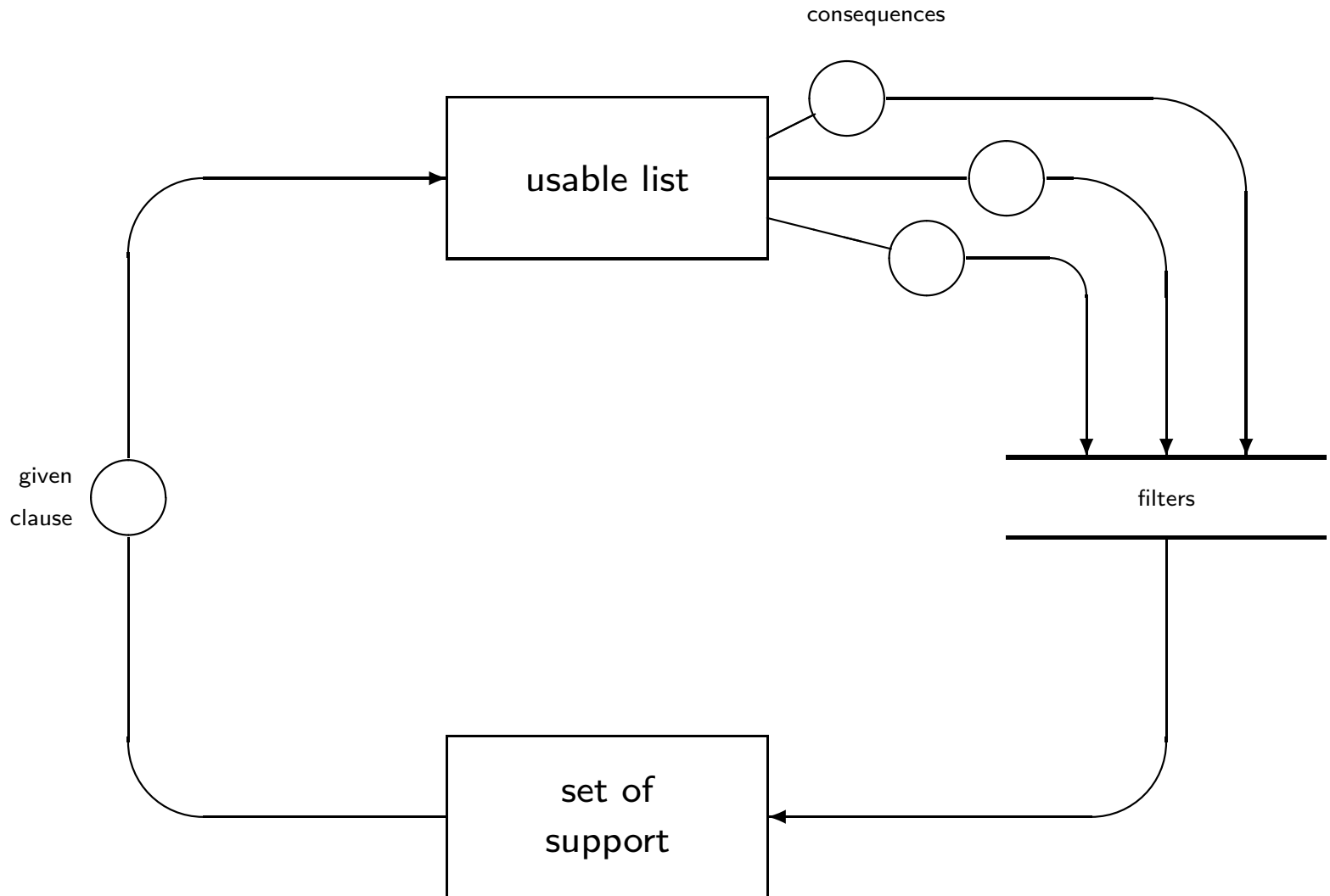
While (sos is not empty and no refutation has been found)

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: define clause weight e.g. as “depth + length” of clause.

The “Given Clause Loop” - Graphically



Calculi for First-Order Logic Theorem Proving

Recall:

- 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

Resolution inferences operate directly on clauses, not on their ground instances

Next: Instance-Based Method “Inst-Gen”

Idea: “semantic” guidance: add only instances that are falsified by a “candidate model”

Eventually, all repairs will be made or there is no more candidate model

Important notation: \perp denotes both a unique constant and a substitution that maps every variable to \perp

Example (S is “current clause set”):

$$S : \begin{array}{l} P(x, y) \vee P(y, x) \\ \neg P(x, x) \end{array}$$

$$S\perp : \begin{array}{l} P(\perp, \perp) \vee P(\perp, \perp) \\ \neg P(\perp, \perp) \end{array}$$

Analyze $S\perp$:

Case 1: SAT detects unsatisfiability of $S\perp$

Then Conclude S is unsatisfiable

But what if $S\perp$ is satisfied by some model, denoted by I_\perp ?

Main idea: associate to model I_{\perp} of S_{\perp} a **candidate model** I_S of S .

Calculus goal: add instances to S so that I_S becomes a model of S

Example:

$$S : \begin{array}{l} \underline{P(x) \vee Q(x)} \\ \underline{\neg P(a)} \end{array} \qquad S_{\perp} : \begin{array}{l} \underline{P(\perp) \vee Q(\perp)} \\ \underline{\neg P(a)} \end{array}$$

Analyze S_{\perp} :

Case 2: SAT detects model $I_{\perp} = \{P(\perp), \neg P(a)\}$ of S_{\perp}

Case 2.1: candidate model $I_S = \{\neg P(a)\}$ derived from
literals selected in S by I_{\perp} is not a model of S

Add “problematic” instance $P(a) \vee Q(a)$ to S to refine I_S

Clause set after adding $P(a) \vee Q(a)$

$$S : \begin{array}{l} \underline{P(x)} \vee Q(x) \\ P(a) \vee \underline{Q(a)} \\ \underline{\neg P(a)} \end{array}$$

$$S_{\perp} : \begin{array}{l} \underline{P(\perp)} \vee Q(\perp) \\ P(a) \vee \underline{Q(a)} \\ \underline{\neg P(a)} \end{array}$$

Analyze S_{\perp} :

Case 2: SAT detects model $I_{\perp} = \{P(\perp), Q(a), \neg P(a)\}$ of S_{\perp}

Case 2.2: candidate model $I_S = \{Q(a), \neg P(a)\}$ derived from literals selected in S by I_{\perp} **is** a model of S

Then conclude S is satisfiable

How to derive candidate model I_S ?

Inst-Gen - Model Construction

It provides (partial) interpretation for S_{ground} for given clause set S

$$S : \begin{array}{l} \underline{P(x)} \vee Q(x) \\ P(a) \vee \underline{Q(a)} \\ \underline{\neg P(a)} \end{array} \quad \Sigma = \{a, b\}, S_{\text{ground}} : \begin{array}{l} \underline{P(b)} \vee Q(b) \\ P(a) \vee \underline{Q(a)} \\ \underline{\neg P(a)} \end{array}$$

- For each $C_{\text{ground}} \in S_{\text{ground}}$ find most specific $C \in S$ that can be instantiated to C_{ground}
- Select literal in C_{ground} corresponding to selected literal in that C
- Add selected literal of that C_{ground} to I_S if not in conflict with I_S

Thus, $I_S = \{P(b), Q(a), \neg P(a)\}$

Model Generation

Scenario: no “theorem” to prove, or disprove a “theorem”

A model provides further information then

Why compute models?

Planning: Can be formalised as propositional satisfiability problem.

[Kautz& Selman, AAAI96; Dimopolous et al, ECP97]

Diagnosis: Minimal models of *abnormal* literals (circumscription). [Reiter, AI87]

Databases: View materialisation, View Updates, Integrity Constraints.

Nonmonotonic reasoning: Various semantics (GCWA, Well-founded, Perfect, Stable, . . .), all based on minimal models. [Inoue et al, CADE 92]

Software Verification: Counterexamples to conjectured theorems.

Theorem proving: Counterexamples to conjectured theorems.

Finite models of quasigroups, (MGTP/G). [Fujita et al, IJCAI 93]

Model Generation

Why compute models (cont'd)?

Natural Language Processing:

- Maintain models $\mathcal{J}_1, \dots, \mathcal{J}_n$ as different readings of discourses:

$$\mathcal{J}_i \models BG\text{-Knowledge} \cup Discourse_so_far$$

- Consistency checks (“Mia’s husband loves Sally. She is not married.”)

$$BG\text{-Knowledge} \cup Discourse_so_far \not\models \neg New_utterance$$

iff $BG\text{-Knowledge} \cup Discourse_so_far \cup New_utterance$ is **satisfiable**

- Informativity checks (“Mia’s husband loves Sally. She is married.”)

$$BG\text{-Knowledge} \cup Discourse_so_far \not\models New_utterance$$

iff $BG\text{-Knowledge} \cup Discourse_so_far \cup \neg New_utterance$ is **satisfiable**

Example - Group Theory

The following axioms specify a group

$$\forall x, y, z : (x * y) * z = x * (y * z) \quad (\text{associativity})$$

$$\forall x : e * x = x \quad (\text{left - identity})$$

$$\forall x : i(x) * x = e \quad (\text{left - inverse})$$

Does

$$\forall x, y : x * y = y * x \quad (\text{commutat.})$$

follow?

No, it does not

Example - Group Theory

Counterexample: a group with finite domain of size 6, where the elements 2 and 3 are not commutative: Domain: $\{1, 2, 3, 4, 5, 6\}$

$e : 1$

$i :$

	1	2	3	4	5	6
	1	2	3	5	4	6

$*$:

	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	1	4	3	6	5
3	3	5	1	6	2	4
4	4	6	2	5	1	3
5	5	3	6	1	4	2
6	6	4	5	2	3	1

Finite Model Finders - Idea

- Assume a fixed domain size n .
- Use a tool to decide if there exists a model with domain size n for a given problem.
- Do this starting with $n = 1$ with increasing n until a model is found.
- Note: domain of size n will consist of $\{1, \dots, n\}$.

1. Approach: SEM-style

- Tools: SEM, Finder, Mace4
- Specialized constraint solvers.
- For a given domain generate all ground instances of the clause.
- Example: For domain size 2 and clause $p(a, g(x))$ the instances are $p(a, g(1))$ and $p(a, g(2))$.

1. Approach: SEM-style

- Set up multiplication tables for all symbols with the whole domain as cell values.
- Example: For domain size 2 and function symbol g with arity 1 the cells are $g(1) = \{1, 2\}$ and $g(2) = \{1, 2\}$.
- Try to restrict each cell to exactly 1 value.
- The clauses are the constraints guiding the search and propagation.
- Example: if the cell of a contains $\{1\}$, the clause $a = b$ forces the cell of b to be $\{1\}$ as well.

2. Approach: Mace-style

- Tools: Mace2, Paradox
- For given domain size n transform first-order clause set into equisatisfiable propositional clause set.
- Original problem has a model of domain size n iff the transformed problem is satisfiable.
- Run SAT solver on transformed problem and translate model back.

Paradox - Example

Domain:	$\{1, 2\}$
Clauses:	$\{p(a) \vee f(x) = a\}$
Flattened:	$p(y) \vee f(x) = y \vee a \neq y$
Instances:	$p(1) \vee f(1) = 1 \vee a \neq 1$ $p(2) \vee f(1) = 1 \vee a \neq 2$ $p(1) \vee f(2) = 1 \vee a \neq 1$ $p(2) \vee f(2) = 1 \vee a \neq 2$
Totality:	$a = 1 \vee a = 2$ $f(1) = 1 \vee f(1) = 2$ $f(2) = 1 \vee f(2) = 2$
Functionality:	$a \neq 1 \vee a \neq 2$ $f(1) \neq 1 \vee f(1) \neq 2$ $f(2) \neq 1 \vee f(2) \neq 2$

A model is obtained by setting the **blue literals** true

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)
- “An equational theory” $\models \exists s_1 = t_1 \wedge \dots \wedge s_n = t_n$
(E-Unification problem)
- “Some group theory” $\models s = t$ (Word problem)

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

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, inference rules like paramodulation

Satisfiability Modulo Theories (SMT)

- Decide the T -validity problem, $T \models \phi$?
- Usual restriction: ϕ is quantifier-free, i.e. all variables implicitly universally quantified
- Applications in particular to formal verification

Checking Satisfiability Modulo Theories

Given: A quantifier-free formula ϕ (implicitly existentially quantified)

Task: Decide whether ϕ is T -satisfiable

(T -validity via “ $T \models \forall \phi$ ” iff “ $\exists \neg\phi$ is not T -satisfiable”)

Approach: eager translation into SAT

- Encode problem into a T -equisatisfiable propositional formula
- Feed formula to a SAT-solver
- Example: $T =$ equality (Ackermann encoding)

Approach: lazy translation into SAT

- Couple a SAT solver with a given 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

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

Theory: Equality

Lazy Translation into SAT

$$\underbrace{g(a) = c}_1 \wedge \underbrace{f(g(a)) \neq f(c)}_{\bar{2}} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{\bar{4}}$$

Lazy Translation into SAT

$$\underbrace{g(a) = c}_1 \wedge \underbrace{f(g(a)) \neq f(c)}_{\bar{2}} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{\bar{4}}$$

- Send $\{1, \bar{2} \vee 3, \bar{4}\}$ to SAT solver.

Lazy Translation into SAT

$$\underbrace{g(a) = c}_1 \wedge \underbrace{f(g(a)) \neq f(c)}_{\bar{2}} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{\bar{4}}$$

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

Lazy Translation into SAT

$$\underbrace{g(a) = c}_1 \wedge \underbrace{f(g(a)) \neq f(c)}_{\bar{2}} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{\bar{4}}$$

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

Lazy Translation into SAT

$$\underbrace{g(a) = c}_1 \wedge \underbrace{f(g(a)) \neq f(c)}_{\bar{2}} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{\bar{4}}$$

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

Lazy Translation into SAT

$$\underbrace{g(a) = c}_1 \wedge \underbrace{f(g(a)) \neq f(c)}_{\bar{2}} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{\bar{4}}$$

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

Lazy Translation into SAT: Summary

- Abstract T -atoms as propositional variables
- SAT solver computes a model, i.e. satisfying boolean assignment for propositional abstraction (or fails)
- Solution from SAT solver may not be a T -model. If so,
 - Refine (strengthen) propositional formula by incorporating reason for false solution
 - Start again with computing a model

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 restarting from scratch)

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

Combining Theories

Theories:

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

Problem: Is

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

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:

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

Nelson-Oppen Combination Method

Variable abstraction + equality propagation:

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

Nelson-Oppen Combination Method

Variable abstraction + equality propagation:

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

\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:

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

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

Nelson-Oppen Combination Method

Variable abstraction + equality propagation:

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

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

Nelson-Oppen Combination Method

Variable abstraction + equality propagation:

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

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

Nelson-Oppen Combination Method

Variable abstraction + equality propagation:

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

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

Nelson-Oppen Combination Method

Variable abstraction + equality propagation:

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

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

Nelson-Oppen Combination Method

Variable abstraction + equality propagation:

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

v_2

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

Conclusions

- Talked about the role of first-order theorem proving
- Talked about some standard techniques (Normal forms of formulas, Resolution calculus, unification, Instance-based method, Model computation)
- Talked about DPLL and Satisfiability Modulo Theories (SMT)

Further Topics

- Redundancy elimination, efficient equality reasoning, adding arithmetics to first-order theorem provers
- FOTP methods as decision procedures in special cases
E.g. reducing planning problems and temporal logic model checking problems to function-free clause logic and using an instance-based method as a decision procedure
- Implementation techniques
- Competition CASC and TPTP problem library
- Instance-based methods (a lot to do here, cf. my home page)
Attractive because of complementary features to more established methods