Speaking Logic

N. Shankar

Computer Science Laboratory
SRI International
Menlo Park, CA

May 27, 2012

Why Logic?

- Computing, like mathematics, is the study of reusable abstractions.
- Abstractions in computing include numbers, lists, channels, processes, protocols, and programming languages.
- These abstractions have algorithmic value in designing, representing, and reasoning about computational processes.
- Logic is the calculus of computing — it is used to delineate the precise meaning and scope of these abstractions, and to calculate at the abstract level.

The Unreasonable Effectiveness of Logic in Computing

- Logic has been unreasonably effective in computing, with an impact that spans
  - Theoretical computer science
  - Hardware design and verification
  - Software verification
  - Computer security
  - Programming languages
  - Artificial intelligence.
  - Databases
- Our course is about the effective use of logic in computing.

Speaking Logic

- In mathematics, logic is studied as a source of interesting (meta-)theorems, but the reasoning is typically informal.
- In philosophy, logic is studied as a minimal set of foundational principles from which knowledge can be derived.
- Computing involves *using* rigorous, possibly formal, logical reasoning.
- Logic is a medium for problem specification and an aid to creative problem solving.
- We will examine how logic is used to formulate problems, find solutions, and build proofs.
- We will also examine useful metalogical properties of logics, as well as algorithmic methods for effective inference.

Course Schedule

- The course is spread over six hours:
  - **10AM-noon:** Propositional Logics
  - **1PM-3PM:** First and Higher-Order Logic
  - **3.30PM-5.30PM:** Automated Tools
- The goal is to learn how to speak logic fluently through the use of propositional, modal, equational, first-order, and higher-order logic.
- This will serve as a background for the more sophisticated ideas in the main lectures.

A Small Puzzle [Wason]

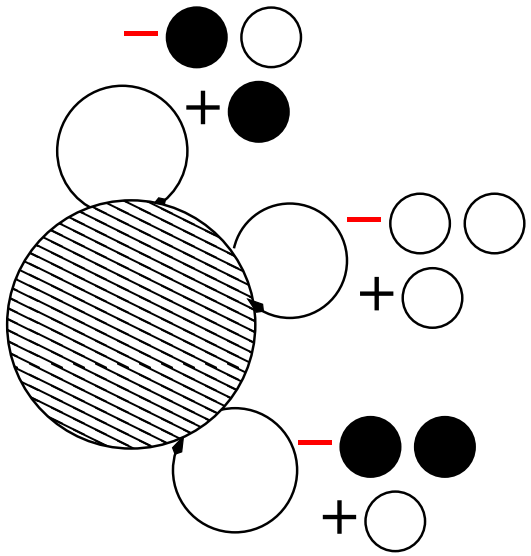
- Given four cards laid out on a table as: $\boxed{\text{D}}$, $\boxed{3}$, $\boxed{\text{F}}$, $\boxed{7}$, where each card has a letter on one side and a number on the other.
- Which cards should you flip over to determine if every card with a $\boxed{\text{D}}$ on one side has a $\boxed{7}$ on the other side?

A Small Problem

Given a bag containing some black balls and white balls, and a stash of black/white balls. Repeatedly

1. Remove a random pair of balls from the bag
2. If they are the same color, insert a white ball into the bag
3. If they are of different colors, insert a black ball into the bag

What is the color of the last ball?



Truthtellers and Liars [Smullyan]

- You are confronted with two gates.
- One gate leads to the castle, and the other leads to a trap
- There are two guards at the gates: one always tells the truth, and the other always lies.
- You are allowed to ask one of the guards on question with a yes/no answer.
- What question should you ask in order to find out which gate leads to the castle?

The Monty Hall Problem

- There are three doors with a car behind one, and goats behind the other two.
- You have chosen one door.
- Monty Hall, knowing where the car is hidden, opens one of the other two doors to reveal a goat.
- He allows you to switch your choice to the other closed door.
- If you want to win the car, should you switch?

Mr. S and Mr. P

- Two integers m and n are picked from the interval $[2, 99]$.
- Mr. S is given the sum $m + n$. and Mr. P is given the product mn.
- They then have the following dialogue:

  S: *I don't know m and n.*
  P: *Me neither.*
  S: *I know that you don't.*
  P: *In that case, I do know m and n.*
  S: *Then, I do too.*

- How would you determine the numbers m and n?

Gilbreath's Card Trick

- Start with a deck consisting of a stack of quartets, where the cards in each quartet appear in suit order ♠, ♡, ♣, ♢:

$$\langle 5\spadesuit\rangle, \langle 3\heartsuit\rangle, \langle Q\clubsuit\rangle, \langle 8\diamondsuit\rangle,$$
$$\langle K\spadesuit\rangle, \langle 2\heartsuit\rangle, \langle 7\clubsuit\rangle, \langle 4\diamondsuit\rangle,$$
$$\langle 8\spadesuit\rangle, \langle J\heartsuit\rangle, \langle 9\clubsuit\rangle, \langle A\diamondsuit\rangle$$

- Cut the deck, say as $\langle 5\spadesuit\rangle, \langle 3\heartsuit\rangle, \langle Q\clubsuit\rangle, \langle 8\diamondsuit\rangle, \langle K\spadesuit\rangle$ and $\langle 2\heartsuit\rangle, \langle 7\clubsuit\rangle, \langle 4\diamondsuit\rangle, \langle 8\spadesuit\rangle, \langle J\heartsuit\rangle, \langle 9\clubsuit\rangle, \langle A\diamondsuit\rangle$.
- Reverse one of the decks as $\langle K\spadesuit\rangle, \langle 8\diamondsuit\rangle, \langle Q\clubsuit\rangle, \langle 3\heartsuit\rangle, \langle 5\spadesuit\rangle$.
- Now shuffling, for example, as

$$\langle 2\heartsuit\rangle, \langle 7\clubsuit\rangle, \underline{\langle K\spadesuit\rangle}, \underline{\langle 8\diamondsuit\rangle},$$
$$\langle 4\diamondsuit\rangle, \langle 8\spadesuit\rangle, \underline{\langle Q\clubsuit\rangle}, \langle J\heartsuit\rangle,$$
$$\underline{\langle 3\heartsuit\rangle}, \langle 9\clubsuit\rangle, \underline{\langle 5\spadesuit\rangle}, \langle A\diamondsuit\rangle$$

- *Each quartet contains a card from each suit. Why?*

Pigeonhole Principle

Why can't you park $n+1$ cars in n parking spaces, if each car needs its own space?

Hard Sudoku [Wikipedia/Algorithmics_of_Sudoku]

					3		8	5
		1		2				
			5		7			
		4				1		
	9							
5							7	3
		2		1				
				4				9

What is Logic?

- Logic is the art and science of effective reasoning.
- How can we draw general and reliable conclusions from a collection of facts?
- Formal logic: Precise, syntactic characterizations of well-formed expressions and valid deductions.
- Formal logic makes it possible to *calculate* consequences so that each step is verifiable by means of proof.
- Computers can be used to automate such symbolic calculations.

Logic Basics

- Logic studies the *trinity* between *language*, *interpretation*, and *proof*.
- *Language* circumscribes the syntax that is used to construct sensible assertions.
- *Interpretation* ascribes an intended sense to these assertions by *fixing* the meaning of certain symbols, e.g., the logical connectives, equality, and *delimiting the variation* in the meanings of other symbols, e.g., variables, functions, and predicates.
- An assertion is *valid* if it holds in all interpretations.
- Checking validity through interpretations is not always efficient and often, not even possible, so *proofs* in the form axioms and inference rules are used to demonstrate the validity of assertions.

Propositional Logic

- Propositional logic can be more accurately described as a logic of conditions – *propositions are always true or always false.* [Couturat, *Algebra of Logic*]
- A condition can be represented by a propositional variable, e.g., p, q, etc., so that distinct propositional variables can range over possibly different conditions.
- The conjunction, disjunction, and negation of conditions are also conditions.
- The syntactic representation of conditions is using propositional formulas:

$$\phi := P \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2$$

- P is a class of propositional variables: $p_0, p_1, \ldots$.
- Examples of formulas are p, $p \wedge \neg p$, $p \vee \neg p$, $(p \wedge \neg q) \vee \neg p$.
- Define the operation of substituting a formula A for a variable p in a formula B, i.e., $B[p \mapsto A]$. Is the result always a well-formed formula? Can the variable p occur in $B[p \mapsto A]$?

Meaning

- In logic, the meaning of an expression is constructed compositionally from the meanings of its subexpressions.
- The meanings of the symbols are either *fixed*, as with $\neg$, $\wedge$, and $\vee$, or allowed to vary, as with the propositional variables.
- An interpretation (truth assignment) M assigns truth values $\{\top, \bot\}$ to propositional variables: $M(p) = \top \iff M \models p$.
- $M[\![A]\!]$ is the meaning of A in M and is computed using truth tables:

ϕ	p	q	$\neg p$	$p \vee q$	$p \wedge q$
$M_1(\phi)$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$
$M_2(\phi)$	$\bot$	$\top$	$\top$	$\top$	$\bot$
$M_3(\phi)$	$\top$	$\bot$	$\bot$	$\top$	$\bot$
$M_4(\phi)$	$\top$	$\top$	$\bot$	$\top$	$\top$

Truth Tables

We can use truth tables to evaluate formulas for validity/satisfiability.

p	q	$(\neg p \vee q)$	$(\neg(\neg p \vee q) \vee p)$	$\neg(\neg(\neg p \vee q) \vee p) \vee p$
$\bot$	$\bot$	$\top$	$\bot$	$\top$
$\bot$	$\top$	$\top$	$\bot$	$\top$
$\top$	$\bot$	$\bot$	$\top$	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

How many rows are there in the truth table for a formula with n distinct propositional variables?

Defining New Connectives

- How do you define $\wedge$ in terms of $\neg$ and $\vee$?
- Give the truth table for $A \Rightarrow B$ and define it in terms of $\neg$ and $\vee$.
- Define bi-implication $A \iff B$ in terms of $\Rightarrow$ and $\wedge$ and show its truth table.
- An n-ary Boolean function maps $\{\top, \bot\}^n$ to $\{\top, \bot\}$
- Show that every n-ary Boolean function can be defined using $\neg$ and $\vee$.
- Using $\neg$ and $\vee$ define an n-ary parity function which evaluates to $\top$ iff the parity is odd.
- Define an n-ary function which determines that the unsigned value of the little-endian input $p_0, \ldots, p_{n-1}$ is even?
- Define the *NAND* operation, where $NAND(p, q)$ is $\neg(p \wedge q)$ using $\neg$ and $\vee$.

Satisfiability and Validity

- An interpretation M is a model of a formula ϕ if $M \models \phi$.
- If $M \models \neg\phi$, then M is a *countermodel* for ϕ.
- When ϕ has a model, it is said to be *satisfiable*.
- If it has no model, then it is *unsatisfiable*.
- If $\neg\phi$ is unsatisfiable, then ϕ is valid, i.e., alway evaluates to $\top$.
- We write $\phi \models \psi$ if every model of ϕ is a model of ψ.
- If $\phi \wedge \neg\psi$ is unsatisfiable, then $\phi \models \psi$.

Which Formulas are Satisfiable/Unsatisfiable/Valid?

- $p \vee \neg p$
- $p \wedge \neg p$
- $\neg p \Rightarrow p$
- $((p \Rightarrow q) \Rightarrow p) \Rightarrow p$

Some Valid Laws

- $\neg(A \wedge B) \iff \neg A \vee \neg B$
- $\neg(A \vee B) \iff \neg A \wedge \neg B$
- $((A \vee B) \vee C) \iff A \vee (B \vee C)$
- $(A \Rightarrow B) \iff (\neg A \vee B)$
- $(\neg A \Rightarrow \neg B) \iff (B \Rightarrow A)$
- $\neg\neg A \iff A$
- $A \Rightarrow B \iff \neg A \vee B$
- $\neg(A \wedge B) \iff \neg A \vee \neg B$
- $\neg(A \vee B) \iff \neg A \wedge \neg B$
- $\neg A \Rightarrow B \iff \neg B \Rightarrow A$

What Can Propositional Logic Express?

- Constraints over bounded domains can be expressed as satisfiability problems in propositional logic (SAT).
- Define a 1-bit full adder in propositional logic.
- The Pigeonhole Principle states that if $n+1$ pigeons are assigned to n holes, then some hole must contain more than one pigeon. Formalize the pigeonhole principle for four pigeons and three holes.
- Write a propositional formula for checking that a given finite automaton $\langle Q, \Sigma, q, F, \delta \rangle$ with alphabet Σ, set of states S, initial state q, set of final states F, and transition function δ from $\langle Q, \Sigma \rangle$ to Q accepts some string of length 5.
- Formalize the statement that a graph of n elements is k-colorable for given k and n such that $k < n$.
- Formalize and prove the statement that given a symmetric and transitive graph over 3 elements, either the graph is complete or contains an isolated point.
- Formalize *Sudoku* and Latin Squares in propositional logic.

Cook's Theorem

- A Turing machine consists of a finite automaton reading (and writing) symbols from a tape.
- The finite automaton reads the symbol at the current position of the head, and
  1. Writes a new symbol at the head position
  2. Moves the head by a step either to the left or right of the current position
  3. Transitions to the next state of the finite automaton
- A Turing machine is nondeterministic if the transition function computes a set of states.
- Show that SAT is solvable in polynomial time (in the size of the input) by a nondeterministic Turing machine.
- Show that for any nondeterministic Turing machine and polynomial bound $p(n)$ for input tape of size n, one can (in polynomial time) construct a propositional formula which is satisfiable iff there is a terminating computation of the Turing machine on the input.

Proof Systems

- There are three basic styles of proof systems.
- These are distinguished by their basic judgement.
  1. Hilbert systems: A formula is provable.
  2. Natural deduction: A formula is provable from a set of formulas.
  3. Sequent Calculus: Some consequent formula is a consequence of the antecedent formulas.

Hilbert System (H) for Propositional Logic

- The basic judgement here is $\vdash A$ asserting that a formula is *provable*.
- We can pick $\Rightarrow$ as the basic connectives
- The axioms are
  - $\dfrac{}{\vdash A \Rightarrow A}$
  - $\dfrac{}{\vdash A \Rightarrow (B \Rightarrow A)}$
  - $\dfrac{}{(\vdash A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))}$
- A single rule of inference (Modus Ponens) is given

$$\frac{\vdash A \qquad \vdash A \Rightarrow B}{\vdash B}$$

- Can you prove $((p \Rightarrow q) \Rightarrow p) \Rightarrow p$ using the above system?

Deduction Theorem

- We write $\Gamma \vdash A$ for a set of formulas Γ, if $\vdash A$ can be proved given $\vdash B$ for each $B \in \Gamma$.
- Deduction theorem: Show that if $\Gamma, A \vdash B$, then $\Gamma \vdash A \Rightarrow B$, where Γ, A is $\Gamma \cup \{A\}$.
- A *derived* rule of inference has the form

$$\frac{P_1, \ldots, P_n}{C}$$

  where there is a derivation in the base logic from the premises $P_1, \ldots, P_n$ to the conclusion C.
- An *admissible* rule of inference is one where the premises $P_1, \ldots, P_n$ must be provable for C to be provable.

Natural Deduction for Propositional Logic

- In natural deduction (ND), the basic judgement is $\Gamma \vdash A$.
- The rules are classified according to the introduction or elimination of connectives from A in $\Gamma \vdash A$.
- The axiom, introduction, and elimination rules of natural deduction are
  - $\dfrac{}{\Gamma, A \vdash A}$
  - $\dfrac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash A \Rightarrow B}{\Gamma_1 \cup \Gamma_2 \vdash B}$
  - $\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}$
- Use ND to prove the axioms of the Hilbert system.
- A proof is in normal form if no introduction rule appears above an elimination rule. Can you ensure that your proofs are always in normal form? Can you write an algorithm to convert non-normal proofs to normal ones.

Sequent Calculus (LK) for Propositional Logic

The basic judgement is $\Gamma \vdash \Delta$ asserting that $\bigwedge \Gamma \Rightarrow \bigvee \Delta$, where Γ and Δ are sets (or bags) of formulas.

	Left	Right
Ax	$\dfrac{}{\Gamma, A \vdash A, \Delta}$	
$\neg$	$\dfrac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}$	$\dfrac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}$
$\vee$	$\dfrac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}$	$\dfrac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}$
$\wedge$	$\dfrac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}$	$\dfrac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}$
$\Rightarrow$	$\dfrac{\Gamma, B \vdash \Delta \quad \Gamma \vdash A, \Delta}{\Gamma, A \Rightarrow B \vdash \Delta}$	$\dfrac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \Rightarrow B, \Delta}$
Cut	$\dfrac{\Gamma \vdash A, \Delta \quad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta}$	

Peirce's Formula

- A sequent calculus proof of Peirce's formula $((p \Rightarrow q) \Rightarrow p) \Rightarrow p$ is given by

$$\dfrac{\dfrac{\dfrac{\dfrac{}{p \vdash p, q}\,Ax}{\vdash p, p \Rightarrow q}\,\vdash\Rightarrow \quad \dfrac{}{p \vdash p}\,Ax}{(p \Rightarrow q) \Rightarrow p \vdash p}\,\Rightarrow\vdash}{\vdash ((p \Rightarrow q) \Rightarrow p) \Rightarrow p}\,\vdash\Rightarrow$$

- The sequent formula that is introduced in the conclusion is the *principal* formula, and its components in the premise(s) are *side* formulas.

Metatheory

- Metatheorems about proof systems are useful in providing reasoning short-cuts.
- The deduction theorem for H and the normalization theorem for ND are examples.
- Prove that the Cut rule is admissible for the LK. (Difficult!)
- A bi-implication is a formula of the form $A \iff B$, and it is an equivalence when it is valid. Show that the following is a derived inference rule.

$$\frac{A \iff B}{C[p \mapsto A] \iff C[p \mapsto B]}$$

- State a similar rule for implication where

$$\frac{A \Rightarrow B}{C[p \mapsto A] \Rightarrow C[p \mapsto B]}$$

Normal Forms

- A formula where negation is applied only to propositional atoms is said to be in negation normal form (NNF).
- A *literal* l is either a propositional atom p or its negation $\neg p$.
- A *clause* is a multiary disjunction of a set of literals $\bigvee_{i=1}^{n} l_i$.
- A multiary disjunction of n formulas $A_1, \ldots, A_n$ is $\bigwedge_{i=1}^{n} A_i$.
- A formula that is a multiary conjunction of multiary disjunctions of literals is in conjunctive normal form (CNF).
- A formula that is a multiary disjunction of multiary conjunctions of literals is in disjunctive normal form (DNF).
- Show that every propositional formula built using $\neg$, $\vee$, and $\wedge$ is equivalent to one in NNF, CNF, and DNF. Define algorithms for conversion to these normal forms.

Soundness

- A proof system is *sound* if all provable formulas are valid, i.e., $\vdash A$ implies $\models A$.
- Demonstrate the soundness of the proof systems shown so far, i.e.,
  1. Hilbert system H
  2. Natural deduction ND
  3. Sequent Calculus LK

Completeness

- A proof system is *complete* if all valid formulas are provable, i.e., $\models A$ implies $\vdash A$.
- A countermodel M of $\Gamma \vdash \Delta$ is one where either $M \models A$ for all A in Γ, and $M \models \neg B$ for all $B \in \Delta$.
- In LK, any countermodel of some premise of a rule is also a countermodel for the conclusion.
- We can then show that a non-provable sequent $\Gamma \vdash \Delta$ has a countermodel.
- Each non-Cut rule has premises that are simpler than its conclusion.
- By applying the rules starting from $\Gamma \vdash \Delta$ to completion, you end up with a set of premise sequents $\{\Gamma_1 \vdash \Delta_2, \ldots, \Gamma_n \vdash \Delta_n\}$ that are *atomic*, i.e., that contain no connectives.
- If an atomic sequent $\Gamma_i \vdash \Delta_i$ is unprovable, then it has a countermodel, i.e., one in which each formula in Γ_i holds but no formula in Δ_i holds.
- Hence, $\Gamma \vdash \Delta$ has a countermodel.

Completeness, More Generally

- A set of formulas Γ is *consistent*, i.e., $Con(\Gamma)$ iff there is no formula A in Γ such that $\Gamma \vdash \neg A$ is provable.
- If Γ is consistent, then $\Gamma \cup \{A\}$ is consistent iff $\Gamma \vdash \neg A$ is not provable.
- If Γ is consistent, then at least one of $\Gamma \cup \{A\}$ or $\Gamma \cup \{\neg A\}$ must be consistent.
- A set of formulas Γ is *complete* if for each formula A, it contains A or $\neg A$.

Completeness

- Any consistent set of formulas Γ can be made complete as $\hat{\Gamma}$.
- Let A_i be the i'th formula in some enumeration of PL formulas. Define

$$\begin{aligned} \Gamma_0 &= \Gamma \\ \Gamma_{i+1} &= \Gamma_i \cup \{A_i\}, \text{ if } Con(\Gamma_i \cup \{A_i\}) \\ &= \Gamma_i \cup \{\neg A_i\}, \text{ otherwise.} \\ \hat{\Gamma} &= \Gamma_\omega = \bigcup_i \Gamma_i \end{aligned}$$

- Ex: Check that $\hat{\Gamma}$ yields an interpretation $\mathcal{M}_{\hat{\Gamma}}$ satisfying Γ.
- Is it enough to just enumerate as A_i, the propositional variables in Γ?
- If $\Gamma \vdash \Delta$ is unprovable, then $\Gamma \cup \overline{\Delta}$ is consistent, and has a model.

Compactness

- A logic is *compact* if any set of sentences Γ is satisfiable if all finite subsets of it are.
- Propositional logic is compact.
- If Γ is not satisfiable, then it is not consistent.
- The completeness argument can be adapted to a proof of compactness by replacing $Con(\Gamma)$ by finite satisfiability.
- Then the set $\hat{\Gamma}$ yields a complete finitely satisfiable set and hence it does not contain both A and $\neg A$ for any A.
- Since $\hat{\Gamma}$ is a superset of Γ, the latter is also satisfiable.
- Alternately, we can use completeness so show that if Γ is unsatisfiable, then there is a proof of $\Gamma \vdash \neg A$ for some $A \in \Gamma$.
- Since proofs are finite, there is some finite subset Γ' such that $\Gamma', A \vdash \neg A$ is provable, but $\Gamma' \cup \{A\}$ is satisfiable, contradicting soundness.

Interpolation

- Craig's interpolation property states that given two sets of formulas Γ_1 and Γ_2 in propositional variables Σ_1 and Σ_2, respectively, $\Gamma_1 \cup \Gamma_2$ is unsatisfiable iff there is a formula A in propositional variables $\Sigma_1 \cap \Sigma_2$ such that $\Gamma_1 \models A$ and Γ_2, A is unsatisfiable.
- An alternative statement of interpolation is that if $\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2$, then there is a formula C in the intersection of $\mathit{vars}(\Gamma_1 \cup \Delta_1) \cap \mathit{vars}(\Gamma_2 \cup \Delta_2))$ such that $\Gamma_1 \vdash C, \Delta_1$ and $\Gamma_2, C \vdash \Delta_2$.
- Show a way of annotating a sequent proofs so that each sequent has an interpolant.

Inference System

- An inference system $\mathcal{I}$ for a Σ-theory $\mathcal{T}$ is a $\Sigma[X]$-inference structure $\langle \Psi, \Lambda, \vdash \rangle$ that is
  1. **Conservative:** Whenever $\varphi \vdash_{\mathcal{I}} \varphi'$, $\Lambda(\varphi)$ and $\Lambda(\varphi')$ are $\mathcal{T}$-equisatisfiable.
  2. **Progressive:** The reduction relation $\vdash_{\mathcal{I}}$ should be well-founded, i.e., infinite sequences of the form $\langle \varphi_0 \vdash \varphi_1 \vdash \varphi_2 \vdash \ldots \rangle$ must not exist.
  3. **Canonizing:** A state is irreducible only if it is either $\bot$ or is $\mathcal{T}$-satisfiable.
- *For any class of $\Sigma[X]$-formulas Ψ, if there is a mapping ν from Ψ to Φ such that $\Lambda(\nu(A)) \iff A$, then a $\mathcal{T}$-inference system is a sound and complete inference procedure for $\mathcal{T}$-satisfiability in Ψ.*
- A computable function f such that $\kappa \vdash f(\kappa)$ whenever there is a κ' such that $\kappa \vdash \kappa'$, is a decision procedure for satisfiability.

Ordered Resolution

- We have already seen that any propositional formula can be written in CNF as a conjunction of clauses.
- Input K is a set of clauses.
- Atoms are ordered by $\succ$ which is lifted to literals so that $\neg p \succ p \succ \neg q \succ q$, if $p \succ q$.
- Literals appear in clauses in decreasing order without duplication.
- Tautologies, clauses containing both l and $\bar{l}$, are deleted from initial input.

Res	$\dfrac{K, l \vee \Gamma_1, \bar{l} \vee \Gamma_2}{K, l \vee \Gamma_1, \bar{l} \vee \Gamma_2, \Gamma_1 \vee \Gamma_2}$ $\Gamma_1 \vee \Gamma_2 \notin K$, $\Gamma_1 \vee \Gamma_2$ is not tautological
Contrad	$\dfrac{K, l, \bar{l}}{\bot}$

Ordered Resolution: Example

$$\dfrac{\dfrac{\dfrac{\dfrac{(K_0 =)\ \neg p \vee \neg q \vee r,\quad \neg p \vee q,\quad p \vee r,\quad \neg r}{(K_1 =)\ \neg q \vee r,\quad K_0}\ \textbf{Res}}{(K_2 =)\ q \vee r,\quad K_1}\ \textbf{Res}}{(K_3 =)\ r,\quad K_2}\ \textbf{Res}}{\bot}\ \textbf{Contrad}$$

Correctness

- **Progress:** Bounded number of clauses in the given literals. Each application of **Res** generates a new clause.
- **Conservation:** For any model M, if $M \models l \vee \Gamma_1$ and $M \models \bar{l} \vee \Gamma_2$, then $M \models \Gamma_1 \vee \Gamma_2$.
- **Canonicity:** Given an irreducible non-$\perp$ configuration K in the atoms $p_1, \ldots, p_n$ with $p_i \prec p_{i+1}$ for $1 \leq i \leq n$, build a series of partial interpretations M_i as follows:
  1. Let $M_0 = \emptyset$
  2. If p_{i+1} is the maximal literal in a clause $p_{i+1} \vee \Gamma \in K$ and $M_i \not\models \Gamma$, then let $M_{i+1} = M_i\{p_{i+1} \mapsto \top\}$.
  3. Otherwise, let $M_{i+1} = M_i\{p_{i+1} \mapsto \perp\}$.
- Each M_i satisfies all the clauses in K in the atoms $p_1, \ldots, p_i$.

CDCL Informally

- Goal: Does a given set of clauses K have a satisfying assignment?
- If M is a total assignment such that $M \models \Gamma$ for each $\Gamma \in K$, then $M \models K$.
- If M is a partial assignment at level h, then *propagation* extends M at level h with the *implied literals* l such that $l \vee \Gamma \in K \cup C$ and $M \models \neg\Gamma$.
- If M detects a conflict, i.e., a clause $\Gamma \in K \cup C$ such that $M \models \neg\Gamma$, then the conflict is *analyzed* to construct a conflict clause that allows the search to be continued from a prior level.
- If M cannot be extended at level h and no conflict is detected, then an unassigned literal l is *selected* and assigned at level $h + 1$ where the search is continued.

Conflict-Driven Clause Learning (CDCL) SAT

Name	Rule	Condition
Propagate	$\dfrac{h, \langle M \rangle, K, C}{h, \langle M, l[\Gamma] \rangle, K, C}$	$\Gamma \equiv l \vee \Gamma' \in K \cup C$ $M \models \neg\Gamma'$
Select	$\dfrac{h, \langle M \rangle, K, C}{h+1, \langle M; l[] \rangle, K, C}$	$M \not\models l$ $M \not\models \neg l$
Conflict	$\dfrac{0, \langle M \rangle, K, C}{\bot}$	$M \models \neg\Gamma$ for some $\Gamma \in K \cup C$
Backjump	$\dfrac{h+1, \langle M \rangle, K, C}{h', \langle M_{\leq h'}, l[\Gamma'] \rangle, K, C \cup \{\Gamma'\}}$	$M \models \neg\Gamma$ for some $\Gamma \in K \cup C$ $\langle h', \Gamma' \rangle$ $= analyze(\psi)(\Gamma)$ for $\psi = h, \langle M \rangle, K, C$

CDCL Example

- Let K be $\{p \vee q, \neg p \vee q, p \vee \neg q, s \vee \neg p \vee q, \neg s \vee p \vee \neg q, \neg p \vee r, \neg q \vee \neg r\}$.
-

step	h	M	K	C	Γ
select s	1	$; s$	K	$\emptyset$	_
select r	2	$; s; r$	K	$\emptyset$	_
propagate	2	$; s; r, \neg q[\neg q \vee \neg r]$	K	$\emptyset$	_
propagate	2	$; s; r, \neg q, p[p \vee q]$	K	$\emptyset$	_
conflict	2	$; s; r, \neg q, p$	K	$\emptyset$	$\neg p \vee q$

CDCL Example (contd.)

step	h	M	K	C	Γ
conflict	2	$; s; r, \neg q, p$	K	$\emptyset$	$\neg p \vee q$
backjump	0	$\emptyset$	K	q	_
propagate	0	$q[q]$	K	q	_
propagate	0	$q, p[p \vee \neg q]$	K	q	_
propagate	0	$q, p, r[\neg p \vee r]$	K	q	_
conflict	0	q, p, r	K	q	$\neg q \vee \neg r$

CDCL Correctness

- **Progress:** Each backjump step adds a new assignment at the level h' so that $\sum_{i=0}^{h} |M_i| * (N+1)^{(N-h)}$ increases toward the bound $(N+1)^{(N+1)}$ for $N = |vars(K)|$. In the example, $N = 4$, the backjump step goes from a value 1300 in base 5 to the value 10000 which is closer to the bound 40000.
- **Conservation:** In each transition from $\langle M, K, C \rangle$ to $\langle M', K', C' \rangle$ (or $\bot$), the clause sets $M_0 \cup K \cup C$ and $M_0 \cup K' \cup C'$ are equisatisfiable.
- **Canonicity:** In an irreducible non-$\bot$ state, M is total assignment and there is no conflict so for each clause Γ in $K \cup C$, $M \models \Gamma$.

Implementation Notes

- The input clauses can be preprocessed by resolution, e.g., to eliminate a variable, and subsumption to discard a clause when a subclause is already available.
- The *selection* heuristic can either pick
- Propagation uses *two-watched literals* per clause, so that a clause is visited only when a watched literal is falsified.
- Learned clauses can be *deleted* when they are unused in the partial assignment and not recently active in conflicts.
- Frequent *restarts* are good for learning useful short clauses in order to better direct the search.
- All level 0 inferences can be applied permanently.

CDCL with Proof

- We can build compact, easily checkable resolution certificates since each literal in M_0 and each conflict clause in C has an associated proof

Num.	Clause	Proof
0	$p \vee q$	
1	$\neg p \vee q$	
2	$p \vee \neg q$	
3	$\neg p \vee r$	
4	$\neg q \vee \neg r$	
5	q	$0, 1$
6	p	$5, 2$
7	r	$3, 6$
8	$\bot$	$4, 5, 7$

Interpolant

- The input clause set K is partitioned into K_1 and K_2.
- If K is unsatisfiable, there is a formula (interpolant) I such that $K_1 \Rightarrow I$ and $K_2 \wedge I \Rightarrow \bot$.
- Furthermore, $atoms(I) \subseteq atoms(K_1) \cap atoms(K_2)$.
- The interpolant for a proof can be constructed from the interpolant I_Γ for each clause Γ in the proof.
- Each clause Γ in the proof is partitioned into $\Gamma_1 \vee \Gamma_2$ with $atoms(\Gamma_2) \subseteq atoms(K_2)$ and $atoms(\Gamma_1) \cap atoms(K_2) = \emptyset$.
- The interpolant I_Γ has the property that $K_1 \vdash \neg\Gamma_1 \Rightarrow I_\Gamma$ and $K_2 \vdash I_\Gamma \Rightarrow \Gamma_2$.

Interpolants from Resolution

- For an input clauses $\kappa = \kappa_1 \vee \kappa_2$ in K_1, the interpolant $I_\kappa = \kappa_2$.
- For input clauses κ_2 in K_2, the interpolant is $\top$.
- When resolving κ', κ'' to get κ,
  - If resolvent p is in κ'_1 (i.e., $p \notin \mathit{atoms}(K_2)$), then $I_\kappa = I_{\kappa'} \vee I_{\kappa''}$ since $\neg(p \vee \kappa'_1) \Rightarrow I_{\kappa'}$ and $I_{\kappa'} \Rightarrow \kappa'_2$, and $\neg(\neg p \vee \kappa''_1) \Rightarrow I_{\kappa''}$ and $I_{\kappa''} \Rightarrow \kappa''_2$.
  -
  - If resolvent p is in κ'_2, then $I_\kappa = I_{\kappa'} \wedge I_{\kappa''}$ since $\neg(\kappa'_1 \vee \kappa''_1) \Rightarrow I_\kappa$ and $I_\kappa \Rightarrow (p \vee \kappa'_2) \wedge (\neg p \vee \kappa''_2) \iff \kappa'_2 \vee \kappa''_2$.

Interpolation Example

- Let $K_1 = \{a \vee e[e], \neg a \vee b[b], \neg a \vee c[c]\}$, and $K_2 = \{\neg b \vee \neg c \vee d[\top], \neg d[\top], \neg e[\top]\}$, with shared variables b, c, and e.
- The annotated proof is given by

Conc.	Interp.	Clauses
a	$[e]$	$a \vee \underline{e}[e], \underline{\neg e}[\top]$
b	$[e \vee b]$	$\underline{a}, \neg a \vee b$
c	$[e \vee c]$	$\neg a \vee c, a$
$\neg c \vee d$	$[e \vee b]$	$a \vee e, \neg a \vee b$
d	$[(e \vee b) \wedge (e \vee c)]$	$\neg c \vee d, c$
$\bot$	$[(e \vee b) \wedge (e \vee c)]$	$d, \neg d$

AIISAT

- To find *all* satisfying assignments for K, add a field B to CDCL to collect the *blocking clauses* corresponding to the current set of assignments.
- For input $\neg a \vee b, c$, the first assignment yields $M = c; a, b$. Add the negation $\neg c \vee \neg a \vee \neg b$ as a blocking clause to B and continue. (This could be reduced to $\neg c \vee \neg b$.)
- The next assignment $M' = c; a, \neg b$ generates a confict, so we add the conflict clause $\neg c \vee \neg a$ to C.
- Next, $c, \neg a; b$ is a satisfying assignment, so $\neg c \vee a \vee \neg b$ is added to B. Finally, $c, \neg a, \neg b$ is also satisfying, and hence $\neg c \vee a \vee b$ is added to B.
- There is a conflict at level 0, and $\neg \bigwedge B$ is the required DNF form of input K.
- **Exercise:** Show a method for compute the DNF of $\exists X.K$, where $X \subseteq \mathit{atoms}(K)$.

MaxSAT

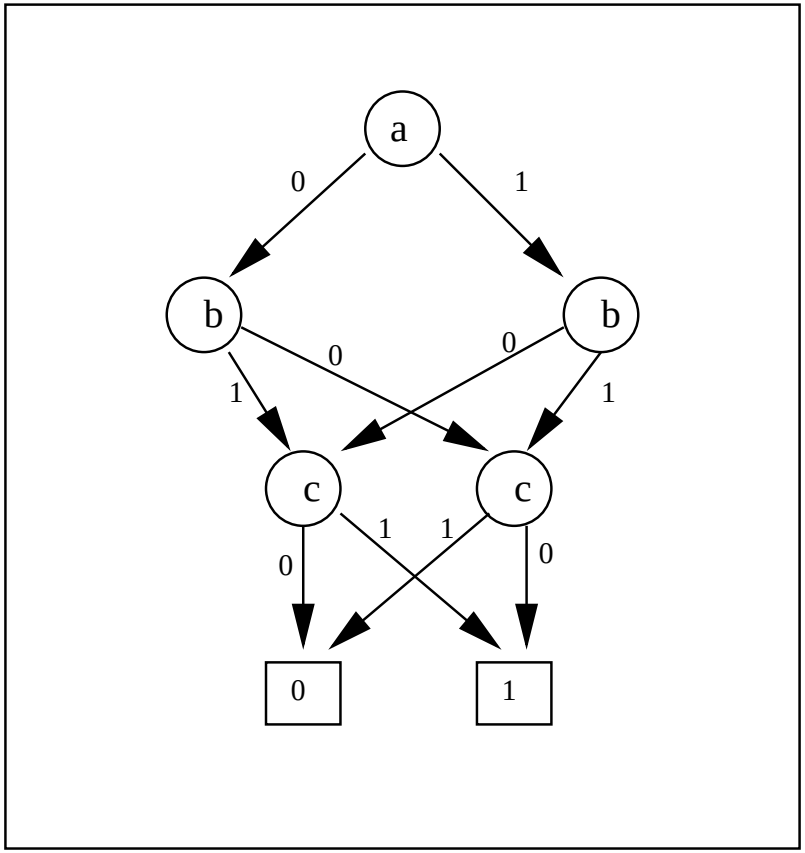
- With soft constraints, all constraints may not be satisfiable, but the goal is to satisfy as many as possible.
- Each constraint A_i can be augmented as $a_i \vee A_i$, for a fresh variable a_i.
- We can add constraints indicating that at most k of the a_i literals can be assigned $\top$.
- By shrinking k, we can determine the minimal value of k.
- Weighted MaxSAT can be solved similarly.
- More generally, pseudo-Boolean constraints $\Sigma_i w_i * a_i \leq k$ can be encoded.

ROBDD

- Boolean functions map $\{0,1\}^n$ to $\{0,1\}$.
- We have already seen how n-ary Boolean functions can be represented by propositional formulas of n variables.
- ROBDDs are a canonical representation of boolean functions as a decision diagram where
  1. Literals are uniformly ordered along every branch: $f(x_1,\ldots,x_n) = \mathtt{IF}(x_1, f(\top, x_2,\ldots,x_n), f(\bot, x_2,\ldots,x_n))$
  2. Common subterms are identified
  3. Redundant branches are removed: $\mathtt{IF}(x_i, A, A) = A$
- Efficient implementation of boolean operations: $f_1.f_2$, $f_1 + f_2$, $-f$, including quantification.
- Canonical form yields free equivalence checks (for convergence of fixed points).

ROBDD for Even Parity

ROBDD for even parity boolean function of a, b, c.



Construct an algorithm to compute $f_1 \odot f_2$, where $\odot$ is . or $+$.

Equality Logic (EL)

In the process of creeping toward first-order logic, we introduce a modest but interesting extension of propositional logic.
In addition to propositional atoms, we add a set of constants τ given by $c_0, c_1, \ldots$ and equalities $c = d$ for constants c and d.

$$\phi \quad := \quad P \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \tau_1 = \tau_2$$

The structure M now has a domain $|M|$ and maps propositional variables to $\{\top, \bot\}$ and constants to $|M|$.

$$M[\![c = d]\!] \quad = \quad \begin{cases} \top, & \text{if } M[\![c]\!] = M[\![d]\!] \\ \bot, & \text{otherwise} \end{cases}$$

Proof Rules for Equality Logic

Reflexivity	$\Gamma \vdash a = a, \Delta$
Symmetry	$\dfrac{\Gamma \vdash a = b, \Delta}{\Gamma \vdash b = a, \Delta}$
Transitivity	$\dfrac{\Gamma \vdash a = b, \Delta \qquad \Gamma \vdash b = c, \Delta}{\Gamma \vdash a = c, \Delta}$

- Show that the above proof rules (on top of propositional logic) are sound and complete.
- Show that Equality Logic is decidable.
- Adapt the above logic to reason about partial orders.

Term Equality Logic (TEL)

- One further extension is to add function symbols to form terms τ, so that constants are just 0-ary function symbols.

$$\begin{array}{rcl} \tau & := & f(\tau_1, \ldots, \tau_n), \text{ for } n \geq 0 \\ \phi & := & P \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \tau_1 = \tau_2 \end{array}$$

- For an n-ary function f, $M(f)$ maps $|M|^n$ to $|M|$.

$$\begin{array}{rcl} M[\![a = b]\!] & = & M[\![a]\!] = M[\![b]\!] \\ M[\![f(a_1, \ldots, a_n)]\!] & = & (M[\![f]\!])(M[\![a_1]\!], \ldots, M[\![a_n]\!]) \end{array}$$

- We need one additional proof rule.

Congruence	$\dfrac{\Gamma \vdash a_1 = b_1, \Delta \ldots \Gamma \vdash a_n = b_n, \Delta}{\Gamma \vdash f(a_1, \ldots, a_n) = f(b_1, \ldots, b_n), \Delta}$

Term Equality Proof Examples

Let $f^n(a)$ represent $\underbrace{f\,(\ldots f}_{n}(a)\ldots)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f^3(a)=f(a)\vdash f^3(a)=f(a)}\,Ax}{f^3(a)=f(a)\vdash f^4(a)=f^2(a)}\,C}{f^3(a)=f(a)\vdash f^5(a)=f^3(a)}\,C \qquad \dfrac{}{f^3(a)=f(a)\vdash f^3(a)=f(a)}\,Ax}{f^3(a)=f(a)\vdash f^5(a)=f(a)}\,T$$

Show soundness and completeness of the above system.

First-Order Logic

We can now complete the transition to first-order logic by adding

$$\begin{array}{rcl} \tau & := & X \\ & & \mid \; f(\tau_1, \ldots, \tau_n), \text{ for } n \geq 0 \\ \phi & := & \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \tau_1 = \tau_2 \\ & & \mid \forall x.\phi \mid \exists x.\phi \mid q(\tau_1, \ldots, \tau_n), \text{ for } n \geq 0 \end{array}$$

Terms contain variables, and formulas contain atomic and quantified formulas.

Semantics for Variables and Quantifiers

$M[\![q]\!]$ is a map from D^n to $\{\top, \bot\}$, where n is the arity of predicate q.

$$
\begin{aligned}
M[\![x]\!]\rho &= \rho(x) \\
M[\![q(a_1, \ldots, a_n)]\!]\rho &= M[\![q]\!](M[\![a_1]\!]\rho, \ldots, M[\![a_n]\!]\rho) \\
M[\![\forall x.A]\!]\rho &= \begin{cases} \top, & \text{if } M[\![A]\!]\rho[x := d] \text{ for all } d \in D \\ \bot, & \text{otherwise} \end{cases} \\
M[\![\exists x.A]\!]\rho &= \begin{cases} \top, & \text{if } M[\![A]\!]\rho[x := d] \text{ for some } d \in D \\ \bot, & \text{otherwise} \end{cases}
\end{aligned}
$$

Atomic formulas are either equalities or of the form $q(a_1, \ldots, a_n)$.

First-Order Logic

	Left	Right
$\forall$	$\dfrac{\Gamma, A[t/x] \vdash \Delta}{\Gamma, \forall x.A \vdash \Delta}$	$\dfrac{\Gamma \vdash A[c/x], \Delta}{\Gamma \vdash \forall x.A, \Delta}$
$\exists$	$\dfrac{\Gamma, A[c/x] \vdash \Delta}{\Gamma, \exists x.A \vdash \Delta}$	$\dfrac{\Gamma \vdash A[t/x], \Delta}{\Gamma \vdash \exists x.A, \Delta}$

- Constant c must be chosen to be new so that it does not appear in the conclusion sequent.
- Demonstrate the soundness of first-order logic.

Using First-Order Logic

- Prove $\exists x.(p(x) \Rightarrow \forall y.p(y))$.
- Give at least two satisfying interpretations for the statement $(\exists x.p(x)) \implies (\forall x.p(x))$.
- A sentence is a formula with no free variables. Find a sentence A such that both A and $\neg A$ are satisfiable.
- Write a formula asserting the unique existence of an x such that $p(x)$.
- Define operations for collecting the free variables $vars(A)$ in a given formula A, and substituting a term a for a free variable x in a formula A to get $A\{x \mapsto a\}$.
- Is $M[\![A\{x \mapsto a\}]\!]\rho = M[\![A]\!]\rho[x := M[\![a]\!]\rho]$? If not, show an example where it fails. Under what condition does the equality hold?
- Show that any quantified formula is equivalent to one in *prenex normal form*, i.e., where the only quantifiers appear at the head of the formula and the body is purely a propositional combination of atomic formulas.

More Exercises

- $\neg\forall x.A \iff \exists x.\neg A$
- $(\forall x.A \wedge B) \iff (\forall x.A) \wedge (\forall x.B)$
- $(\exists x.A \vee B) \iff (\exists x.A) \vee (\exists x.B)$
- $((\forall x.A) \vee (\forall x.B)) \Rightarrow (\forall x.A \vee B)$
- Can you write first-order formulas whose models
  1. Have exactly (at most, at least) three elements?
  2. Are infinite
  3. Are finite but unbounded
- Can you write a first-order formula asserting that
  1. A relation is transitively closed
  2. A relation is the transitive closure of another relation.

Equational Logic

- Equational logic (or Universal Algebra) is a heavily used fragment of first-order logic.
- This can be seen as a fragment of first-order logic where sequents are of the form $\Gamma \vdash A$, where the sequent formulas are all of the form $a = b$ or $\forall \overline{x}.a = b$, where $\overline{x}$ is a sequence of variables $x_1, \ldots, x_n$ and $\forall \overline{x}.a = b$ is $\forall x_1 \ldots \forall x_2.a = b$.
- Use equational logic to formalize (i.e., find axioms whose models are)
  1. Semigroups: A set G with an associative binary operator ..
  2. Monoids: A set M with associative binary operator . and unit 1.
  3. Groups: A monoid with an right-inverse operator x^{-1}
- Prove that every group element has a left inverse.

Completeness of First-Order Logic

- The quantifier rules for sequent calculus require copying.
- Proof branches can be extended without bound.
- Ex: Show that LK is sound: $\vdash A$ implies $\models A$.
- The Henkin closure $H(\Gamma)$ is the smallest extension of a set of sentences Γ that is Henkin-closed, i.e., contains $B \Rightarrow A(c_B)$ for every $B \in H(\Gamma)$ of the form $\exists x : A$. (c_B is a fresh constant.)
- Any consistent set of formulas Γ has a *consistent* Henkin closure $H(\Gamma)$.
- As before, any consistent, Henkin closed set of formulas Γ has a complete, Henkin-closed extension $\hat{\Gamma}$.
- Ex: Construct an interpretation $M_{\widehat{H(\Gamma)}}$ from $\widehat{H(\Gamma)}$ and show that it is a model for Γ.

Inference Systems for First-Order Theories

- Recall that inference systems are triples $\langle \Psi, \Lambda, \vdash \rangle$ where $\vdash$ is conservative, progressive, and canonizing.
- We have already seen inference systems for resolution and CDCL.
- A theory in a signature Σ is a set of Σ-structures (closed under isomorphism).
- Satisfiability procedures for many first-order theories and theory combinations can be presented as inference systems.
  1. Union-find for equality
  2. Basic superposition for equality/propositional reasoning
  3. Simplex-based linear arithmetic reasoning
  4. Satisfiability Modulo Theories

SMT Overview

- In SMT solving, the Boolean atoms represent constraints over individual variables ranging over integers, reals, datatypes, and arrays.
- The constraints can involve theory operations, equality, and inequality.
- The SAT solver has to interact with a theory constraint solver which propagates truth assignments and adds new clauses.
- The theory solver can detect conflicts involving theory reasoning, e.g.,
  1. $f(x) = f(y) \vee x \neq y$
  2. $f(x-2) \neq f(y+3) \vee x - y \leq 5 \vee y - z \leq -2 \vee z - x \leq -3$
  3. $x\ XOR\ y \neq 0b0000000 \vee select(store(A, x, v), y) = v$
- The theory solver must produce efficient explanations, incremental assertions, and efficient backtracking.

Example Constraint Solvers

- **Core theory:** Equalities between variables $x = y$, offset equalities $x = y + c$.
- **Term equality:** Congruence closure for uninterpreted function symbols
- **Difference constraints:** Incremental negative cycle detection for inequality constraints of the form $x - y \leq k$.
- **Linear arithmetic constraints:** Fourier's method, Simplex.

Theory Constraint Solver Interface

The satisfiability procedure uses a theory constraint solver oracle which maintains the theory state S with the interface operations:

1. $\mathit{assert}(l, S)$ adds literal l to the theory state S returning a new state S' or $\bot[\Delta]$
2. $\mathit{check}(S)$ checks if the conjunction of literals asserted to S is satisfiable, and returns either $\top$ or $\bot[\Delta]$.
3. $\mathit{retract}(S, l)$: Retracts, in reverse chronological order, the assertions up to and including l from state S.
4. $\mathit{model}(S)$: Builds a model for a state known to be satisfiable.

Satisfiability Modulo Theories

- SMT deals with formulas with theory atoms like $x = y$, $x \neq y$, $x - y \leq 3$, and $\mathit{select}(\mathit{store}(A, i, v), j) = w$.
- The CDCL search state is augmented with a *theory state* S in addition to the partial assignment.
- Total assignments are *checked* for theory satisfiability.
- When a literal is added to M by unit propagation, it is also *asserted* to S.
- When a literal is implied by S, it is *propagated* to M.
- When backjumping, the literals deleted from M are also *retracted* from S.

SMT example

The state extends CDCL with a find structure F and disquality set D.

Input is $y = z, \quad x = y \vee x = z, \quad x \neq y \vee x \neq z$

Step	M	F	D	C
Assert	$y = z$	$\{y \mapsto z\}$	$\emptyset$	$\emptyset$
Select	$y = z; x \neq y$	$\{y \mapsto z\}$	$\{x \neq y\}$	$\emptyset$
Prop	$\ldots, x \neq z$ $[x \neq z \vee y \neq z \vee x = y]$	$\{y \mapsto z\}$	$\{x \neq y\}$	$\emptyset$
Conflict	$\ldots$	$\{y \mapsto z\}$	$\{x \neq y\}$	$\emptyset$
Analyze	$\ldots$	$\{y \mapsto z\}$	$\{x \neq y\}$	$\{y \neq z \vee x = y\}$
Bkjump	$y = z, x = y$	$\{y \mapsto z\}$	$\emptyset$	$\ldots$
Assert	$y = z, x = y$	$\{x \mapsto y, y \mapsto z\}$	$\emptyset$	$\ldots$
Prop	$\ldots, x = z$ $[x = z \vee x \neq y \vee y \neq z]$	$\{x \mapsto y, y \mapsto z\}$	$\emptyset$	$\ldots$
Conflict				

Inference Systems for Theories

- Define an inference system for congruence closure for satisfiability for a set of equality and disequality assertions over terms.
- Define an inference system for difference logic constraints of the form $x - y \leq k$, where k is a numeric constant. Does it make a difference whether you are solving over integers, rationals, or reals?
- Define an inference system for linear inequality constraints $c_1 x_1 + \ldots c_n x_n \leq c$.
- A theory supports quantifier elimination if to any formula ϕ, there is an theory-equivalent quantifier-free formula $\hat{\phi}$.
- Which of the above theories support quantifier elimination?
- Show that first-order theory over infinite models in the empty signature supports quantifier elimination.
- Use this with first-order interpolation to combine two constraint solvers for disjoint theories. (Nelson-Oppen combination).

Skolemization

- Consider a formula of the form $\forall x.\exists y.q(x,y)$.
- It is equisatisfiable with the formula $\forall x.q(x,f(x))$ for a new function symbol f.
- If $M \models \forall x.\exists y.q(x,y)$, then for any $c \in |M|$, there is $d_c \in |M|$ such that $M[\![q(x,y)]\!]\{x \mapsto c, y \mapsto d_c\}$. let M' extend M so that $M(f)(c) = d_c$, for each $c \in |M|$: $M' \models \forall x.q(x,f(y))$.
- Conversely, if $M \models \forall x.q(x,f(y))$, then for every $c \in |M|$, $M[\![q(x,y)]\!]\{x \mapsto c, y \mapsto M(f)(c)\}$.
- Prove the general case that any prenex formula can be Skolemized by replacing each existentially quantified variable y by a term $f(\overline{x})$, where f is a distinct, new function symbol for each y, and $\overline{x}$ are the universally quantified variables *governing* y.

Herbrand's Theorem

- For any sentence A there is a quantifier-free sentence A_H (the Herbrand form of A) such that $\vdash A$ in LK iff $\vdash A_H$ in TEL_0.
- The Herbrand form is a *dual* of Skolemization where each universal quantifier is replaced by a term $f(\overline{y})$, where $\overline{y}$ is the set of governing existentially quantified variables.
- Then, $\exists x : (p(x) \Rightarrow \forall y : p(y))$ has the Herbrand form $\exists x.p(x) \Rightarrow p(f(x))$, and the two formulas are equi-valid.
- How do you prove the latter formula?

Herbrand's Theorem

- Herbrand terms are those built from function symbols in A_H (adding a constant, if needed).
- Show that if A_H is of the form $\exists \overline{x}.B$, then $\vdash A_H$ iff $\bigvee_{i=0}^{n} \sigma_i(B)$, for some Herbrand term substitutions $\sigma_1, \ldots, \sigma_n$.
- [Hint: In a cut-free sequent proof of a prenex formula, the quantifier rules can be made to appear below all the other rules. Such proofs must have a quantifier-free mid-sequent above which the proof is entirely equational/propositional.]
- Show that if a formula has a counter-model, then it has one built from Herbrand terms (with an added constant if there isn't one).

Unification

- A substitution is a map $\{x_1 \mapsto a_1, \ldots, x_n \mapsto a_n\}$ from a finite set of variables $\{x_1, \ldots, x_n\}$ to a set of terms.
- Define the operation $\sigma(a)$ of applying a substitution (such as the one above) to a term a to replace any free variables x_i in t with a_i.
- Define the operation of composing two substitutions $\sigma_1 \circ \sigma_2$ as $\{x_1 \mapsto \sigma_1(a_1), \ldots, x_n \mapsto \sigma_1(a_n)\}$, if σ_2 is of the form $\{x_1 \mapsto a_1, \ldots, x_n \mapsto a_n\}$.
- Given two terms $f(x, g(y, y))$ and $f(g(y, y), x)$ (possibly containing free variables), find a substitution σ such that $\sigma(a) \equiv \sigma(b)$.
- Such a σ is called a unifier.
- Not all terms have such unifiers, e.g., $f(g(x))$ and $f(x)$.
- A substitution σ_1 is more general than σ_2 if the latter can be obtained as $\sigma \circ \sigma_1$, for some σ.
- Define the operation of computing the most general unifier, if there is one, and reporting failure, otherwise.

Resolution

- To prove $(\exists y.\forall x.p(x,y)) \Rightarrow (\forall x.\exists y.p(x,y))$
- Negate: $(\exists y.\forall x.p(x,y)) \wedge (\exists x.\forall y.\neg p(x,y))$
- Prenexify: $\exists y_1.\forall x_1.\exists x_2.\forall y_2.p(x_1,y_1) \wedge \neg p(x_2,y_2)$
- Skolemize: $\forall x_1, y_2.p(x_1,c) \wedge \neg p(f(x_1),y_2)$
- Distribute and clausify: $\{p(x_1,c), \neg p(f(x_3),y_2)\}$
- Unify and resolve with unifier $\{x_1 \mapsto f(x_3), y_2 \mapsto c\}$
- Yields an empty clause
- Now try to show $(\forall x.\exists y.p(x,y)) \Rightarrow (\exists y.\forall x.p(x,y))$.

Dedekind-Peano Arithmetic

- The natural numbers consist of $0, s(0), s(s(0))$, etc.
- Clearly, $0 \neq s(x)$, for any x.
- Also, $s(x) = s(y) \Rightarrow x = y$, for any x and y.
- Next, we would like to say that this is all there is, i.e., every domain element is reachable from 0 through applications of s.
- This requires induction:
  $P(0) \wedge (\forall n.P(n) \Rightarrow P(n+1)) \Rightarrow (\forall n.P(n))$, for every property P.
- But there is no way to write this — there are uncountably many properties (subset of natural numbers) but only finitely many formulas.
- Induction is therefore given as a scheme, an infinite set of axioms, with the template

$$A\{x \mapsto 0\} \wedge (\forall x.A \Rightarrow A\{x \mapsto s(x)\}) \Rightarrow (\forall x.A).$$

- We still need to define $+$ and $\times$. How?
- How do you define the relation $x < y$?

Set Theory

Set theory has can also be axiomatized using axiom schemes, using a membership relation $\in$:

- Extensionality: $x = y \iff (\forall z.z \in x \iff z \in y)$
- The existence of the empty set $\forall x.\neg x \in \emptyset$
- Pairs: $\forall x, y.\exists z.\forall u \in z.u = x \vee u = y$ (Define the singleton set containing the empty set. Construct a representation for the ordered pair of two sets.)
- Union: How? (Define a representation for the finite ordinals using singleton, or using singleton and union.)
- Comprehension: $\{x \in y | A\}$, for any formula A, $y \notin \mathit{vars}(A)$. (Define the intersection and disjointness of two sets.)
- Infinity: There is a set containing all the finite ordinals.
- Power set: For any set, we have the set of all its subsets.
- Regularity: Every set has a minimal element (that is disjoint from it).
- Replacement: The image Y of a set X with respect to a functional $(\forall x \in X.\exists! y.A(x, y))$ rule $A(x, y)$, is a set.

Incompleteness

- Can all mathematical truths (valid sentences) be formally proved?
- *No.* There are valid statements about numbers that have no proof. (Gödel's first incompleteness theorem)
- Suppose Z is some formal theory claiming to be a sound and complete formalization of arithmetic, i.e., it proves all and only valid statements about numbers.
- Gödel showed that there is a valid but unprovable statement.

The First Incompleteness Theorem

- The expressions of Z can be represented as numbers as can the proofs.
- The statement "p is a proof of A" can then be represented by a formula $Pf(x, y)$ about numbers x and y.
- If p is represented by the number $\underline{p}$ and A by $\underline{A}$, then $Pf(\underline{p}, \underline{A})$ is provable iff p is a proof of A.
- Numbers such as $\underline{A}$ are representable as numerals in Z and these numerals can also be represented by numbers, $\underline{\underline{A}}$.
- Then $\exists x.Pf(x, y)$ says that the statement represented by y is *provable*. Call this $Pr(y)$.

The Undecidable Sentence

- Let $S(x)$ represent the numeric encoding of the operation such that for any number k, $S(k)$ is the encoding of the expression obtained by substituting the numeral for k for the variable 'x' in the expression represented by the number k.
- Then $\neg Pr(S(x))$ is represented by a number k, and the undecidable sentence U is $\neg Pr(S(k))$.
- $\underline{U}$ is $S(k)$, i.e., the sentence obtained by substituting the numeral for k for 'x' in $\neg Pr(S(x))$ which is represented by k.
- Since U is $\neg Pr(\underline{U})$, we have a situation where either
  1. U, i.e., $\neg Pr(\underline{U})$, is provable, but from the numbering of the proof of U, we can also prove $Pr(\underline{U})$.
  2. $\neg U$, i.e., $Pr(\underline{U})$ is provable, but clearly none of $Pf(0, \underline{U})$ $Pf(1, \underline{U}), \ldots,$ is provable (since otherwise U would be provable), an ω-inconsistency, or
  3. Neither U nor $\neg U$ is provable: an incompleteness.

Second Incompleteness Theorem

- The negation of the sentence U is Σ_1, and Z can verify Σ_1-completeness (*every valid Σ_1-sentence is provable*).
- Then
$$\vdash Pr(\underline{U}) \Rightarrow Pr(\underline{Pr(\underline{U})}).$$
- But this says $\vdash Pr(\underline{U}) \Rightarrow Pr(\underline{\neg U})$.
- Therefore $\vdash Con(Z) \Rightarrow \neg Pr(\underline{U})$.
- Hence $\neg \vdash Con(Z)$, by the first incompleteness theorem.
- **Exercise:** The theory Z is consistent if $A \wedge \neg A$ is not provable for any A. Show that ω-consistency is stronger than consistency. Show that the consistency of Z is adequate for proving the first incompleteness theorem.

Higher-Order Logic

- Thus far, variables ranged over ordinary datatypes such as numbers, and the functions and predicates were fixed (constants).
- Second-order logic allows free and bound variables to range over the functions and predicates of first-order logic.
- In n'th-order logic, the arguments (and results) of functions and predicates are the functions and predicates of m'th-order logic for $m < n$.
- This kind of strong typing is required for consistency, otherwise, we could define $R(x) = \neg x(x)$, and derive $R(R) = \neg R(R)$.
- Higher-order logic, which includes n'th-order logic for any $n > 0$, can express a number of interesting concepts and datatypes that are not expressible within first-order logic: transitive closure, fixpoints, finiteness, etc.

Types in Higher-Order Logic

- Base types: e.g., `bool`, `nat`, `real`
- Tuple types: $[T_1, \ldots, T_n]$ for types $T_1, \ldots, T_n$.
- Tuple terms: $(a_1, \ldots, a_n)$
- Projections: $\pi_i(a)$
- Function types: $[T_1 \rightarrow T_2]$ for domain type T_1 and range type T_2.
- Lambda abstraction: $\lambda(x : T_1) : a$
- Function application: $f\ a$.

Semantics of Higher Order Types

$$
\begin{aligned}
[\![\texttt{bool}]\!] &= \{0,1\} \\
[\![\texttt{real}]\!] &= \mathbf{R} \\
[\![[T_1,\ldots,T_n]]\!] &= [\![T_1]\!] \times \ldots \times [\![T_n]\!] \\
[\![[T_1 \rightarrow T_2]]\!] &= [\![T_2]\!]^{[\![T_1]\!]}
\end{aligned}
$$

Higher-Order Proof Rules

β-reduction	$\dfrac{}{\Gamma \vdash (\lambda(x:T):a)(b) = a[b/x], \Delta}$
Extensionality	$\dfrac{\Gamma \vdash (\forall(x:T):f(x) = g(x)), \Delta}{\Gamma \vdash f = g, \Delta}$
Projection	$\dfrac{}{\Gamma \vdash \pi_i(a_1, \ldots, a_n) = a_i, \Delta}$
Tuple Ext.	$\dfrac{\Gamma \vdash \pi_1(a) = \pi_1(b), \Delta, \ldots, \Gamma \vdash \pi_n(a) = \pi_i(b), \Delta}{\Gamma \vdash a = b, \Delta}$

Conclusions: Speak Logic!

- Logic is a powerful tool for
  1. Formalizing concepts
  2. Defining abstractions
  3. Proving validities
  4. Solving constraints
  5. Reasoning by calculation
  6. Mechanized inference
- The power of logic is when it is used as an aid to effective reasoning.
- > *Logic can become enormously difficult, and it would undoubtedly be well to produce more assurance in its use. . . . We may some day click off arguments on a machine with the same assurance that we now enter sales on a cash register.*
  >
  > *Vannevar Bush, As We May Think*
- The machinery of logic has made it possible to solve large and complex problems; formal verification is now a practical technology.

References

- Barwise, *Handbook of Mathematical Logic*
- Johnstone, *Notes on logic and set theory*
- Ebbinghaus, Flum, and Thomas, *Mathematical Logic*
- Kees Doets, *Basic Model Theory*
- Huth and Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*
- Girard, Lafont, and Taylor, *Proofs and Types*
- Shankar, *Automated Reasoning for Verification*, ACM Computing Surveys, 2009