Speaking Logic

N. Shankar

Computer Science Laboratory
SRI International
Menlo Park, CA

May 19, 2018

Proofs and Things

Perhaps I can best describe my experience of doing mathematics in terms of a journey through a dark unexplored mansion. You enter the first room of the mansion and it's completely dark. You stumble around bumping into the furniture, but gradually you learn where each piece of furniture is. Finally, after six months or so, you find the light switch, you turn it on, and suddenly it's all illuminated. You can see exactly where you were. Then you move into the next room and spend another six months in the dark. So each of these breakthroughs, while sometimes they're momentary, sometimes over a period of a day or two, they are the culmination of and couldn't exist without the many months of stumbling around in the dark that precede them. Andrew Wiles[1]

[1] http://www.pbs.org/wgbh/nova/physics/andrew-wiles-fermat.html

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.
- Properties of abstractions are captured by precisely stated laws through *formalization* using axioms, definitions, theorems, and proofs.
- Logic is the *medium* for expressing these abstract laws and the *method* for deriving consequences of these laws using sound reasoning principles.
- Computing is *abstraction engineering*.
- Logic is the calculus of computing.

- The world is increasingly an interplay of abstractions.
- Caches, files, IP addresses, avatars, friends, likes, hyperlinks, packets, network protocols, and cyber-physical systems are all examples of abstractions in daily use.
- Such abstract entities and the relationships can be expresses clearly and precisely in logic.
- In computing, and elsewhere, we are increasingly dependent on formalization as a way of managing the abstract universe.

Where Logic has Been Effective

Logic has been *unreasonably* effective in computing, with an impact that spans

- Theoretical computer science: Algorithms, Complexity, Descriptive Complexity
- Hardware design and verification: Logic design, minimization, synthesis, model checking
- Software verification: Specification languages, Assertional verification, Verification tools
- Computer security: Information flow, Cryptographic protocols
- Programming languages: Logic/functional programming, Type systems, Semantics
- Artificial intelligence: Knowledge representation, Planning
- Databases: Data models, Query languages
- Systems biology: Process models

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.
- In computing, the challenge is to solve large and complex problems through abstraction and decomposition.
- Formal, logical reasoning is needed to achieve scale and correctness.
- We examine how logic is used to formulate problems, find solutions, and build proofs.
- We also examine useful metalogical properties of logics, as well as algorithmic methods for effective inference.

Course Schedule

- The course is spread over Four lectures:
  - **Lecture 1:** Proofs and Things
  - **Lecture 2:** Propositional Logics
  - **Lecture 3:** First-Order and Higher-Order Logic
  - **Lecture 4:** Advanced topics
- 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 in the school.
- To get the most out of the course, please do the exercises.

A Small Puzzle [Wason]

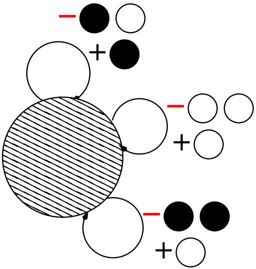
- Given four cards laid out on a table as: $\boxed{\text{D}}$, $\boxed{3}$, $\boxed{\text{F}}$, $\boxed{7}$, where each card has a letter on one side and a number on the other.
- Which cards should you flip over to determine if every card with a $\boxed{\text{D}}$ on one side has a $\boxed{7}$ on the 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, one at each gate: one always tells the truth, and the other always lies, but you can't tell which is which.
- You are allowed to ask one of the guards one question with a yes/no answer.
- What question should you ask in order to find out which gate leads to the castle?

When is Cheryl's Birthday?

- Albert and Bernard have just become friends with Cheryl, and they want to know her date of birth. Cheryl gives them 10 possible dates:
-
May 15	May 16	May 19
June 17	June 18	
July 14	July 16	
August 14	August 15	August 17

- Cheryl then tells Albert and Bernard separately the month and the day of her birthday, respectively.
- **Albert:** I don't know when Cheryl's birthday is, but I know that Bernhard does not know too.
  Bernard: At first I didn't know Cheryl's birthday, but now I do.
  Albert: Then I also know Cheryl's birthday.
- When is Cheryl's birthday?

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.*

- Write a program to determine the numbers m and n.

Pigeonhole Principle & Cantor's Theorem

- Why can't you park $n+1$ cars in n parking spaces, if each car needs its own space?
- Let $m..n$ represent the subrange of integers from m to, but not including, n.
- An injection from set A to set B is a map f such that $f(x) = f(y)$ implies $x = y$, for any x, y in A.
- The Pigeonole principle can be restated as asserting that there is no injection from $0..n+1$ to $0..n$. Prove it.
- Let $\mathbb{N}$ be the set of natural numbers $0, 1, 2, \ldots$, and let $\wp(\mathbb{N})$ be the set of subsets of $\mathbb{N}$.
- Show that there is no injection from $\wp(\mathbb{N})$ to $\mathbb{N}$.

Hard Sudoku [Wikipedia/Algorithmics_of_Sudoku]

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

Gilbreath's Card Trick

- Start with a deck consisting of a stack of quartets, where the cards in each quartet appear in suit order $\spadesuit$, $\heartsuit$, $\clubsuit$, $\diamondsuit$:

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

- 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

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

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

A Sorting Card Trick

- Arrange 25 cards from a deck of cards in a 5x5 grid.
- First, sort each of the rows individually.
- Then, sort each of the columns individually.
- Now both the rows and columns are sorted. How come?

Length of the Longest Increasing Subsequence

- You have a sequence of numbers, e.g.,
  9, 7, 10, 9, 5, 4, 10.
- The task is to find the length of the longest increasing subsequence.
- Here the longest subsequence is 7, 9, 10, and its length is 3.
- Patience solitaire is a card game where cards are placed, one by one, into a sequence of columns.
- Each card is placed at the bottom of the leftmost column where it is no bigger than the current bottom card in the column.
- If there is no such column, we start a new column at the right.
- Show that the number of columns left at the end yields the length of the longest increasing subsequence.

Computing Majority

- An election has five candidates: Alice, Bob, Cathy, Don, and Ella.
- The votes have come in as:
  E, D, C, B, C, C, A, C, E, C, A, C, C.
- You are told that some candidate has won the majority (over half) of the votes.
- You successively remove pairs of dissimilar votes, until there are no more such pairs.
- That is, the remaining votes, if any, are all for the same candidate.
- Show that this candidate has the majority.

Maximum Segment Sum

- Given an array $a[0..N-1]$ of integers, a segment sum over the segment $a[l..h]$ is $\Sigma_{j=l}^{h} a[j]$ for $0 \leq l, h < N$.
- The maximum segment sum is $max_{l,h} \Sigma_{j=l}^{h} a[j]$.
- Since segments can be empty, the minimum segment sum is 0.
- For example, if the array elements are $a[0] = -3, a[1] = 4, a[2] = -2, a[3] = 6, a[4] = -5$, then the maximum segment sum is 8, which is the sum over $a[1..3]$.
- Write and verify an algorithm for computing the maximum segment sum of a given array.

Proofs, informally

- For $n \in \mathbb{N}$, prove that $\Sigma_{i=0}^{n} = n(n+1)/2$. Why is the right-hand side always a natural number.
- With $n, k \in \mathbb{N}$ with $n \geq k > 0$, $\left(\begin{array}{c} n \\ k \end{array}\right) = \frac{n!}{(n-k)!k!}$, show that $\left(\begin{array}{c} n \\ k \end{array}\right)$ is a natural number.
- Define $\mathbb{N}$ as the smallest set containing 0 and closed under the successor operation S, where $S(x) \neq x$.
- Define addition recursively as

$$\begin{array}{rcl} 0 + y & = & y \\ S(x) + y & = & x + S(y) \end{array}$$

Prove that $+$ is associative.

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.

Naïve Set Theory

- We will be using sets informally when talking about logic.
- Sets have members $x \in X$ (x is an element of X), and can be related through equality $X = Y$ (X and Y have the same elements), and subset $X \subset Y$ or $X \subseteq Y$ (every element of X is an element of Y).
- Sets include the emptyset $\emptyset$, the singleton set $\{a\}$ containing just a as an element, the two-element set $\mathbf{2} = \{0, 1\}$, the set $\mathbb{N}$ of natural numbers $\{0, 1, 2, \ldots\}$.
- Other examples include the set of integers, odd integers, even integers, prime numbers, rational numbers, algebraic numbers, real numbers, etc.
- The set of elements satisfying a property P is represented as $\{x|P(x)\}$, e.g., $\{i|0 \leq i \leq 5\}$.
- Let F be a map, e.g., $x \mapsto x^2$, then $F[X]$ represents the image of X with respect to X.
- The $\{F(x)|P(x)\}$ contains all and only the elements $F(a)$ for each element a satisfying $P(a)$, e.g., $\{x^2|0 \leq x \leq 5\}$.

Naïve Set Theory

- The set $\{a, b\}$ represents the set that is the pair of elements a and b, which can themselves be sets.
- Ordered pairing $\langle x, y\rangle$ can be represented as $\{\{x, y\}, \{y\}\}$.
- The union $\bigcup X$ is the set $\{x|x \in y, y \in X\}$. $X \cup Y$ is just $\bigcup\{X, Y\}$.
- The intersection $\bigcap X$ is the set $\{x|x \in y, \text{for each } y \in X\}$. $X \cap Y$ is just $\bigcap\{X, Y\}$.
- Define projections π_1 and π_2 such that $\pi_1(\langle x, y\rangle) = x$ and $\pi_2(\langle x, y\rangle) = y$.
- The relative complement $X - Y$ of two sets is the set $\{x|x \in X, x \notin Y\}$.
- The Cartesian product $X \times Y$ is the set $\{\{x, y\}|x \in X, y \in Y\}$ of ordered pairs $\langle x, y\rangle$ for $x \in X$ and $y \in Y$.
- Two sets are equal if they have exactly the same elements. Prove $(X \cup Y) \cup Z = X \cup (Y \cup Z)$, $X \cup Y = Y \cup X$, $X \cup X = X$, and similarly for intersection.

Naïve Set Theory

- The set of integers $\mathbb{Z}$ is $\{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$.
- A map F between X and Y is injective if whenever $F(x) = F(x')$ for x, x' in X, we have $x = x'$.
- A map F between X and Y is surjective if for each $y \in Y$, there is an $x \in X$, such that $F(x) = y$.
- A map F between X and Y is bijective if F is both injective and surjective.
- The *graph* of a map F between X and Y can be represented as a subset of $X \times Y$ as $\{\langle x, y\rangle | F(x) = y\}$.
- A subset G of $X \times Y$ is a graph if for any $x \in X$, there is exactly one y such that $\langle x, y\rangle \in G$.
- Define the operation of applying a graph G to an argument x.
- Define Y^X represent the set of graphs with X as domain and Y as range.
- Show that there is a bijection between $X \times (Y \times Z)$ and $(X \times Y) \times Z$ and 2^X and the power set of X: $\{Y | Y \subseteq X\}$.

Naïve Set Theory Exercises

- Show that each integer can be represented (non-uniquely) by a pair of natural numbers.
- Define an equivalence relation $\simeq$ on this representation of integers.
- Show that this representation of the integers is order-isomorphic to the set $\mathbb{Z}$ of integers.
- For any set X, define the set of infinite sequences over X.
- Define the set of Cauchy sequences of rational numbers, where a sequence σ is Cauchy if for any rational number $\epsilon > 0$, there is some i such that for every $m, n > i$, $|\sigma_m - \sigma_n| < \epsilon$.
- Exhibit a Cauchy sequence that converges to $\sqrt{2}$.

Naïve Set Theory Exercises

- One set is *equinumerous* with another if there is a bijection between them.
- Is the set $\mathbb{Z}$ equinumerous with $\mathbb{N}$.
- Can a set X be equinumerous with its powerset $\mathbf{2}^X$?
- Is the set of ordered pairs of natural numbers $\mathbb{N} \times \mathbb{N}$ equinumerous with $\mathbb{N}$.
- Is the set of rational numbers $\mathbb{Q}$ equinumerous with $\mathbb{N}$?
- Is the set of real numbers in the interval $[0, 1]$ equinumerous with $\mathbb{N}$?

Orderings

- A binary relation $<$ on a set U (a *poset*) is a *partial ordering* if it is
  - Reflexive: $x < x$ for all $x \in U$
  - Transitive: $x < z$ if $x < y$ and $y < x$, for all $x, yz \in U$
  - Anti-Symmetric: $x = y$ if $x < y$ and $y < x$.
- A partial ordering is *total* (or *linear*) if for all $x, y \in U$: $x < y$ or $y < x$.
- For a subset X of U, element $x \in X$ is
  - Minimal, if for $y \in X$, $y = x$ or $y \not< x$.
  - Least, if $x < y$ for $y \in X$.
  - Maximal, if for $y \in X$, $y = x$ or $x \not< y$.
  - Greatest, if $y < x$ for $y \in X$.
- A *filter* is a nonempty subset of U that is upward closed and and contains x whenever $x < y$ and $x < z$ for y, z in U.
- An *ultrafilter* on U is a *proper filter* (i.e., not U itself) that is maximal. Formally define the concepts of filter and ultrafilter.

Orderings

- A *strict partial ordering* is irreflexive, transitive, and anti-symmetric.
- An *antichain* is a subset X of U such that $x \not< y$, for $x, y \in X$.
- A partial order is *well-founded* if every nonempty subset X of U has a minimal element.
- A linear order is *well-ordered* if every nonempty subset X of U has a least element.
- Every well-ordering is well-founded.

Ordinal Numbers

- The ordinal numbers can be constructed as: 0 is an ordinal number, and the next ordinal number is the set of all preceding ordinal numbers.
- The ordinals are well-ordered, and any well-ordered set is order-isomorphic, i.e., has an order-preserving bijection, to some ordinal.
- Is the set $\mathbb{N}$ under the usual $<$ ordering of natural numbers well ordered?
- Is the set $\mathbb{Z}$ under the usual $<$ ordering of integers well ordered? Is there a well-ordering for $\mathbb{Z}$?
- Is the set $\mathbb{Q}$ under the usual $<$ ordering of rationals well ordered? Is there a well-ordering for $\mathbb{Q}$?

Ordinal Numbers

- Let ω represent the ordinal number for $\mathbb{N}$ with $0 < 1 < 2 \ldots$.
- $\mathbb{N}$ can be ordered so that $i < j$ for any even number i and odd number j to get $0 < 2 < 4 \ldots 1 < 3 < 5 \ldots$.
- This has the order type $\omega + \omega$.
- A lexicographic ordering on $\mathbb{N} \times \mathbb{N}$ has $\langle x, y\rangle < \langle x', y'\rangle$ if $x < x'$, or $x = x'$ and $y < y'$, e.g., $\langle 5, 3\rangle < \langle 5, 4\rangle$.
- *What is the ordinal corresponding to the above lexicographic ordering?*
- *Can you define an ordering on $\mathbb{N}$ that is order isomorphic to the lexicographic ordering on $\mathbb{N} \times \mathbb{N}$?*

Paradoxes

- The Liar paradox has Epimenides, a Cretan, asserting that *All Cretans are liars.*? *Could Epimenides be telling the truth? Could he be lying?*
- What if Epimenides said "Everything I say is false"?
- Berry's paradox: What is "the smallest natural number not definable in fewer than twelve words"?
- Richards paradox: Let $p_0, p_1, \ldots,$ be an enumeration of the properties of natural numbers. We say that i is *Richardian* if $\neg p_i(i)$. Is there a property p_r in the enumeration that captures the property of being Richardian?
- *Let R be the set $\{x | x \notin x\}$. Is $R \in R$?*
- *Is there a universal set V given by $\{x | x = x\}$?*
- *The set of ordinals* **ON** *is itself well-ordered. Is* **ON** $\in$ **ON***?*

Logic Basics

- Logic studies the *trinity* between *language*, *interpretation*, and *proof*.
- *Language*: What are you allowed to say?
- *Interpretation*: What is the intended meaning?
  - Meaning is usually *compositional*: Follows the syntax
  - Some symbols have fixed meaning: connectives, equality, quantifiers
  - Other symbols are allowed to vary variables, functions, and predicates
  - *Assertions* either hold or fail to hold in a given interpretation
  - A *valid* assertion holds in every interpretation
- *Proofs* are used to demonstrate validity

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, $p \wedge \neg p$, $p \vee \neg p$, $(p \wedge \neg q) \vee \neg p$.

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?
How many distinct truth tables are there in n distinct propositional variables?

Problems

- 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 $B[p \mapsto A]$?
- What is the truth-table meaning of $B[p \mapsto A]$ in terms of the meaning of B and A?

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 $\vee$. Conversely, define $\neg$ and $\vee$ using $NAND$.

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 $\phi \models \psi$.

Satisfiable, Unsatisfiable, or Valid?

- Classify these formulas as satisfiable, unsatisfiable, or valid?
  - $p \vee \neg p$
  - $p \wedge \neg p$
  - $\neg p \Rightarrow p$
  - $((p \Rightarrow q) \Rightarrow p) \Rightarrow p$
- Make up some examples of formulas that are satisfiable (unsatisfiable, valid)?
- If A and B are satisfiable, is $A \wedge B$ satisfiable? What about $A \vee B$.
- Can A and $\neg A$ both be satisfiable (unsatisfiable, valid)?

Some Valid Laws

- A bi-implication $A \iff B$ is valid (and hence A and B are *equivalent*) iff every model of A is a model of B and vice-versa.
- Check that following formulas are valid for any assignment to A and B?
    1. $\neg(A \wedge B) \iff \neg A \vee \neg B$
    2. $\neg(A \vee B) \iff \neg A \wedge \neg B$
    3. $((A \vee B) \vee C) \iff A \vee (B \vee C)$
    4. $(A \Rightarrow B) \iff (\neg A \vee B)$
    5. $(\neg A \Rightarrow \neg B) \iff (B \Rightarrow A)$
    6. $\neg\neg A \iff A$
    7. $A \Rightarrow B \iff \neg A \vee B$
    8. $\neg(A \wedge B) \iff \neg A \vee \neg B$
    9. $\neg(A \vee B) \iff \neg A \wedge \neg B$
    10. $\neg A \Rightarrow B \iff \neg B \Rightarrow A$

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.
- 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.

Using Propositional Logic

- 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.
- Describe an N-bit ripple carry adder with a carry-in and carry-out bits as a formula.

Cook's Theorem

- A Turing machine consists of a finite automaton reading (and writing) symbols from a finite set Σ (including a blank symbol '␣') from a tape $\ldots, T(-1), T(0), T(1), \ldots$.
- Initially, the tape is blank except at the input $T(0), \ldots, T(n-1)$.
- The finite automaton has a finite set of states Q, a subset F of which are accepting states.
- In each step, if the automaton is at a non-accepting state, the machine reads the symbol at the current position of the head, and nondeterministically executes a step consisting of
  1. A new symbol to write at the head position
  2. A move (left or right) of the head from the current position
  3. A next automaton state

Cook's Theorem

- For some bound N on the number of machine steps, show that the it is possible to represent the following using a polynomial number (in n) of Boolean variables
  1. The k'th symbol is on the i'th cell in the j'th state of the computation.
  2. The head is at the i'th cell in the j'th state of the computation.
  3. The automaton is in the m'th state in the j'th state of the computation.
- Show that SAT is solvable in polynomial time (in the size n of the input) by a nondeterministic Turing machine.
- Show that for any nondeterministic Turing machine and polynomial bound $p(n)$ for input of size n, one can (in polynomial time) construct a propositional formula which is satisfiable iff there is the Turing machine accepts the input in at most $p(n)$.

Reductions to SAT

- Encode the following problems as SAT problems
  1. 3-colorability of an undirected graph.
  2. The k-colorability for a given k.
  3. The existence of a Hamiltonian path in a graph, one that visits each vertex exactly once.
  4. The existence of a k-clique in a graph: a set of k vertices that are pairwise connected by edges.
- What is the size of your encoding?
- A problem is *NP-hard* if there is a polynomial-time (many-to-one, Turing, truth-table) reduction from SAT (or another NP-hard) problem to it.

Proof Systems

- There are three basic styles of proof systems.
- These are distinguished by their basic judgement.
  1. Hilbert systems: $\vdash A$ means the formula A is provable.
  2. Natural deduction: $\Gamma \vdash A$ means the formula A is provable from a set of assumption formulas Γ.
  3. Sequent Calculus: $\Gamma \vdash \Delta$ means the consequence of $\bigvee \Delta$ from $\bigwedge \Gamma$ is derivable.

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
  - $\overline{\vdash A \Rightarrow A}$
  - $\overline{\vdash A \Rightarrow (B \Rightarrow A)}$
  - $\overline{\vdash (A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))}$
- 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?

Hilbert System (H)

- Are any of the axioms redundant? [Hint: See if you can prove the first axiom from the other two.]
- Can you prove
  1. $A \Rightarrow (B \Rightarrow B)$
  2. $(A \Rightarrow B) \Rightarrow ((B \Rightarrow C) \Rightarrow (A \Rightarrow C))$.
- Write Hilbert-style axioms for $\neg$, $\wedge$ and $\vee$.

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 $\Gamma \cup \{A\}$. [Hint: Use induction on proofs.]
- 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 conclusion C is provable if the premises $P_1, \ldots, P_n$ are provable.
- Every derived rule is admissible, but what is an example of an admissible rule that is not a derived one?

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?

Minimal Logic

- Add a propositional constant $\perp$ to the implicational logic above.
- Define negation $\neg A$ as $A \Rightarrow \perp$.
- Can you prove
  1. $\neg A \Rightarrow (A \Rightarrow B)$
  2. $\neg A \Rightarrow (A \Rightarrow \neg B)$
  3. $A \Rightarrow \neg\neg A$
  4. $\neg\neg A \Rightarrow A$
  5. $\perp \Rightarrow A$
- If you take Formula 1 as an axiom, can you prove the others?
- Conjunction $A \wedge B$ can be encoded as $(A \Rightarrow (B \Rightarrow \perp)) \Rightarrow \perp$.
- Show that $A \Rightarrow (B \Rightarrow (A \wedge B))$, $(A \wedge B) \Rightarrow A$, and $(A \wedge B) \Rightarrow B$.

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

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

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

- 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 for Formulas

- A formula where negation is applied only to propositional atoms is said to be in negation normal form (NNF).
- For example, $\neg(p \vee \neg q)$ can be represented as $\neg p \wedge q$.
- Show that every propositional formula built using $\neg$, $\vee$, and $\wedge$ is equivalent to one in 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 $l_1 \vee \ldots \vee l_n$.
- A multiary conjunction of n formulas $A_1, \ldots, A_n$ is $\bigwedge_{i=1}^{n} A_i$.

Conjunctive and Disjunctive Normal Forms

- A formula that is a multiary conjunction of multiary disjunctions of literals is in conjunctive normal form (CNF).
- CNF Example:
$$\begin{array}{rl} & (\neg p \vee q \vee \neg r) \\ \wedge & (p \vee r) \\ \wedge & (\neg p \vee \neg q \vee r) \end{array}$$
- Define an algorithm for converting any propositional formula to CNF.
- A formula is in k-CNF if it uses at most k literals per clause. Define an algorithm for converting any formula to 3-CNF.
- A formula that is a multiary disjunction of multiary conjunctions of literals is in disjunctive normal form (DNF).
- Define an algorithm for converting any formula to DNF.

Soundness

- A proof system is *sound* if all provable formulas are valid, i.e., $\vdash A$ implies $\models A$, i.e., $M \models A$ for all M.
- To prove soundness, show that for any inference rule of the form
$$\frac{\vdash P_1, \ldots, \vdash P_n}{\vdash C},$$
any model of all of the premises is also a model of the conclusion.
- Since the axioms are valid, and each step preserves validity, we have that the conclusion of a proof is also valid.
- 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. *What is the countermodel for $p \vee q \vdash p \wedge q$?*.
- 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 $\{\Gamma_1 \vdash \Delta_1, \ldots, \Gamma_n \vdash \Delta_n\}$ 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.

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

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

- Ex: Check that $\hat{\Gamma}$ yields an interpretation $\mathcal{M}_{\hat{\Gamma}}$ satisfying 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 iff all finite subsets of it are, i.e., if it is *finitely satisfiable*.
- Propositional logic is compact — hard direction is showing that every finitely satisfiable set is satisfiable.
- Zorn's lemma states that if in a partially ordered set A, every chain L has an upper bound $\widehat{L}$ in A, then A has a maximal element.
- Given a finitely satisfiable set Γ, the set A_Γ of finitely satisfiable supersets of Γ satisfies the conditions of Zorn's lemma.
- Hence there is a maximal extension $\widehat{\Gamma}$ that is finitely satisfiable.
- For any atom p, either $p \in \widehat{\Gamma}$ or $\neg p \in \widehat{\Gamma}$, but not both. Why?
- We can similarly define the model $M_{\widehat{\Gamma}}$ to show that $\widehat{\Gamma}$ is satisfiable.

Interpolation

Craig's interpolation property for one-sided sequents: If $\vdash \Gamma; \Delta$, then there is an I in the variables common to Γ and Δ such that $\vdash \Gamma, I$ and $\vdash \neg I, \Delta$.

Ax_1	$\dfrac{}{[\bot] \vdash \Gamma, P, \overline{P}; \Delta}$
Ax_2	$\dfrac{}{[\top] \vdash \Gamma; P, \overline{P}, \Delta}$
Ax_3	$\dfrac{}{[P] \vdash \Gamma, \overline{P}; P, \Delta}$
$\neg\neg$	$\dfrac{[I] \vdash \Gamma, P, \Delta}{[I] \vdash \Gamma, \neg\neg P, \Delta}$
$\vee$	$\dfrac{[I] \vdash \Gamma, A, B, \Delta}{[I] \vdash \Gamma, A \vee B, \Delta}$
$\neg\vee_1$	$\dfrac{[I_1] \vdash \Gamma, \neg A; \Delta \quad [I_2] \vdash \Gamma, \neg B; \Delta}{[[I_1 \vee I_2] \vdash \Gamma, \neg(A \vee B); \Delta}$
$\neg\vee_2$	$\dfrac{[I_1] \vdash \Gamma; \neg A, \Delta \quad [I_2] \vdash \Gamma; \neg B, \Delta}{[I_1 \wedge I_2] \vdash \Gamma; \neg(A \vee B), \Delta}$

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.
- Tautologies, i.e., 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	$\dfrac{K, l, \bar{l}}{\bot}$

Resolution: Example

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

Show that resolution is a sound and complete procedure for checking satisfiability.

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 $\Gamma \in K \cup C$ $\langle h', \Gamma' \rangle$ $= \mathit{analyze}(\psi)(\Gamma)$ for $\psi = h, \langle M \rangle, K, C$

- 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	$\neg q \vee \neg r$

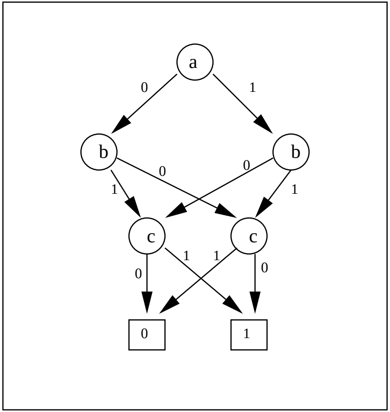
Show that CDCL is sound and complete.

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: $f(x_1,\ldots,x_n) = \texttt{IF}(x_1, f(\top, x_2,\ldots,x_n), f(\bot, x_2,\ldots,x_n))$
  2. Common subterms are identified
  3. Redundant branches are removed: $\texttt{IF}(x_i, A, A) = A$
- 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 $\wedge$ or $\vee$.
Construct an algorithm to compute $\exists \overline{x}.f$.

First and Higher-Order Logic

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 a partial ordering relation $\leq$, i.e., one that is reflexive, transitive, and anti-symmetric ($x \leq y \wedge y \leq x \Rightarrow x = y$).

Term Equality Logic (TEL)

- One further extension is to add function symbols from a signature Σ that assigns an arity to each symbol.
- Function symbols are used 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 TEL.

Show that TEL is decidable.

Equational Logic

- Equational Logic is a heavily used fragment of first-order logic.
- It consists of term equalities $s = t$, with proof rules
  1. Reflexivity: $\frac{}{s=s}$
  2. Symmetry: $\frac{s=t}{t=s}$
  3. Transitivity: $\frac{r=s \quad s=t}{r=t}$
  4. Congruence: $\frac{s_1=t_1,\ldots,s_n=t_n}{f(s_1,\ldots,s_n)=f(t_1,\ldots,t_n)}$
  5. Instantiation: $\frac{s=t}{\sigma(s)=\sigma(t)}$, for substitution σ.
- We say $\Gamma \vdash s = t$ when the equality $s = t$ can be derived from the equalities in Γ.
- Show that equational logic is sound and complete.

Use equational logic to formalize

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 a right-inverse operator x^{-1}
4. Commutative groups and semigroups
5. Rings: A set R with commutative group $\langle R, +, ^{-}, 0\rangle$, semigroup $\langle R, .\rangle$, and distributive laws $x.(y+z) = x.y + x.z$ and $(y+z).x = y.x + z.x$
6. Semilattice: A commutative semigroup $\langle S, \wedge\rangle$ with idempotence $x \wedge x = x$
7. Lattice: $\langle L, \wedge, \vee\rangle$ where $\langle L, \wedge\rangle$ and $\langle L, \vee\rangle$ are semilattices, and $x \vee (x \wedge y) = x$ and $x \wedge (x \vee y) = x$.
8. Distributive lattice: A lattice with $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$.
9. Boolean algebra: Distributive lattice with constants 0 and 1 and unary operation $-$ such that $x \wedge 0 = 0$, $x \vee 1 = 1$, $x \wedge -x = 0$, and $x \vee -x = 1$.

- Prove that every group element has a left inverse.
- For a lattice, define $x \leq y$ as $x \wedge y = x$. Show that $\leq$ is a partial order (reflexive, transitive, and antisymmetric).
- Show that a distributive lattice satisfies $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$.
- Prove the de Morgan laws, $-(x \vee y) = -x \wedge -y$ and $-(x \wedge y) = -x \vee -y$ for Boolean algebras.
- Prove that the set of integers $\mathbb{Z}$ form a commutative ring under addition and multiplication.
- A field is a ring where nonzero elements have a multiplicative inverse. Prove that the rationals and reals form a field under addition and multiplication.

First-Order Logic

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

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

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.

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

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

	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.
- A theory consists of a signature Σ for the function and predicate symbols and non-logical axioms.
- If a T is obtained from S by extending the signature and adding axioms, then T is conservative with respect to S, if all the formulas in S provable in T are also provable in S.

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 $\mathit{vars}(A)$ 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

- Prove
  1. $\neg\forall x.A \iff \exists x.\neg A$
  2. $(\forall x.A \wedge B) \iff (\forall x.A) \wedge (\forall x.B)$
  3. $(\exists x.A \vee B) \iff (\exists x.A) \vee (\exists x.B)$
  4. $((\forall x.A) \vee (\forall x.B)) \Rightarrow (\forall x.A \vee B)$
- Write the axioms for a partially ordered relation $\leq$.
- Write the axioms for a bijective (1-to-1, onto) function f.
- Write a formula asserting that for any x, there is a unique y such that $p(x, y)$.
- 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.

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.

What is an Inference Algorithm?

- An *Σ-inference structure* $\langle \Psi, \vdash, \Lambda, \mathcal{M} \rangle$ consists of
  - Ψ, a set of *logical states*
  - $\vdash$, the *reduction relation* between states
  - Λ, a map from states to Σ-formulas
  - $\mathcal{M}$, which extracts models from canonical states
- An *inference system* is an inference structure that is
  - *Conservative*: If $\psi \vdash \psi'$, then $\Lambda(\psi)$ and $\Lambda(\psi')$ are equisatisfiable.
  - *Progressive*: $\vdash$ is well-founded.
  - *Canonizing*: If $\psi \nvdash \psi'$ for any ψ', then either ψ is $\bot$ (i.e., unsatisfiable) or ψ is in a canonical form so that $\mathcal{M}(\psi)$ is a model for $\Lambda(\psi)$.
- It is *strongly conservative* if whenever $\psi \vdash \psi'$, then ψ and ψ' are equisatisfiable and any model of ψ' is also a model of ψ.
- We focus here on basic inference systems, but there are interesting variants.

What is an Inference Algorithm?

- An *inference algorithm* is an inference system where the reduction relation is presented as a collection of effective *inference rules* that transform an inference state ψ to an inference state ψ' such that $\psi \vdash \psi'$. **Example:** Ordered resolution is an algorithm for CNF satisfiability.
- Input K is a set of ordered clauses where the literals appear in decreasing order w.r.t. some order e.g., $q \prec \neg q \prec p \prec \neg p$.
- Tautologies, i.e., clauses containing both p and $\neg p$, are deleted from initial input.

Res	$\dfrac{K, p \vee \Gamma_1, \neg p \vee \Gamma_2}{K, p \vee \Gamma_1, \neg p \vee \Gamma_2, \Gamma_1 \vee \Gamma_2}$	$\Gamma_1 \vee \Gamma_2 \notin K$ $\Gamma_1 \vee \Gamma_2$ is not tautological
Contrad	$\dfrac{K, p, \neg p}{\perp}$	

- A set of clauses is canonical if it is closed under applications of **Res** and the **Contrad** rule is inapplicable.

Resolution: Example

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

- Drop the clause $\neg r$, and we reach an irreducible state from which a truth assignment $\{r \mapsto \top, q \mapsto \bot, p \mapsto \bot\}$ can be constructed.

Resolution as an Inference Algorithm

- The resolution inference system is *strongly conservative*: $\Gamma_1 \vee \Gamma_2$ is satisfiable if $p \vee \Gamma_1$ and $\neg p \vee \Gamma_2$ are.
- It is *progressive*: Bounded number of new clauses in the input variables.
- It is *canonizing*: Build a model M by assigning to atoms p_1 to p_n within a series of partial assignments $M_0, \ldots, M_n$:
    - M_0 is the empty truth assignment.
    - $M_{i+1} = M_i[p_{i+1} \mapsto v]$, where $v = \top$ iff there is some clause $p_{i+1} \vee \Gamma$ in the irreducible state K such that $M_i \models \neg\Gamma$.
- If $M_i \models \neg\Gamma$, then for any clause $\neg p_i \vee \Delta$, $M_i \models \Delta$ since $\Gamma \vee \Delta \in K$.
- **Invariant:** $M_i \models \Gamma$ for all clauses Γ in K in the atoms $p_1, \ldots, p_i$.
- Unordered resolution is also conservative, progressive, and canonizing, but it does not have the same set of canonical states.

Maintaining Equivalence with Union-Find

The logical state is a triple $\langle G, F \rangle$ with the input equalities and disequalities G and the find structure F which is a set of oriented equalities, i.e., orient $y = x$ as $x = y$ if $y \prec x$.

Delete	$\dfrac{x = y, G; F}{G; F}$ if $F(x) \equiv F(y)$
Merge	$\dfrac{x = y, G; F}{G; F' \circ F}$ if $F(x) \not\equiv F(y)$, $F' = \{orient(F(x) = F(y))\}$
Contrad	$\dfrac{x \neq y, G; F}{\bot}$ if $F(x) = F(y)$

- The above inference system is (strongly) conservative, progressive, and canonizing.
- Example: $x = y, x = z, u = v; \emptyset$ reduces to $\emptyset; x = z, y = z, u = v$.

Satisfiability Modulo Theories

- SMT deals with formulas with theory atoms like $x = y$, $x \neq y$, $x - y \leq 3$, and $select(store(A, i, v), j) = w$.
- 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				

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, ..., 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.	$\dfrac{\Gamma \vdash \pi_1(a) = \pi_1(b), \Delta, \ldots, \Gamma \vdash \pi_n(a) = \pi_i(b), \Delta}{\Gamma \vdash a = b, \Delta}$

Sets in Higher-Order Logic

- For a type T, the type of predicates over T is $[T \rightarrow \texttt{bool}]$.
- Predicates can be viewed as sets of elements from T.
- Define the empty set, the full set, the complement of a set, the union, intersection, and difference of two sets, the subset relation between two sets.
- Define a type that is a set of sets over T, and define the operation of taking the union and intersection over these set of sets.

Sequences in Higher-Order Logic

- Given the type $\mathbb{N}$ of natural numbers, a sequence σ over type T can be represented as $[\mathbb{N} \rightarrow T]$.
- If T is the type of $\mathbb{R}$ of real numbers, define the concept of a convergent series, i.e., there is some limit x such that for any $\epsilon > 0$, there is an N such that for any $n > N$, $|\sigma_n - x| \leq \epsilon$.
- Write a formal definition for the convergence of a series.
- Write a formal definition that x is the limit of a series σ.
- A Cauchy sequence σ is one where for any $\epsilon > 0$, there is an N such that for all $i, j > N$, $|\sigma_m - \sigma_n| < \epsilon$.
- Write a formal definition of a Cauchy sequence.
- Define a predicate that checks if one sequence σ is a subsequence of another sequence ρ.
- Show that every bounded sequence of reals has a convergent subsequence.

Functions in Higher-Order Logic

- Let f be a function from domain D to range R, i.e., in type $[D{\rightarrow}R]$.
- If D is some subtype of $\mathbb{R}$ and R is $\mathbb{R}$, then f is monotonically nondecreasing if $f(x) \leq f(y)$ whenever $x \leq y$.
- Define a predicate that checks that f is monotonically nondecreasing.
- A function is *continuous* in an interval I if for any $x \in I$ and $\epsilon > 0$, there is a $\delta > 0$ such that for any y if $|x - y| < \delta$, then $|f(x) - f(y)| < \epsilon$. Formalize.
- A function is *uniformly continuous* in I if for any $\epsilon > 0$ there is a $\delta > 0$ such that for any $x, y \in I$, if $|x - y| < \delta$, then $|f(x) - f(y)| < \epsilon$. Formalize.
- Formalize Lipschitz continuity: for any x in I, $|f(x) - f(y)| \leq K|x_1 - x_2|$.

Using Higher-Order Logic

- Define universal quantification using equality in higher-order logic.
- Express and prove Cantor's theorem (there is no injection from a type T to a $[T \rightarrow bool]$) in higher-order logic.
- Write the induction principle for Peano arithmetic in higher-order logic.
- Write a definition for the transitive closure of a relation in higher-order logic.
- Describe the modal logic CTL in higher-order logic.
- State and prove the Knaster-Tarski theorem.

Metric Spaces (from Wikipedia)

- A metric space is given by an ordered pair (M, d), where $d : M \times M \to \mathbb{R}$, where
  1. $d(x, y) \geq 0$ non-negativity or separation axiom
  2. $d(x, y) = 0 \Leftrightarrow x = y$ identity of indiscernibles
  3. $d(x, y) = d(y, x)$ symmetry
  4. $d(x, z) \leq d(x, y) + d(y, z)$ subadditivity or triangle inequality
- Define a complete metric space as a metric space that contains all limits of Cauchy sequences.
- Define compact metric spaces where every infinite set contains a sequence that converges to a limit point in the space.
- Define sequentially compact metric spaces where every infinite sequence has a convergent subsequence.

Continuity at a Point

- A function f from $\langle M_1, d_1\rangle$ to $\langle M_2, d_2\rangle$ is continuous at c if for any $\epsilon > 0$, there is a $\delta > 0$ such that for all x, $d_1(x, c) < \delta$, we have $d_2(f(x), f(c)) < \epsilon$.
- Show that a function f from $\langle M_1, d_1\rangle$ to $\langle M_2, d_2\rangle$ is continuous at c iff whenever a sequence $\langle x_i\rangle_{i\in\mathbf{N}}$, $lim_{i\to\infty} f(x_i) = f(c)$ if $lim_{i\to\infty} x_i = c$.
- Define uniform continuity and Lipschitz continuity at a point.
- Formalize the notion of A topological space $\langle X, T\rangle$ with T the open subsets of X such that $\emptyset \in T$, $X \in T$, and T is closed under finite/infinite unions and finite intersections.
- Define a function between $\langle X_1, T_1\rangle$ and $\langle X_2, T_2\rangle$ as continuous if the inverse image of open sets is always open.
- Define the derivative of a function on the reals.
- Define the Riemann integral of a function on the reals.

Vector Spaces

- A *vector space* V over a field F is closed under addition ($v + w \in V$ for $v, w \in V$) and scalar multiplication ($av \in V$ for $v \in V$), such that
  1. $+$ is associative and commutative with identity $\mathbf{0}$ and inverse $-$.
  2. $1v = v$; $a(bv) = (ab)v$; $(a + b)v = av + bv$; $a(v + w) = av + aw$;
- A *basis* for a vector space $\langle b_i \rangle_{i \in I}$ is a set of linearly independent vectors such that for any $v \in V$, there exists $\langle a_i \rangle_{i \in I}$ such that $v = \Sigma_{i \in I} a_i b_i$.
- A linear map L from vector space V to W preserves sums and scalar multiplication: $L(u + v) = L(u) + L(v)$ and $L(av) = a.L(v)$.
- Define the kernel $ker(L)$ of a linear map as $\{v \in V | L(v) = \mathbf{0}\}$. Show that $ker(L)$ is a vector space.

Partial Derivatives

- Define the partial derivative $\frac{\partial f(x_1,\ldots,x_n)}{\partial x_i}$ of a n-ary function on the reals at $a = \langle a_1, \ldots, a_n \rangle$.
- Define the *gradient* of f, ∇f, at a.
- Define the *Jacobian* of a vector-valued function f from $\mathbf{R}^m$ to $\mathbf{R}^n$.
- Define the *Hessian* of a function f from $\mathbf{R}^m$ to $\mathbf{R}$.

Advanced Topics

- 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.

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).

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.

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 Example

- 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 relations $x < y$ and $x \leq y$?

Using Dedekind–Peano Arithmetic

- Prove that
  1. $\forall x. x = 0 \vee (\exists y. s(y) = x)$
  2. $\forall x, y, z. (x + y) + z = x + (y + z)$
  3. $\forall x, y. x + y = y + x$
  4. $\forall x, y. x < y \implies \neg(y < x)$

Set Theory

Set theory can 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: $\forall x, y. \exists z. \forall u. u \in z. \iff u = x \vee u = y$ (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.)
- Separation: $\{x \in y | A\}$, for any formula A, $y \notin vars(A)$. (Define the intersection and disjointness of two sets.)
- Infinity: There is a set containing all the finite ordinals.
- Power set: For any set, there is a set of all its subsets.
- Regularity: Every set has an element that is disjoint from it.
- Replacement: There is a set that is the image Y of a set X with respect to a functional $(\forall x \in X. \exists! y. A(x, y))$ rule $A(x, y)$.

Using Set Theory

- Can two different sets be empty?
- For your definition of ordered pairing, define the first and second projection operations.
- Define the Cartesian product $x \times y$ of two sets, as the set of ordered pairs $\langle u, v\rangle$ such that $u \in x$ and $v \in y$.
- Define a subset of $x \times y$ to be functional if it does not contain any ordered pairs $\langle u, v\rangle$ and $\langle u', v\rangle$ such that $u \neq u'$.
- Define the function space y^x of the functions that map elements of x to elements of y.
- Define the join of two relations, where the first is a subset of $x \times y$ and the second is a subset of $y \times z$.

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.
- Let the formula $\neg Pr(S(x))$ be represented by the 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 $Pf(0, \underline{U})$ $Pf(1, \underline{U})$, $\dots$, 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

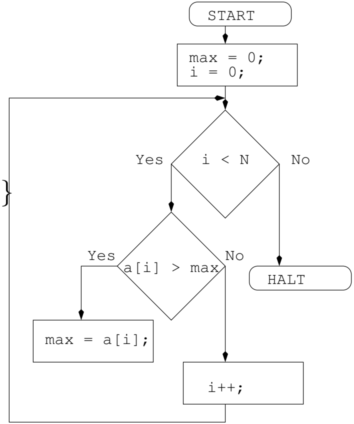
- The negation of the sentence U is Σ_1, and Z can verify Σ_1-completeness (*every valid Σ_1-sentence is provable*).
- Then
$$\vdash Pr(\underline{U}) \Rightarrow Pr(\underline{Pr(\underline{U})}).$$
- But this says $\vdash Pr(\underline{U}) \Rightarrow Pr(\underline{\neg U})$.
- Therefore $\vdash Con(Z) \Rightarrow \neg Pr(\underline{U})$.
- Hence $\not\vdash Con(Z)$, 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.

Floyd's method for Flowchart programs

- A flowchart has a *start* vertex with a single outgoing edge, a *halt* vertex with a single incoming edge.
- Each vertex corresponds to a program block or a decision conditions.
- Each edge corresponds to an assertion; the start edge is the flowchart *precondition*, and the halt edge is the flowchart *postcondition*.
- *Verification conditions* check that for each vertex, each incoming edge assertion through the block implies the outgoing edge assertion.
- *Partial correctness*: If each verification condition has been discharged, then every halting computation starting in a state satisfying the precondition terminates in a state satisfying the postcondition.
- *Total correctness*: If there is a ranking function mapping states to ordinals that strictly decreases for any cycle in the flowchart, then every computation terminates in the halt

Floyd's Method

```
max = 0;
i = 0;
{i ≤ N ∧ ∀(j < i): a[j] ≤ max}
while (i < N){
 if (a[i] > max){
   max = a[i];
   }
 i++;
 }
{∀(j < N): a[j] ≤ max}
```



Precondition is true, and postcondition is $\forall(\mathtt{j} < \mathtt{N}) : \mathtt{a}[\mathtt{j}] \leq \mathtt{max}$.

The *loop invariant* is $\mathtt{i} \leq \mathtt{N} \wedge \forall(\mathtt{j} < \mathtt{i}) : \mathtt{a}[\mathtt{j}] \leq \mathtt{max}$.

Hoare Logic

- A Hoare triple has the form $\{P\}S\{Q\}$, where S is a program statement in terms of the program variables drawn from the set Y and P and Q are assertions containing logical variables from X and program variables.
- A program statement is one of
  1. A *skip* statement *skip*.
  2. A *simultaneous assignment* $\overline{y} := \overline{e}$ where $\overline{y}$ is a sequence of n distinct program variables, e is a sequence of n $\Sigma[Y]$-terms.
  3. A *conditional* statement $e\ ?\ S_1\ :\ S_2$, where C is a $\Sigma[Y]$-formula.
  4. A *loop* *while* e *do* S.
  5. A sequential composition $S_1; S_2$.
- Express the `max` program using the language constructs above.

Skip	$\{P\}skip\{P\}$
Assignment	$\{P[\overline{e}/\overline{y}]\}\overline{y} := \overline{e}\{P\}$
Conditional	$\dfrac{\{C \wedge P\}S_1\{Q\} \quad \{\neg C \wedge P\}S_2\{Q\}}{\{P\}C \; ? \; S_1 \; : \; S_2\{Q\}}$
Loop	$\dfrac{\{P \wedge C\}S\{P\}}{\{P\}while \;\; C \;\; do \;\; S\{P \wedge \neg C\}}$
Composition	$\dfrac{\{P\}S_1\{R\} \quad \{R\}S_2\{Q\}}{\{P\}S_1; S_2\{Q\}}$
Consequence	$\dfrac{P \Rightarrow P' \quad \{P'\}S\{Q'\} \quad Q' \Rightarrow Q}{\{P\}S\{Q\}}$

Prove the partial correctness of the `max` program using the rules above.

Maximum Segment Sum Revisited

- Write a `while`-program for the maximum segment sum problem.
- Prove the program correct using Hoare logic.

Hoare Logic Semantics

- Both assertions and statements contain operations from a first-order signature Σ.
- An assignment σ maps program variables in Y to values in $dom(M)$.
- A program expression e has value $M[\![e]\!]\sigma$.
- The meaning of a statement $M[\![S]\!]$ is given by a sequence of states (of length at least 2).
  1. $\sigma \circ \sigma \in M[\![skip]\!]$, for any state σ.
  2. $\sigma \circ \sigma[M[\![\overline{e}]\!]\sigma/\overline{y}] \in M[\![\overline{y} := \overline{e}]\!]$, for any state σ.
  3. $\psi_1 \circ \sigma \circ \psi_2 \in M[\![S_1; S_2]\!]$ for $\psi_1 \circ \sigma \in M[\![S_1]\!]$ and $\sigma \circ \psi_2 \in M[\![S_2]\!]$
  4. $\psi \in M[\![C \ ? \ S_1 \ : \ S_2]\!]$ if either $M[\![C]\!]\psi[0] = \top$ and $\psi \in M[\![S_1]\!]$, or $M[\![C]\!]\psi[0] = \bot$ and $\psi \in M[\![S_2]\!]$
  5. $\sigma \circ \sigma \in M[\![while\ C\ do\ S]\!]$ if $M[\![C]\!]\sigma = \bot$
  6. $\psi_1 \circ \sigma \circ \psi_2 \in M[\![while\ C\ do\ S]\!]$ if $M[\![C]\!](\psi_1[0]) = \top$, $\psi_1 \circ \sigma \in M[\![S]\!]$, and $\sigma \circ \psi_2 \in M[\![while\ C\ do\ S]\!]$

Soundness of Hoare Logic

- $\{P\}S\{Q\}$ is *valid* in a Σ-structure M if for every sequence $\sigma \circ \psi \circ \sigma' \in M[\![S]\!]$ and any assignment ρ of values in *dom*(M) to logical variables in X, either
  1. $M[\![Q]\!]^{\rho}_{\sigma'} = \top$, or
  2. $M[\![P]\!]^{\rho}_{\sigma} = \bot$.
- Informally, every computation sequence for S either ends in a state satisfying Q or starts in a state falsifying P.
- Demonstrate the soundness of the Hoare calculus.

Completeness of Hoare Logic

- The proof of a valid triple $\{P\}S\{Q\}$ can be decomposed into
  1. The valid triple $\{wlp(S)(Q)\}S\{Q\}$, and
  2. The valid assertion $P \Rightarrow wlp(S)(Q)$
- $wlp(S)(Q)$ (the *weakest liberal precondition*) is an assertion such that for any $\psi \in M[\![S]\!]$ with $|\psi| = n+1$ and ρ, either $M[\![Q]\!]^{\rho}_{\psi_n} = \bot$ or $M[\![wlp(S)(Q)]\!]^{\rho}_{\psi_0} = \top$.
- Show that for any S and Q, the valid triple $\{wlp(S)(Q)\}S\{Q\}$ can be proved in the Hoare calculus. (Hint: Use induction on S.)
- First-order arithmetic over $\langle +, ., 0, 1\rangle$ is sufficient to express $wlp(S)(Q)$ since it can code up sequences of states representing computations.

Transition Systems: Mutual Exclusion

initially
$\mathtt{try}[1] = \mathtt{critical}[1] = \mathtt{turn} = \mathtt{false}$
transition

$$\begin{array}{rcl}
\neg\mathtt{try}[1] & \to & \mathtt{try}[1] := \mathtt{true}; \\
 & & \mathtt{turn} := \mathtt{false}; \\
\neg\mathtt{try}[2] \vee \mathtt{turn} & \to & \mathtt{critical}[1] := \mathtt{true}; \\
\mathtt{critical}[1] & \to & \mathtt{critical}[1] := \mathtt{false}; \\
 & & \mathtt{try}[1] := \mathtt{false};
\end{array}$$

$\|$

initially
$\mathtt{try}[2] = \mathtt{critical}[2] = \mathtt{false}$
transition

$$\begin{array}{rcl}
\neg\mathtt{try}[2] & \to & \mathtt{try}[2] := \mathtt{true}; \\
 & & \mathtt{turn} := \mathtt{true}; \\
\neg\mathtt{try}[1] \vee \neg\mathtt{turn} & \to & \mathtt{critical}[2] := \mathtt{true}; \\
\mathtt{critical}[2] & \to & \mathtt{critical}[2] := \mathtt{false}; \\
 & & \mathtt{try}[2] := \mathtt{false};
\end{array}$$

Model Checking Transition Systems

- A transition system is given as a triple $\langle W, I, N\rangle$ of states W, an initialization predicate I, and a transition relation N.
- Symbolic Model Checking: Fixpoints such as $\mu X.I \sqcup post(N)(X)$ which is the set of reachable states can be constructed as an ROBDD.
- Bounded Model Checking: $I(s_0) \wedge \bigwedge_{i=0}^{k} N(s_i, s_{i+1})$ represents the set of possible $(k+1)$-step computations and $\neg P(s_{k+1})$ represents the possible violations of state predicate P at the state s_{k+1}.
- k-Induction: A variant of bounded model checking can be used to prove properties:
  - Base: Check that P holds in the first k states of the computation
  - Induction: If P holds for any sequence of k steps in a computation, it holds in the $k+1$-th state.
- Prove the mutual exclusion property by k-induction.

- Many computational systems can be modeled as transition systems.
- A transition system is a triple $\langle W, I, N\rangle$ consisting of a set of states W, an initialization predicate I, and transition relation N.
- Transition system properties include invariance, stability, eventuality, and refinement.
- Finite-state transition systems can be analyzed by means of state exploration.
- Properties of infinite-state transition systems can be proved using various combinations of theorem proving and model checking.

States and Transitions in PVS

Given some `state` type, an assertion is a predicate on this type, and action is a relation between states, and a computation is an infinite sequence of states.

```
state[state: TYPE] : THEORY
BEGIN
  IMPORTING  sequences[state]

  statepred: TYPE = PRED[state]  %assertions

  Action: TYPE =  PRED[[state, state]]

  computation : TYPE = sequence[state]

  pp: VAR statepred
  action: VAR Action
  aa, bb, cc: VAR computation
```

States and Transitions

- A run is valid if the initialization predicate pp holds initially, and the action aa holds of each pair of adjacent states.
- An invariant assertion holds of each state in the run.

```
  Init(pp)(aa) : bool = pp(aa(0))

   Inv(action)(aa) : bool =
    (FORALL (n : nat) : action(aa(n), aa(n+1)))

   Run(pp, action)(aa): bool =
    (Init(pp)(aa) AND Inv(action)(aa))

   Inv(pp)(aa) : bool =
   (FORALL (n : nat) : pp(aa(n)))
END state
```

(Simplified) Peterson's Mutual Exclusion Algorithm

- The algorithm ensures mutual exclusion between two processes `P` and `Q`.
- The global state of the algorithm is a record consisting of the program counters `PCP` and `PCQ`, and boolean `turn` variable.

```
mutex  : THEORY
  BEGIN

    PC : TYPE = sleeping, trying, critical

    state : TYPE = [# pcp : PC,
                      turn: bool,
                     pcq : PC #]

    IMPORTING  state[state]

    s, s0, s1: VAR state
```

Defining Process P

P is initially sleeping. It moves to trying by setting the turn variable to FALSE, and enters the critical state if Q is sleeping or turn is TRUE.

```
I_P(s) : bool = (sleeping?(pcp(s)))

G_P(s0, s1): bool =
  (   (s1 = s0)                         %stutter
   OR (sleeping?(pcp(s0)) AND      %try
         s1 = s0 WITH [pcp := trying, turn := FALSE])
   OR (trying?(pcp(s0)) AND        %enter critical
        (turn(s0) OR sleeping?(pcq(s0))) AND
       s1 = s0 WITH [pcp := critical])
   OR (critical?(pcp(s0)) AND      %exit critical
        s1 = s0 WITH [pcp := sleeping, turn := FALSE ]))
```

Defining Process Q

Process Q is similar to P with the dual treatment of the turn variable.

```
I_Q(s) : bool = (sleeping?(pcq(s)))

G_Q(s0, s1): bool =
  (  (s1 = s0)                          %stutter
   OR (sleeping?(pcq(s0)) AND     %try
        s1 = s0 WITH [pcq := trying, turn := TRUE])
   OR (trying?(pcq(s0))  AND      %enter
       (NOT turn(s0) OR sleeping?(pcp(s0))) AND
      s1 = s0 WITH [pcq := critical])
   OR (critical?(pcq(s0)) AND     %exit critical
       s1 = s0 WITH [pcq := sleeping, turn := TRUE]))
```

The Combined System

The system consists of:

- The conjunction of the initializations for `P` and `Q`
- The disjunction of the actions for `P` and `Q` (interleaving).

```
   I(s) : bool = (I_P(s) AND I_Q(s))

   G(s0, s1) : bool = (G_P(s0, s1) OR G_Q(s0, s1))

END mutex
```

Proving Mutual Exclusion

`safe` is the assertion that `P` and `Q` are not simultaneously `critical`.

```
mutex_proof: THEORY
 BEGIN
   IMPORTING mutex, connectives[state]
   s, s0, s1: VAR state

   safe(s) : bool = NOT (critical?(pcp(s)) AND critical?(pcq(s)))

   safety_proved: CONJECTURE
      (FORALL (aa: computation):
        Run(I, G)(aa)
        IMPLIES Inv(safe)(aa))
```

`safety_proved` asserts the invariance of `safe`.

Proving Mutual Exclusion

```
safety_proved :

  |-------
{1}   (FORALL (aa: computation):
            Run(I, G)(aa) IMPLIES Inv(safe)(aa))

Rule? (reduce-invariant)
 :
 :
Apply the invariance rule,,
this yields  11 subgoals:
```

reduce-invariant is a proof strategy that reduces the task to that of showing that each transition preserves the invariant.

Proving Mutual Exclusion

```
safety_proved.1 :

{-1}  Init(I)(aa!1)
  |-------
{1}   safe(aa!1(0))

Rule? (grind)

  ⋮
Trying repeated skolemization, instantiation, and if-lifting,

This completes the proof of safety_proved.1.
```

Proving Mutual Exclusion

```
safety_proved.2 :

{-1}  (aa!1(1 + (j!1 + 1 - 1)) = aa!1(j!1 + 1 - 1))
{-2}  safe(aa!1(j!1))
  |-------
{1}   safe(aa!1(j!1 + 1))

Rule? (grind)

 ⋮
Trying repeated skolemization, instantiation, and if-lifting,

This completes the proof of safety_proved.2.
```

Proving Mutual Exclusion

```
safety_proved.3 :

{-1}  sleeping?(pcp(aa!1(j!1 + 1 - 1)))
{-2}  aa!1(1 + (j!1 + 1 - 1)) =
       aa!1(j!1 + 1 - 1) WITH [pcp := trying, turn := FALSE]
{-3}  safe(aa!1(j!1))
  |-------
{1}   safe(aa!1(j!1 + 1))

Rule? (grind)

 :

Trying repeated skolemization, instantiation, and if-lifting,

This completes the proof of safety_proved.3.
```

Proving Mutual Exclusion

```
safety_proved.4 :

{-1}  turn(aa!1(j!1 + 1 - 1))
{-2}  trying?(pcp(aa!1(j!1 + 1 - 1)))
{-3}  aa!1(1 + (j!1 + 1 - 1))
        = aa!1(j!1 + 1 - 1) WITH [pcp := critical]
{-4}  safe(aa!1(j!1))
  |-------
{1}   safe(aa!1(j!1 + 1))

Rule? (grind)
safe rewrites safe(aa!1(j!1))
  to TRUE
safe rewrites safe(aa!1(1 + j!1))
  to NOT critical?(pcq(aa!1(1 + j!1)))
Trying repeated skolemization, instantiation, and if-lifting,
this simplifies to:
```

Proving Mutual Exclusion

```
safety_proved.4 :

{-1}  aa!1(j!1)`turn
{-2}  trying?(pcp(aa!1(j!1)))
{-3}  aa!1(1 + j!1) = aa!1(j!1) WITH [pcp := critical]
[-4]  safe(aa!1(j!1))
{-5}  critical?(aa!1(j!1)`pcq)
  |-------
```

Unprovable subgoal!

Invariant is too weak, and is not inductive.

Strengthening the Invariant

```
strong_safe(s) : bool =
     ((critical?(pcp(s)) IMPLIES (turn(s) OR sleeping?(pcq(s))))
     AND
     (critical?(pcq(s)) IMPLIES (NOT turn(s) OR sleeping?(pcp(s)))))

  strong_safety_proved: THEOREM
      (FORALL (aa: computation):
          Run(I, G)(aa)
        IMPLIES Inv(strong_safe)(aa))
```

Verified by `(then (reduce-invariant) (grind))`.

Strong Invariant Implies Weak

```
strong_safe_implies_safe :

  |-------
{1}   FORALL (s: state): (strong_safe IMPLIES safe)(s)

Rule? (grind)
  :
  :
Trying repeated skolemization, instantiation, and if-lifting,
Q.E.D.
```

Predicate Transformers

- Given a state type `state`, we already saw that assertions over this state type have the type `pred[state]`.
- Predicate transformers over this type can be given the type `[pred[state] -> pred[state]]`.

```
relation_defs [T1, T2: TYPE]: THEORY
 BEGIN
  R: VAR pred[[T1, T2]]
  X: VAR set[T1]
  Y: VAR set[T2]
   preimage(R)(Y): set[T1] = preimage(R, Y)
   postcondition(R)(X): set[T2] = postcondition(R, X)
   precondition(R)(Y): set[T1] = precondition(R, Y)
 END relation_defs
```

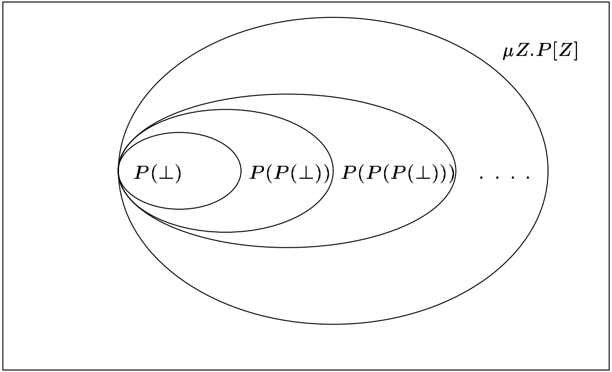
The Mu-Calculus

```
mucalculus[T:TYPE]: THEORY
 BEGIN
  s: VAR T
  p, p1, p2: VAR pred[T]
  predicate_transformer: TYPE = [pred[T]->pred[T]]
  pt: VAR predicate_transformer
  setofpred: VAR pred[pred[T]]

  <=(p1,p2): bool = FORALL s: p1(s) IMPLIES p2(s)

  monotonic?(pt): bool =
      FORALL p1, p2: p1 <= p2 IMPLIES pt(p1) <= pt(p2)

  pp: VAR (monotonic?)

  glb(setofpred): pred[T] =
      LAMBDA s: (FORALL p: member(p,setofpred) IMPLIES p(s))
```

The Mu-Calculus

```
 % least fixpoint
 lfp(pp): pred[T] = glb({p | pp(p) <= p})

 mu(pp): pred[T] = lfp(pp)

 lub(setofpred): pred[T] =
     LAMBDA s: EXISTS p: member(p,setofpred) AND p(s)

 % greatest fixpoint
 gfp(pp): pred[T] = lub({p | p <= (pp(p))})

 nu(pp): pred[T] = gfp(pp)

END mucalculus
```

The Least Fixed Point



Exercises

1. P is $\cup$-continuous if $\langle X_i | i \in \mathbf{N}\rangle$ is a family of sets (predicates) such that $X_i \subseteq X_{i+1}$, then $P(\bigcup_i(X_i)) = \bigcup_i(P(X_i))$.
2. Show that $(\mu Z.P[Z])(z_1, \ldots, z_n) = \bigvee_i P^i[\bot](z_1, \ldots, z_n)$, where $\bot = \lambda z_1, \ldots, z_n : \mathbf{false}$.
3. Similarly, P is P is $\cap$-continuous if $\langle X_i | i \in \mathbf{N}\rangle$ is a family of sets (predicates) such that $X_{i+1} \subseteq X_i$, then $P(\bigcap_i(X_i)) = \bigcap_i(P(X_i))$.
4. Show that $(\nu Z.P[Z])(z_1, \ldots, z_n) = \bigwedge_i P^i[\top](z_1, \ldots, z_n)$, where $\top = \lambda z_1, \ldots, z_n : \mathbf{true}$.

Fixed Point Computations

- The set of reachable states is fundamental to model checking
  - Any initial state is reachable.
  - Any state that can be reached in a single transition from a reachable state is reachable.
  - These are all the reachable states.
- This is a least fixed point:
  `mu X: LAMBDA y: I(y) OR EXISTS x: N(x, y) AND X(x).`
- An invariant is an assertion that is true of all reachable states: $\mathbf{AG}p$.

Temporal Connectives

```
ctlops[state : TYPE]: THEORY
 BEGIN
  u,v,w: VAR state
  f,g,Q,P,p1,p2: VAR pred[state]
  Z: VAR pred[[state, state]]

  N: VAR [state, state -> bool]

  EX(N,f)(u):bool = (EXISTS v: (f(v) AND N(u, v)))

  EU(N,f,g):pred[state] = mu(LAMBDA Q: (g OR (f AND EX(N,Q))))

  EF(N,f):pred[state] = EU(N, TRUE, f)

  AG(N,f):pred[state] = NOT EF(N, NOT f)
 END ctlops
```

Symbolic Fixed Point Computations

- If the computation state is represented as a boolean array $b[1..N]$,
- Then a set of states can be represented by a boolean function mapping $\{0,1\}^N$ to $\{0,1\}$.
- Boolean functions can represent
  - Initial state set
  - Transition relation
  - Image of transition relation with respect to a state set
- Set of reachable states computable as a boolean function.
- ROBDD representation of boolean functions empirically efficient.

Model Checking Peterson's Algorithm

```
mutex_mc: THEORY
 BEGIN
   IMPORTING mutex_proof
   s, s0, s1: VAR state

   safety: LEMMA
     I(s) IMPLIES
          AG(G, safe)(s)
    ⋮
  END mutex_mc
```

The model-check Command

```
safety :

  |-------
{1}   FORALL (s: state): I(s) IMPLIES AG(G, safe)(s)

Rule? (auto-rewrite-theories "mutex" "mutex_proof")
Installing rewrites from theories:  mutex mutex_proof,
this simplifies to:
safety :

  |-------
[1]   FORALL (s: state): I(s) IMPLIES AG(G, safe)(s)

Rule? (model-check)
  :
  :
By rewriting and mu-simplifying,
Q.E.D.
```

Fairness

- For state `s`, the property `fairEG(N, f)(Ff)(s)` holds when the predicate `f` holds along every *fair* path.
- For fairness condition `Ff`, a fair path is one where `Ff` holds infinitely often.
- This is given by the set of states that can `P` that can always reach `f AND Ff AND EX(N, P)` along an `f` path.
-
```
  fairEG(N, f)(Ff): pred[state] =
    nu(LAMBDA P: EU(N, f, f AND Ff AND EX(N, P)))
```

Linear-Time Temporal Logic (LTL)

$$
\begin{array}{rcl}
s \models a & = & s(a) = \textbf{true} \\
s \models \neg A & = & s \not\models A \\
s \models A_1 \vee A_2 & = & s \models A_1 \text{ or } s \models A_2 \\
s \models \mathbf{A}L & = & \forall \sigma : \sigma(0) = s \text{ implies } \sigma \models L \\
s \models \mathbf{E}L & = & \exists \sigma : \sigma(0) = s \text{ and } \sigma \models L \\
\sigma \models a & = & \sigma(0)(a) = \textbf{true} \\
\sigma \models \neg L & = & \sigma \not\models L \\
\sigma \models L_1 \vee L_2 & = & \sigma \models L_1 \text{ or } \sigma \models L_2 \\
\sigma \models \mathbf{X}A & = & \sigma\langle 1\rangle \models A \\
\sigma \models A_1 \;\mathbf{U}\; A_2 & = & \exists j : \sigma\langle j\rangle \models A_2 \text{ and } \forall i < j : \sigma\langle i\rangle \models A_1
\end{array}
$$

Exercise: Embed LTL semantics in PVS.

Interpolation-Based Model Checking

- Interpolation: The unsatisfiability of the BMC query yields an interpolant Q such that $I(s_0) \wedge N(s_0, s_1)$ and $\bigwedge_{i=1}^{k} N(s_i, s_{i+1}) \wedge \neg P(s_{k+1})$ are jointly unsatisfiable.
- The proof yields an interpolant $Q(s_1)$.
- Let $I'(s_0)$ be $I(s_0) \vee Q(s_0)$.
- If $I(s_0) = I'(s_0)$ then this is an invariant. Otherwise, repeat the process with I replaced by I'.
- Prove the mutual exclusion property using interpolation-based model checking.

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
- Shankar, *Model Checking and Deduction*, Handbook of Model Checking (To appear), 2018.