Announcements

- Midterm is next Tuesday (02/24)
- Covers all lectures so far
- Exam is closed-book, closed-notes, closed-laptops
- But allowed to bring two sheets of notes prepared by you

Agenda for Today

- Properties of first order logic:
  - compactness
  - inexpressibility of transitive closure in FOL
- Ingredients of first-order theorem proving:
  - unification
  - Most general unifiers

Compactness of First-Order Logic

- Another important property of FOL is compactness.
- A logic is called compact if an infinite set of sentences $\Gamma$ that are unsat, but every finite subset $\Sigma$ is sat.
- Theorem (due to Gödel): First-order logic is compact.
- Proof of compactness of FOL follows from the completeness of proof rules.

Proof of Compactness

- Recall: Completeness means that if a formula is unsatisfiable, then there exists a finite-length proof of unsatisfiability.
- Suppose FOL was not compact, i.e., there is an infinite set of sentences $\Gamma$ that are unsat, but every finite subset $\Sigma$ is sat.
- By completeness of proof rules, if $\Gamma$ is unsat, there exists a finite-length proof of unsatisfiability.
- But this means the proof must use a finite subset of sentences $\Sigma$ of $\Gamma$, otherwise proof could not be finite.
- But this implies there is also a proof of unsatisfiability of $\Sigma$.
- Thus, by soundness of proof rules, $\Sigma$ must be unsat.

Consequences of Compactness

- Proof of compactness might look like a useless property, but it has very interesting consequences!
- Compactness can be used to show that a variety of interesting properties are not expressible in first-order logic.
- For instance, we can use compactness theorem to show that transitive closure is not expressible in first order logic.
**Transitive Closure**

- Given a directed graph $G = (V, E)$, the transitive closure of $G$ is defined as the graph $G^* = (V, E^*)$ where:
  $$E^* = \{ (n, n') \mid \text{if there is a path from vertex } n \text{ to } n' \}$$

- **Observe:** A binary predicate $p(t, t')$ be viewed as a graph containing an edge from node $t$ to $t'$.

- Thus, the concept of transitive closure applies to binary predicates as well.

- A binary predicate $T$ is the transitive closure of predicate $p$ if 
  $$\langle h_0, h_n \rangle \in T \iff \text{there exists some sequence } h_0, h_1, \ldots, h_n \text{ such that } (h_i, h_{i+1}) \in p$$

**“Expressing” Transitive Closure in FOL**

- At first glance, it looks like transitive closure $T$ of binary relation $p$ is expressible in FOL:
  $$\forall x, \forall z, (T(x, z) \iff (p(x, z) \lor \exists y. p(x, y) \land T(y, z)))$$

- But this formula does not describe transitive closure at all!

- To see why, consider $U = \mathbb{N}$, $p$ is equality predicate, and $T$ is relation that is true for any number $x, y$.

- Clearly, this $T$ is not the transitive closure of equality, but this structure is actually a model of the formula.

- Thus, the formula above is not a definition of transitive closure at all!

**Proof I**

- **Ψn(a, b) encode the proposition:** there is no path of length $n$ from $a$ to $b$.

- In particular, $\Psi^1 = \neg p(a, b)$

- Similarly,
  $$\Psi^n = \neg \exists x_1, \ldots, x_{n-1}. (p(a, x_1) \land p(x_1, x_2) \land \ldots \land p(x_{n-1}, b))$$

**Proof II**

- Recall: $\Gamma$ is a set of propositions encoding $T$ is transitive closure of $p$.

- Now, construct $\Gamma'$ as follows:
  $$\Gamma' = \Gamma \cup \{ T(a, b), \Psi^1, \Psi^2, \Psi^3, \ldots \}$$

- **Observe:** $\Gamma'$ is unsatisfiable because:
  1. Since $\Gamma$ encodes that $T$ is transitive closure of $p$, $T(a, b)$ says there is some path from $a$ to $b$.
  2. The infinite set of propositions $\Psi^1, \Psi^2, \ldots$ says that there is no path of any length from $a$ to $b$.

**Proof III**

- Now, consider any finite subset of $\Gamma'$:
  $$\Gamma' = \Gamma \cup \{ T(a, b), \Psi^1, \Psi^2, \Psi^3, \ldots \}$$

- Clearly, any finite subset does not contain $\Psi_i$ for some $i$.

- **Observe:** This finite subset is satisfied by a model where there is a path of length $i$ from $a$ to $b$.

- Thus, every finite subset of $\Gamma'$ is satisfiable.

- By the compactness theorem, this would imply $\Gamma'$ is also satisfiable.

- But we just showed that $\Gamma'$ is unsatisfiable!

- Thus, transitive closure cannot be expressed in FOL!
First-Order Theorem Provers

- A first-order theorem prover is a computer program that proves the validity of formulas in first-order logic.
- Since validity in FOL is only semi-decidable, first-order theorem provers are not guaranteed to terminate.
- Despite this limitation, many automated theorem provers exist and are useful: Vampire, SPASS, Otter, ...
- There are even annual competitions between these theorem provers! (just Google “CADE ATP competition”)
- Main applications: software verification and synthesis, artificial intelligence, and proving mathematical theorems.

Theorem Provers and Mathematical Theorems

- First-order theorem provers have been used to prove some mathematical theorems not previously proven by humans.
- Robbins conjecture (1933): Mathematician Herbert Robbins conjectured that a group of axioms he came up with are equivalent to boolean algebra.
- Neither he nor anyone else could prove this for decades.

Robbins Conjecture and Automated Theorem Proving

- 1996: Conjecture eventually proven by first-order theorem prover EQP after 8 days of search!
- That a computer can prove theorems that humans could not was shocking.
- The automated proof of Robbins conjecture even appeared as New York Times article!
- Not the only success story: Otter used by mathematician Ken Kunnen to prove results in quasi-group theory.

Overview

- Today’s lecture and next lecture: Discuss basic principles underlying first-order theorem provers.
- Not meant to be a complete survey of the field, but just to highlight basic principles.
- The basis underlying all theorem provers today is the principle of first-order resolution.
- First-order theorem provers prove formulas unsatisfiable by showing there is a resolution refutation for that formula.

Recall: Propositional Resolution

- Earlier we talked about resolution in propositional logic.
- Recall: Consider two clauses in CNF:
  \( C_1: (l_1 \lor \ldots \lor l_k) \quad C_2: (l'_1 \lor \ldots \lor l'_n) \)
- Propositional resolution: Deduction of a new clause \( C_3 \), called resolvent:
  \( C_3: (l_1 \lor \ldots \lor l_k \lor l'_1 \lor \ldots \lor l'_n) \)
- First-order resolution is the same basic principle, but a little bit more involved.

First-Order Resolution Prerequisites

- To perform resolution in first-order logic, we need two new ingredients:
  1. Unification: Which expressions can be made identical?
  2. Clausal form: A new normal form for FOL
- Today, we'll talk about unification.
- Save resolution, clausal form for next lecture.
Unification

- **Unification**: problem of determining if two expressions can be made identical by appropriate substitutions for their variables
- **Substitution**: finite mapping from variables to terms
- **Example**: Can expressions $p(x)$ and $p(a)$ be unified? Yes, using the substitution $[x → a]$
- Can $p(a)$ and $p(b)$ be unified? No
- We’ll write $eσ$ to denote the application of substitution $σ$ to expression $e$
- What is $p(x)[x → a]$? $p(a)$

Non-Uniqueness of Unifiers

- If two expressions are unifiable, they don’t necessarily have a unique unifier.
- **Example**: $p(x, y)$ and $p(a, v)$
  - Unifier 1: $[x → a, y → b, v → b]$
  - Unifier 2: $[x → a, y → v]$
  - Unifier 3: $[x → a, y → f(b), v → f(b)]$
- But some unifiers are more desirable than others . . .

Composing Substitutions

- To explain what it means for one unifier to be better than another, we define the composition of substitutions.
- Composition of two substitutions $σ$ and $δ$ is written $σδ = σ'$
- The composition $σδ$ of substitutions $σ$ and $δ$ is obtained by:
  1. applying $δ$ to the range of $σ$
  2. add to $σ$ all mappings $x → t$ from $δ$ where $x \notin \text{dom}(σ)$

Composing Substitutions Examples

- What is $[x → a, y → z][z → b]$?
  - $[x → a, y → b, z → b]$
- What is $[x → a, y → f(z, g(w))][z → 1, w → 2]$?
  - $[x → a, y → f(1, g(2)), z → 1, w → 2]$
- Let
  - $σ = [x → a, y → f(u), z → v]$
  - $δ = [u → d, v → e, z → g]$
  - What is $σδ$?
  - $[x → a, y → f(d), z → e, u → d, v → e]$

Generality of Unifiers

- We prefer unifiers that are as general as possible.
- A unifier $σ$ is **at least as general** as unifier $σ'$ if there exists another substitution $δ$ such that $σδ = σ'$
- Intuition: $σ$ more general than $σ'$ if $σ'$ can be obtained from $σ$ through another substitution
- Which unifier is more general? $σ = [x → a, y → v]$ or $σ' = [x → a, y → f(c), v → f(c)]$?
  - $σ$
- Which unifier is more general? $σ = [x → a, y → z]$ or $σ' = [x → a, y → w]$? equally general
Most General Unifiers

- A substitution $\sigma$ is a most general unifier (mgu) of two expressions $e, e'$ iff $\sigma$ is at least as general as any other unifier of $e$ and $e'$.
- Intuition: A unifier is most general if it only contains mappings necessary to unify, but nothing extra.
- Consider again $p(x, y)$ and $p(a, v)$.
- Is $[x \mapsto a, y \mapsto b, v \mapsto b]$ an mgu? No
- Is $[x \mapsto a, y \mapsto v]$ an mgu? Yes
- Is $[x \mapsto a, y \mapsto v, v \mapsto y]$ an mgu? No, not even unifier

Algorithm to Compute MGU

- We’ll now give an algorithm to find most general unifiers.
- Function $\text{find-mgu}(e, e')$ takes expressions $e, e'$ and returns substitution $\sigma$ that is mgu of $e, e'$ or $\bot$.
- Case 1: $e = e'$. Then $\sigma = []$.
- Case 2: $e$ is variable $x$. If $e'$ does not contain $x$ then $[x \mapsto e']$, otherwise $\bot$.
- Case 3: $e'$ is variable $y$. If $e$ does not contain $y$ then $[y \mapsto e]$, otherwise $\bot$.
- Case 4: $e$ or $e'$ is a constant. Return $\bot$.

Example of Computing MGUs

- Apply algorithm to find mgu for $p(f(x), f(x))$ and $p(y, f(a))$
- Predicates match; unify the arguments.
- Unify first arguments $f(x)$ and $y$
- Result: $[y \mapsto f(x)]$
- Apply unifier to second arguments $f(x)$ and $f(a)$ (unchanged)
- Then, unify $f(x)$ and $f(a)$: $[x \mapsto a]$
- Compose $[y \mapsto f(x)]$ and $[x \mapsto a]$
- Final result: $[y \mapsto f(a), x \mapsto a]$

Uniqueness of Most General Unifiers

- Theorem: If two expressions $e$ and $e'$ are unifiable, then they have an mgu that is unique up to variable permutation.
- “Unique up to variable permutation” means only difference between two most general unifiers is variable names.
- What are all possible most general unifiers of $p(x, y)$ and $p(a, v)$?
- $[x \mapsto a, y \mapsto v]$ and $[x \mapsto a, v \mapsto y]$

Algorithm to Compute MGU, continued

- Case 5: $e = \tau(e_1, \ldots, e_k)$.
  1. If $e' \neq \tau(e'_1, \ldots, e'_k)$, then $\bot$.
  2. Otherwise result of unifying $[e_1, \ldots, e_k]$ and $[e'_1, \ldots, e'_k]$.
- Case 6: $e$ is expression list $[h \ 1]$
  1. If $e'$ is not expression list of the form $[h' \ T']$, return $\bot$.
  2. Let $\sigma = \text{find-mgu}(h, h')$.
  3. Apply $\sigma$ to $T, T'$.
  4. Recursively compute MGU $\sigma'$ for $\sigma T$ and $\sigma T'$.
  5. Return composition of $\sigma$ and $\sigma'$.

Another Example

- Apply algorithm to find mgu for $p(x, x)$ and $p(y, f(y))$
- Predicates match; unify the arguments.
- Unify first arguments $x$ and $y$: $[x \mapsto y]$
- Apply unifier to second arguments $x$ and $f(y)$: $y, f(y)$
- Now unify $y$ and $f(y)$: $\bot$
- Thus $p(x, x)$ and $p(y, f(y))$ not unifiable