Announcements

- Midterm is next Tuesday (March 3)
- 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:
  - decidable fragments
  - compactness
  - inexpressibility of transitive closure in FOL
- Ingredients of first-order theorem proving:
  - unification
  - Most general unifiers

Decidable Fragments of First-Order Logic

- Although full-first order logic is not decidable, there are fragments of FOL that are decidable.
- A fragment of FOL is a syntactically restricted subset of full FOL: e.g., no functions, or only universal quantifiers, etc.
- Some decidable fragments:
  - Quantifier-free first order logic
  - Monadic first-order logic
  - Bernays-Schönfinkel class

Bernays-Schönfinkel Class

- The Bernays-Schönfinkel class is a fragment of FOL where:
  1. there are no function constants,
  2. only formulas of the form:
     \[ \exists x_1, \ldots, \exists x_n, \forall y_1, \ldots, \forall y_m, P(x_1, \ldots, x_n, y_1, \ldots, y_m) \]
- Result: The Bernays-Schönfinkel fragment of FOL is decidable
- Database query language Datalog is based on Bernays-Schönfinkel class of FOL
- However, it has additional restriction that all clauses are Horn clauses (i.e., at most one positive literal in each clause)

Datalog

- Datalog is a programming language that allows adding/querying facts in a deductive databases
- An example Datalog program:
  
  ```datalog
  parent(bill, mary). % Bill is Mary’s parent
  parent(mary, john). % Mary is John’s parent
  ancestor(X,Y) :- parent(X,Y).
  ancestor(X,Z) :- parent(X,Y), ancestor(Y,Z).
  ?-ancestor(X,john).
  ```
- Last statement is a query: Is there anyone in the database who is John’s ancestor (and if so, who?)
### Datalog, cont.

- `parent(bill, mary). % Bill is Mary's parent`
- `parent(mary, john). % Mary is John's parent`

- `ancestor(X,Y) :- parent(X,Y), ancestor(Y,Z).`
- `ancestor(X,Z) :- parent(X,Y), ancestor(X,Y).`
- `?-ancestor(X, john).`

- This program is just syntactic sugar for FOL:
  
  \[
  parent(bill, mary) \land parent(mary, john) \land \forall x, y. parent(x, y) \rightarrow ancestor(x, y) \land 
  (\forall x, y, z. parent(x, y) \land parent(y, z) \rightarrow ancestor(x, z)) \land 
  (\exists x. ancestor(x, john))
  \]

- Thus, if this formula is satisfiable, there is someone in our database who is John’s ancestor

### Compactness of First-Order Logic

- Another important property of FOL is compactness.
- A logic is called compact if an infinite set of sentences \( \Gamma \) is satisfiable iff every finite subset of \( \Gamma \) is satisfiable.
- Theorem (due to Gödel): First-order logic is compact.
- Proof of compactness of FOL follows from the completeness of proof rules.

### 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.

### Datalog and Logic Programming Languages

- A Datalog interpreter is nothing more than a solver for Bernays-Schönfinkel fragment of FOL
- Since this fragment is decidable, Datalog programs always terminate
- In general, interpreters for all logic programming languages decide satisfiability in FOL or a fragment
- A popular logic programming language is Prolog
- Unlike Datalog, it is based on full FOL, so Prolog programs may not terminate

### 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. \( \square \)

### 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 \( \langle t_0, t_n \rangle \in T \) iff there exists some sequence \( t_0, t_1, \ldots, t_n \) such that \( \langle t_i, t_{i+1} \rangle \in p \)
“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 $U = \{a, b\}$, $p$ 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

- $\Psi^n(a, b)$ encode the proposition: there is no path of length $n$ from $a$ to $b$.
- In particular, $\Psi^1 = \neg p(a, b)$
- Similarly, $\Psi^n = \neg \exists x_1, \ldots, x_{n-1}. (p(a, x_1) \land p(x_1, x_2) \land \ldots \land p(x_{n-1}, b))$

Transitive Closure and FOL

- In fact, no matter how hard we try to correct this definition, we cannot express transitive closure in FOL.
- Will use compactness theorem to show that transitive closure is not expressible in FOL.
- Compactness: An infinite set of sentences $\Gamma$ is satisfiable iff every finite subset of $\Gamma$ is satisfiable.
- For contradiction, suppose transitive closure is expressible in first order logic.
- Let $\Gamma$ be a (possibly infinite) set of sentences expressing that $T$ is the transitive closure of $p$.

Proof II

- Recall: $\Gamma$ is a set of propositions encoding $T$ is transitive closure of $p$.
- Now, construct $\Gamma'$ as follows:
  $$\Gamma' = \Gamma \cup \{ \Psi^1, \Psi^2, \Psi^3, \ldots \}$$
- 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 $\Psi^1, \Psi^2, \ldots$ say 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.

Recall: Propositional Resolution

- Basis of first-order theorem provers: resolution
- Recall resolution in PL:

\[ C_1 : (l_1 \lor \ldots \lor l_k) \quad C_2 : (l'_1 \lor \ldots \lor l'_m) \]

- Propositional resolution: Deduction of a new clause \( C_3 \), called resolvent:

\[ C_3 : (l_1 \lor \ldots \lor l_k \lor l'_1 \lor \ldots \lor l'_m) \]

- First-order resolution is the same basic principle, but a little bit more involved

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?
- Can \( p(a) \) and \( p(b) \) be unified?
- We’ll write \( e \sigma \) to denote the application of substitution \( \sigma \) to expression \( e \)
- What is \( p(x)[x \mapsto a] \)?

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: Proof of four color theorem; results in group theory, ...

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
- Resolution, clausal form next lecture

Unification

- A substitution is a unifier for two expressions \( e \) and \( e' \) if \( e \sigma \) is syntactically identical to \( e' \sigma \)
- Two expressions \( e \) and \( e' \) are unifiable if they have a unifier; otherwise non-unifiable.
- Example: Are \( p(x, y) \) and \( p(a, v) \) unifiable?
- A unifier:
- Example 2: Are \( p(x, x) \) and \( p(a, b) \) unifiable?
- Example 3: Are \( p(x) \) and \( p(f(x)) \) unifiable?
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: \([x \mapsto a, y \mapsto b, v \mapsto b]\)
- Unifier 2: \([x \mapsto a, y \mapsto v]\)
- Unifier 3: \([x \mapsto a, y \mapsto f(b), v \mapsto f(b)]\)
- 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 \( \sigma \) and \( \delta \) is written \( \sigma \delta = \sigma' \)
- The composition \( \sigma \delta \) of substitutions \( \sigma \) and \( \delta \) is obtained by:
  1. applying \( \delta \) to the range of \( \sigma \)
  2. add to \( \sigma \) all mappings \( x \mapsto t \) from \( \delta \) where \( x \notin \text{dom}(\sigma) \).

Generality of Unifiers

- We prefer unifiers that are as general as possible.
- A unifier \( \sigma \) is at least as general as unifier \( \sigma' \) if there exists another substitution \( \delta \) such that \( \sigma \delta = \sigma' \)
- Intuition: \( \sigma \) more general than \( \sigma' \) if \( \sigma' \) can be obtained from \( \sigma \) through another substitution
- Which unifier is more general? \( \sigma = [x \mapsto a, y \mapsto v] \) or \( \sigma' = [x \mapsto a, y \mapsto f(c), v \mapsto f(c)] \)?
- Which unifier is more general? \( \sigma = [x \mapsto a, y \mapsto z] \) or \( \sigma' = [x \mapsto a, y \mapsto w] \)?

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 \( p(a, c) \).
- Is \([x \mapsto a, y \mapsto b, v \mapsto b]\) an mgu?
- Is \([x \mapsto a, y \mapsto v]\) an mgu?
- Is \([x \mapsto a, y \mapsto v, v \mapsto y]\) an mgu?

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 \( p(a, v) \)?
Algorithm to Compute MGU

- We’ll now give an algorithm to find most general unifiers.
- Function \texttt{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:
- Apply unifier to second arguments \( f(x) \) and \( f(a) \) (unchanged)
- Then, unify \( f(x) \) and \( f(a) \):
- Compose \([y \mapsto f(x)]\) and \([x \mapsto a]\)
- Final result:

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 \([h \ T] \).
  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 \( y \); result:
- Apply unifier to second arguments \( x \) and \( f(y) \):
- Now unify \( y \) and \( f(y) \):
- Thus \( p(x, x) \) and \( p(y, f(y)) \) not unifiable