9.2 Unification and Lifting

- Inference in propositional logic is obviously too inefficient
- Writing out all variable bindings seems to be futile
- When there is a substitution $\theta$, s.t. $p'_i(\theta) = p_i(\theta)$ for all $i$, where $p'_i$ and $p_i$ are atomic sentences as well as $q$, we can use the Generalized Modus Ponens (GMP):
  \[
  \frac{p'_1, \ldots, p'_n, \ (p_1 \land \ldots \land p_n \Rightarrow q)}{q(\theta)}
  \]
- For example, from the fact $\text{Student}(John)$ and sentences
  \[
  \forall x: \text{EagerToLearn}(x) \text{ and } \forall y: \text{Student}(y) \land \text{EagerToLearn}(y) \Rightarrow \text{Thesis}_2016(y)
  \]
  we can infer $\text{Thesis}_2016(John)$ because of the substitution $\{y/John, x/John\}$

- Generalized Modus Ponens is a sound inference rule
- Similarly as GMP can be lifted from propositional logic to first-order logic, also forward chaining, backward chaining, and the resolution algorithm can be lifted
- A key component of all first-order inference algorithms is unification
- The unification algorithm $\text{Unify}$ takes two sentences and returns a unifier for them if one exists:
  \[
  \text{Unify}(p, q) = \theta, \text{ s.t. } p(\theta) = q(\theta)
  \]
  otherwise unification fails
Unify( Knows(John,x),Knows(John,Jane) ) = \{ x/Jane \}

Unify( Knows(John,x),Knows(y,Bill) ) = \{ x/Bill,y/John \}

Unify( Knows(John,x),Knows(y,Mother(y)) ) =
\{ y/John,x/Mother(John) \}

Unify( Knows(John,x),Knows(x,Eliza) ) = fail

- The last unification fails because \( x \) cannot take on the values \( John \) and \( Eliza \) simultaneously
- Because variables are universally quantified, \( \text{Knows}(x,Eliza) \) means that everyone knows \( Eliza \)
- In that sense, we should be able to infer that \( John \) knows \( Eliza \)

The problem above arises only because the two sentences happen to use the same variable name
- The variable names of different sentences have no bearing (prior to unification) and one can standardize them apart
- There can be more than one unifier, which of them to return?
- The unification algorithm is required to return the (unique) most general unifier
- The fewer restrictions (bindings to constants) the unifier places, the more general it is

Unify( Knows(John,x),Knows(y,z) )
\{ y/John,z/x \}
\{ y/John,x/John,z/John \}
9.3 Forward Chaining

- As before, let us consider knowledge bases in Horn normal form
- A definite clause either is atomic or is an implication whose body is a conjunction of positive literals and whose head is a single positive literal

\[ \text{Student(John)} \]
\[ \text{EagerToLearn}(x) \]
\[ \text{Student}(y) \land \text{EagerToLearn}(y) \implies \text{Thesis}_2016(y) \]

- Unlike propositional literals, first-order literals can include variables
- The variables are assumed to be universally quantified

As in propositional logic we start from facts, and by applying Generalized Modus Ponens are able to do forward chaining inference
One needs to take care that a "new" fact is not just a renaming of a known fact

\[ \text{Likes}(x, \text{Candy}) \]
\[ \text{Likes}(y, \text{Candy}) \]

- Since every inference is just an application of Generalized Modus Ponens, forward chaining is a sound inference algorithm
- It is also complete in the sense that it answers every query whose answers are entailed by any knowledge base of definite clauses
Datalog

- In a Datalog knowledge base the definite clauses contain no function symbols at all
- In this case we can easily prove the completeness of inference
- Let in the knowledge base
  - $p$ be the number of predicates,
  - $k$ be the maximum arity of predicates (= the number of arguments), and
  - $n$ the number of constants

- There can be no more than $pn^k$ distinct ground facts
- So after this many iterations the algorithm must have reached a fixed point, where new inferences are not possible

- In Datalog a polynomial number of steps is enough to generate all entailments
- For general definite clauses we have to appeal to Herbrand’s theorem to establish that the algorithm will find a proof
- If the query has no answer (is not entailed by the KB), forward chaining may fail to terminate in some cases
- E.g., if the KB includes the Peano axioms, then forward chaining adds facts

  $\text{NatNum}(S(0)).$
  $\text{NatNum}(S(S(0))).$
  $\text{NatNum}(S(S(S(0))))$.  

- Entailment with definite clauses is semidecidable
9.4 Backward Chaining

- In predicate logic backward chaining explores the bodies of those rules whose head unifies with the goal.
- Each conjunct in the body recursively becomes a goal.
- When the goal unifies with a known fact – a clause with a head but no body – no new (sub)goals are added to the stack and the goal is solved.
- Depth-first search algorithm.
- The returned substitution is composed from the substitutions needed to solve all intermediate stages (subgoals).
- Inference in Prolog is based on backward chaining.

9.4.2 Logic programming

- Prolog, Alain Colmerauer 1972.
- Program = a knowledge base expressed as definite clauses.
- Queries to the knowledge base.
- Closed world assumption: we assume \( \neg \varphi \) to be true if sentence \( \varphi \) is not entailed by the knowledge base.
- Syntax:
  - Capital characters denote variables,
  - Small character stand for constants,
  - The head of the rule precedes the body,
  - Instead of implication use \( :\).:
  - Comma stand for conjunction,
  - Period ends a sentence.

\[
\text{thesis}_2016(X) :- \text{student}(X), \text{eager_to_learn}(X).
\]

- Prolog has a lot of syntactic sugar, e.g., for lists and arithmetics.
• Prolog program `append(X,Y,Z)` succeeds if list \( Z \) is the result of appending (catenating) lists \( X \) and \( Y \)
  
  \[
  \text{append}([],Y,Y).
  \]
  
  \[
  \text{append}([A|X],Y,[A|Z]) :- \text{append}(X,Y,Z).
  \]

• Query: `append([1],[2],Z)?`  
  \( Z=[1,2] \)

• We can also ask the query  
  `append(A,B,[1,2])?:`  
  Appending what two lists gives the list \([1,2]\)?

• As the answer we get back all possible substitutions
  
<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>([\ ])</td>
<td>([1,2])</td>
</tr>
<tr>
<td>([1])</td>
<td>([2])</td>
</tr>
<tr>
<td>([1,2])</td>
<td>([\ ])</td>
</tr>
</tbody>
</table>

• The clauses in a Prolog program are tried in the order in which they are written in the knowledge base
• Also the conjuncts in the body of the clause are examined in order (from left to right)
• There is a set of built-in functions for arithmetic, which need not be inferred further
  
  - E.g., \( X = 4+3 \rightarrow X=7 \)
• For instance I/O is taken care of using built-in predicates that have side effect when executed
• Negation as failure
  
  \[
  \text{alive}(X) :- \text{not dead}(X).
  \]
  
  “Everybody is alive if not provably dead”
The negation in Prolog does not correspond to the negation of logic (using the closed world assumption)

```
single_student(X):-
    not married(X), student(X).
student(peter).
married(john).
```

- By the closed world assumption, \( X = \text{peter} \) is a solution to the program
- The execution of the program, however, fails because when \( X = \text{john} \)
  the first predicate of the body fails
- If the conjuncts in the body were inverted, it would succeed

An equality goal succeeds if the two terms are unifiable
- E.g., \( X + Y = 2 + 3 \) \( \rightarrow \) \( X = 2, Y = 3 \)

Prolog omits some necessary checks in connection of variable bindings \( \rightarrow \) Inference is not sound
- These are seldom a problem
- Depth-first search can lead to infinite loops (= incomplete)
  ```
  path(X,Z):- path(X,Y), link(Y,Z).
  path(X,Z):- link(X,Z).
  ```
- Careful programming, however, lets us escape such problems
  ```
  path(X,Z):- link(X,Z).
  path(X,Z):- path(X,Y), link(Y,Z).
  ```
• Anonymous variable
  member(X,[X|_]).
  member(X,[_|Y]):- member(X,Y).

• Works just fine
  member(d,[a,b,c,d,e,f,g])?
    yes
  member(2,[3,a,4,f])?
    no

• But queries
  member(a,X)?
  member(a,[a,b,r,a,c,a,d,a,b,r,a])?
  do not necessarily give the intended answers

• We can explicitly prune the execution of Prolog programs by cutting

• Negation using cut
  not X:- X, !, fail.
  not X.

• fail causes the program to fail
• At the point of a cut all bindings that have been made since starting to examine the rule are fixed
• For the conjuncts in the body preceding the cut, no new solutions are searched for
• Neither does one examine other rules having the same head
Prolog may come up with the same answer through several inference paths

Then the same answer is returned more than once

\[
\text{minimum}(X,Y,X) : \geq X, Y.
\]

\[
\text{minimum}(X,Y,Y) : \geq X, Y.
\]

Both rules yield the same answer for the query \text{minimum}(2,2,M)?

One must be careful in using cut for optimizing inference

\[
\text{minimum}(X,Y,X) : \geq X, Y!.
\]

\[
\text{minimum}(X,Y,Y).
\]

This program is erroneous, for instance \text{minimum}(2,8,8) holds according to it.

The key question of Prolog and logic programming obviously is efficiency of execution

Prolog implementations use a wide variety of enhancement techniques

For example, instead of generating all possible solutions for a subgoal before examining the next subgoal, a Prolog interpreter is content (so far) with just one

Similarly variable binding is at each instant unique; only when the search runs into a dead end, can \textit{backing up} to a choice point lead to unbinding of variables

A stack of history, called the \textit{trail}, needs to be maintained to keep track of all variable bindings
9.5 Resolution

- Kurt Gödel’s **completeness theorem** (1930) for first-order logic: any entailed sentence has a finite proof
  \[ T \models \varphi \iff T \vdash \varphi \]

- It was not until Robinson’s (1965) resolution algorithm that a practical proof procedure was found.
- Gödel’s more famous result is the **incompleteness theorem**:
  - a logical system that includes the principle of induction, is necessary incomplete
  - There are sentences that are entailed, but have no finite proof
  - This holds in particular for number theory, which thus cannot be axiomatized

For resolution, we need to convert the sentences to CNF
- E.g., “Everyone who loves all animals is loved by someone”

\[
\forall x: [\forall y: Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y: Loves(y,x)].
\]

- Eliminate implications
  \[
  \forall x: [\neg \forall y: \neg Animal(y) \lor Loves(x,y)] \lor [\exists y: Loves(y,x)].
  \]

- Move negation inwards
  \[
  \forall x: [\exists y: Animal(y) \land \neg Loves(x,y)] \lor [\exists y: Loves(y,x)].
  \]

- Standardize variables
  \[
  \forall x: [\exists y: Animal(y) \land \neg Loves(x,y)] \lor [\exists z: Loves(z,x)].
  \]
• Skolemization
  \[ \forall x: \left[ \left( Animal(x) \land \neg Loves(x,F(x)) \right) \lor Loves(G(z),x) \right]. \]

• Drop universal quantifiers
  \[ \left[ Animal(x) \land \neg Loves(x,F(x)) \right] \lor Loves(G(z),x). \]

• Distribute \( \lor \) over \( \land \)
  \[ \left[ Animal(x) \lor Loves(G(z),x) \right] \land \left[ \neg Loves(x,F(x)) \lor Loves(G(z),x) \right]. \]

• The end result is quite hard to comprehend, but it doesn’t matter, because the translation procedure is easily automated.

• First-order literals are complementary if one unifies with the negation of the other
• Thus the binary resolution rule is
  \[
  \frac{\ell_1 \lor \cdots \lor \ell_k, \ m}{m} \\
  \text{where } \text{Unify}(\ell_i, \neg m) = \theta
  \]

• For example, we can resolve
  \[ [Animal(x) \lor Loves(G(x),x)] \land [\neg Loves(u,v) \lor \neg Kills(u,v)] \]
  by eliminating the complementary literals
  \( Loves(G(x),x) \) and \( \neg Loves(u,v) \)
  with unifier \( \theta = \{ u/G(x), v/x \} \) to produce the resolvent clause
  \[ [Animal(F(x)) \lor \neg Kills(G(x),x)] \]
• Resolution is a complete inference rule also for predicate logic in the sense that we can check (not generate) all logical consequences of the knowledge base

• $KB \models \alpha$ is proved by showing that $\neg (KB \land \neg \alpha)$ is unsatisfiable through a proof by refutation

• Theorem provers (automated reasoners) accept full first-order logic, whereas most logic programming languages handle only Horn clauses.

• For example Prolog intertwines logic and control.

• In most theorem provers, the syntactic form chosen for the sentences does not affect the result.

• Application areas: verification of software and hardware.

• In mathematics theorem provers have a high standing nowadays: they have come up with novel mathematical results.

• For instance, in 1996 a version of well-known Otter was the first to prove (eight days of computation) that the axioms proposed by Herbert Robbins in 1933 really define Boolean algebra.