7.5.2 Proof by Resolution

- The inference rules covered so far are sound
- Combined with any complete search algorithm they also constitute a complete inference algorithm
- However, removing any one inference rule will lose the completeness of the algorithm

- Resolution is a single inference rule that yields a complete inference algorithm when coupled with any complete search algorithm

- Restricted to two-literal clauses, the resolution step is

\[ \ell_1 \lor \ell_2, \neg \ell_2 \lor \ell_3 \Rightarrow \ell_1 \lor \ell_3 \]

- When the agent in the wumpus-world moves to examine [1,2] for the first time, it perceives a stench, but no breeze:

\begin{align*}
R_{11} & : \neg B_{1,2} \\
R_{12} & : B_{1,2} \iff (P_{1,1} \lor P_{2,2} \lor P_{3,1})
\end{align*}

- By the same process that led to \( R_{10} \), we can derive the absence of pits in [2,2] and [1,3]:

\begin{align*}
R_{13} & : \neg P_{2,2} \\
R_{14} & : \neg P_{1,3}
\end{align*}

- Biconditional elimination to \( R_3 \), followed by Modus Ponens with \( R_5 \) yields \( R_{15} : P_{1,1} \lor P_{2,2} \lor P_{3,1} \)

- Resolution rule applied to \( R_{13} \) and \( R_{15} \) gives:

\[ R_{16} : P_{1,1} \lor P_{3,1} \]

- Similarly, the literal \( \neg P_{1,1} \) in \( R_1 \) resolves with the literal \( P_{1,1} \) in \( R_{16} \) to give

\[ R_{17} : P_{3,1} \]
Knowledge Base (3)

\[ R_i: \neg P_{1,1} \]
\[ R_2: B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \]
\[ R_3: B_{2,1} \iff (P_{1,1} \lor P_{2,2} \lor P_{3,1}) \]
\[ R_4: \neg B_{1,1} \]
\[ R_5: B_{2,1} \]
\[ R_6: (B_{1,1} \iff (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \iff B_{1,1}) \]
\[ R_7: ( (P_{1,2} \lor P_{2,1}) \iff B_{1,1} ) \]
\[ R_8: ( \neg B_{1,1} \iff \neg(P_{1,2} \lor P_{2,1}) ) \]
\[ R_9: \neg(P_{1,2} \lor P_{2,1}) \]
\[ R_{10}: \neg P_{1,2} \land \neg P_{2,1} \]
\[ R_{11}: \neg B_{1,2} \]
\[ R_{12}: B_{1,2} \iff (P_{1,1} \lor P_{2,2} \lor P_{1,3}) \]
\[ R_{13}: \neg P_{2,2} \]
\[ R_{14}: \neg P_{1,3} \]
\[ R_{15}: P_{1,1} \lor P_{2,2} \lor P_{3,1} \]
\[ R_{16}: P_{1,1} \lor P_{3,1} \]
\[ R_{17}: P_{3,1} \]

Above we applied the unit resolution inference rule:

\[ \ell_1 \lor \ldots \lor \ell_k, m \]
\[ \ell_1 \lor \ldots \lor \ell_k \lor \ell_k \lor \ldots \lor \ell_k \]

where \( \ell_i \) and \( m \) are complementary literals (one is the negation of the other)

- In the general form the resolution rule allows to have an arbitrary number of complementary literals
- **Factoring:** one must remove duplicate literals from the resulting clause

\[ A \lor B, A \lor \neg B \]
\[ A \lor B \]
\[ A \lor B \]
\[ A \]
Conjunctive normal form

- Resolution is a sound inference rule, and it is also complete.
- Though, given that \( A \) is true, we cannot use resolution to automatically generate the consequence \( A \lor B \).
- However, we can use resolution to answer the question whether \( A \lor B \) is true.
- To show that \( KB \models \alpha \), we show that \( (KB \land \lnot \alpha) \) is unsatisfiable.
- First \( (KB \land \lnot \alpha) \) is converted into conjunctive normal form (CNF) and the resolution rule is applied to the resulting clauses.
- A CNF formula is a conjunction of disjunctions of literals (clauses)
  \[(\ell_{1,1} \lor \ldots \lor \ell_{1,k}) \land \ldots \land (\ell_{n,1} \lor \ldots \lor \ell_{n,k})\]
- Every sentence of propositional logic is logically equivalent to a CNF formula.

Let us convert the sentence \( R_2: B_{1,1} \iff (P_{1,2} \lor P_{2,1}) \) into CNF.
- Eliminate \( \iff \):
  \[(B_{1,1} \iff (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \iff B_{1,1})\]
- Eliminate \( \iff \):
  \[(-B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (-P_{1,2} \lor P_{2,1}) \lor B_{1,1})\]
- Move \( \lnot \) inwards:
  \[(-B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land ((-P_{1,2} \land \lnot P_{2,1}) \lor B_{1,1})\]
- Apply distributivity law to yield \( R_2 \) finally in CNF:
  \[(-B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (-P_{1,2} \lor B_{1,1}) \land (-P_{2,1} \lor B_{1,1})\]
A resolution algorithm

- First \((KB \land \neg \alpha)\) is converted into CNF and the resolution rule is applied to the resulting clauses
- Each pair that contains complementary literals is resolved to produce a new clause, which is added to the set if it not already present
- Eventually,
  - there are no new clauses that can be added, in which case the KB does not entail \(\alpha\), or
  - an application of the resolution rule derives the empty clause \((= F)\), in which case \(KB \models \alpha\)
- Resolution rule applied to a contradiction \(P \land \neg P\) yields the empty clause

7.5.3 Horn clauses and definite clauses

- In real-world knowledge bases it is often enough to use implication rules, like \((L_{1,1} \land \text{Breeze}) \Rightarrow B_{1,1}\)
- The premise of the rule, which is a conjunction of positive literals, is called the body of the clause
- Also the conclusion is non-negated; it is called the head of the clause
- If a Horn clause has no body at all, it is called a definite clause or a fact
- The Horn clause \(P_1 \land \ldots \land P_n \Rightarrow Q\) is logically equivalent with disjunction \((\neg P_1 \lor \ldots \lor \neg P_n \lor Q)\)
- Thus, one allows that at most one of the literals is a positive proposition
7.5.4 Forward and backward chaining

- Inference with Horn clauses can be done through the *forward chaining* or *backward chaining* algorithm.
- Both directions for inference are very natural.
- Deciding entailment with Horn clauses can be done in time that is linear in the size of the knowledge base.
- Forward chaining (*data-driven reasoning*) examines the body of which rules is satisfied by the known facts and adds their heads as new facts to the knowledge base.
- Every inference is essentially an application of Modus Ponens.
- This process continues until the given query \( q \) is added or until no further inferences can be made.

AND-OR graph
• Forward chaining is a sound and also a complete inference algorithm

• Backward chaining is a form of goal-directed reasoning
• We aim at showing the truth of a given query $q$ by considering those implications in the knowledge base that conclude $q$ (have it as the head)
• By backward chaining one examines the whether all the premises of one of those implications can be proved true
• Backward chaining touches only relevant facts, while forward chaining blindly produces all facts
• One can limit forward chaining to the generation of facts that are likely to be relevant to queries that will be solved by backward chaining

7.7 Agents Based on Propositional Logic

• Already in our extremely simple wumpus-world it turns out that using propositional logic as the knowledge representation language suffers from serious drawbacks
• To initialize the knowledge base with the rules of the game, we have to give for each square [x,y] the rule about perceiving a breeze (and the corresponding rule for perceiving a stench)
  \[ B_{x,y} \equiv (P_{x,y+1} \lor P_{x,y-1} \lor P_{x+1,y} \lor P_{x-1,y}) \]
• There is at least one wumpus:
  \[ W_{1,1} \lor W_{1,2} \lor \ldots \lor W_{4,3} \lor W_{4,4} \]
  and, on the other hand, at most one wumpus, which can be expressed by sentences such as
  \[ \neg W_{1,1} \lor \neg W_{1,2} \]
• When we add rules \( \neg P_{1,1} \) and \( \neg W_{1,1} \) to the KB, the simple \( 4 \times 4 \) world has required in total \( 2 + 2 \cdot 16 + 1 \cdot 120 = 155 \) initial sentences containing 64 symbols

• Model checking should enumerate \( 2^{64} \approx 1.8 \times 10^{19} \) possible models

• More efficient inference algorithms, though, can take advantage of propositional knowledge representation

• In addition, the KB does not yet contain all the rules of the game (e.g., arrow, gold, walls) and no knowledge of the possible actions of the agent (direction the agent is facing, taking steps), so it cannot yet be used to choose actions

• In addition to being inefficient, propositional logic is also an unsatisfactory choice

8 FIRST-ORDER LOGIC

\[
\text{Sentence} \rightarrow \text{AtomicSentence} \mid (\text{Sentence Connective Sentence})
\]

\[
\begin{align*}
\rightarrow \mid \text{Quantifier VariableList: Sentence} \mid \neg \text{Sentence} \\
\text{AtomicSentence} \rightarrow \text{Predicate} \mid \text{Predicate(TermList)} \\
\text{Term} = \text{Term} \\
\text{TermList} \rightarrow \text{Term} \mid \text{Term, TermList} \\
\text{Term} \rightarrow \text{Function(TermList)} \mid \text{Constant} \mid \text{Variable} \\
\text{Connective} \rightarrow \land \mid \lor \mid \Rightarrow \mid \Leftrightarrow \\
\text{Quantifier} \rightarrow \forall \mid \exists \\
\text{VariableList} \rightarrow \text{Variable} \mid \text{Variable, VariableList} \\
\text{Constant} \rightarrow A \mid X_i \mid \text{John} \mid \text{Mary} \mid \ldots \\
\text{Variable} \rightarrow a \mid x \mid s \mid \ldots \\
\text{Predicate} \rightarrow T \mid F \mid \text{Before} \mid \text{HasColor} \mid \text{Raining} \mid \ldots \\
\text{Function} \rightarrow \text{Mother} \mid \text{LeftLeg} \mid \ldots
\end{align*}
\]
In the realm of first-order predicate logic there are objects, which have properties, and between which there are relations.

Constant symbols stand for objects, predicate symbols stand for relations, that is, relations of arity \( n \),

\[
\text{Mother}(\text{John}, \text{Mary}),
\]
and properties are unary relations,

\[
\text{Student}(\text{John})
\]

A function is a total mapping that associates one single value to the ordered collection of its arguments.

Quantifiers let us express properties of entire collections of objects, instead of enumerating the objects by name.

We have the possibility to quantify objects universally and existentially:

\[
\forall y: \text{Student}(y) \Rightarrow \text{EagerToLearn}(y)
\]

\[
\exists x: \text{Father}(\text{John}) = x
\]

To determine the semantics for the syntactic notions (constants, predicates, and functions) need to be bound with an interpretation to corresponding (real-)world objects.

The world together with the interpretation of the syntax constitutes the model in predicate logic.
• Term $f(t_1, ..., t_n)$ refers to the object that is the value of function $F$ applied to objects $d_1, ..., d_n$, where $F$ is the interpretation of $f$ and $d_1, ..., d_n$ are the objects that the argument terms $t_1, ..., t_n$ refer to.

• Atomic sentence $P(t_1, ..., t_n)$ is true if the relation referred to by the predicate symbol $P$ holds among the objects referred to by the arguments $t_1, ..., t_n$.

• The semantics of sentences formed with logical connectives is identical to that in propositional logic.

• The sentence $\exists x: \varphi(x)$ is true in a given model under a given interpretation if sentence $\varphi$ is true by assigning some object as the interpretation of variable $x$.

• The sentence $\forall x: \varphi(x)$ is true in a given model under a given interpretation if sentence $\varphi$ is true by assigning any object as the interpretation of variable $x$.

• Because $\forall$ is really a conjunction over the universe of objects and $\exists$ is a disjunction, they obey de Morgan’s rules:

  $\forall x: \neg \varphi(x) \iff \exists x: \varphi(x)$

  $\neg \forall x: \varphi(x) \iff \forall x: \neg \varphi(x)$

  $\exists x: \neg \varphi(x) \iff \neg \exists x: \varphi(x)$

  $\neg \exists x: \varphi(x) \iff \forall x: \neg \varphi(x)$

• Equality $=$ is a special relation, whose interpretation cannot be changed.

• Terms $t_1$ and $t_2$ have this relation if and only if they refer to the same object.
The kinship domain

\[ \forall m, c: \text{Mother}(c) = m \Leftrightarrow \text{Female}(m) \land \text{Parent}(m, c). \]

\[ \forall w, h: \text{Husband}(h, w) \Leftrightarrow \text{Male}(h) \land \text{Spouse}(h, w). \]

\[ \forall x: \text{Male}(x) \Leftrightarrow \neg \text{Female}(x). \]

\[ \forall p, c: \text{Parent}(p, c) \Leftrightarrow \text{Child}(c, p). \]

\[ \forall g, c: \text{Grandparent}(g, c) \Leftrightarrow \exists p: \text{Parent}(g, p) \land \text{Parent}(p, c). \]

\[ \forall x, y: \text{Sibling}(x, y) \Leftrightarrow x \neq y \land \exists p: \text{Parent}(p, x) \land \text{Parent}(p, y). \]

Peano axioms

- Define natural numbers and addition using one constant symbol 0 and a successor function S
  \[ \text{NatNum}(0). \]
  \[ \forall n: \text{NatNum}(n) \Rightarrow \text{NatNum}(S(n)). \]

- So the natural numbers are 0, S(0), S(S(0)), ...

- Axioms to constrain the successor function:
  \[ \forall n: 0 \neq S(n). \]
  \[ \forall m, n: m \neq n \Rightarrow S(m) \neq S(n). \]

- Addition in terms of the successor function:
  \[ \forall m: \text{NatNum}(m) \Rightarrow +(m, 0) = m. \]
  \[ \forall m, n: \text{NatNum}(m) \land \text{NatNum}(n) \Rightarrow +(S(m), n) = S(+m, n)). \]
Axioms for sets

- To define sets we use:
  - Constant symbol $\emptyset$, which refers to the empty set
  - Unary predicate $\text{Set}$, which is true of sets
  - Binary predicates $x \in s$ and $s_1 \subseteq s_2$
  - Binary function are $s_1 \cup s_2$, $s_1 \cap s_2$, and $\{ x | s \}$, which refers to the set resulting from adjoining element $x$ to set $s$

- The only sets are the empty set and those made by adjoining something to a set:
  $$\forall s: \text{Set}(s) \iff s = \emptyset \lor \exists x, s_2: \text{Set}(s_2) \land s = \{ x | s_2 \}.$$

- The empty set has no elements adjoined into it:
  $$\neg \exists x, s: \{ x | s \} = \emptyset.$$

- Adjoining an element already in the set has no effect:
  $$\forall x, s: x \in s \iff s = \{ x | s \}.$$

- The only members of a set are those elements that were adjoined into it:
  $$\forall x, s: x \in s \iff [\exists y, s_2: (s = \{ y | s_2 \} \land (x = y \lor x \in s_2))].$$

- Set inclusion:
  $$\forall s_1, s_2: s_1 \subseteq s_2 \iff (\forall x: x \in s_1 \Rightarrow x \in s_2).$$

- Two sets are equal iff each is a subset of the other:
  $$\forall s_1, s_2: (s_1 = s_2) \iff (s_1 \subseteq s_2 \land s_2 \subseteq s_1).$$

\begin{align*}
\forall x, s_1, s_2: & \quad x \in (s_1 \cap s_2) \iff (x \in s_1 \land x \in s_2), \\
\forall x, s_1, s_2: & \quad x \in (s_1 \cup s_2) \iff (x \in s_1 \lor x \in s_2).
\end{align*}
9 INference IN FIRST-ORDER LOGIC

- By eliminating the quantifiers from the sentences of predicate logic we reduce them to sentences of proposition logic and can turn to familiar inference rules.
- We may substitute the variable \( v \) in an universally quantified sentence \( \forall v : a \) with any ground term, a term without variables.
- **Substitution** \( \theta \) is a binding list – set of variable/term pairs – in which each variable is given the ground term replacing it.
- Let \( a(\theta) \) denote the result of applying the substitution \( \theta \) to the sentence \( a \).
- Universal instantiation to eliminate the quantifier is the inference:
  \[
  \frac{\forall v : a}{a((v/g)),}
  \]
  for any variable \( v \) and ground term \( g \).

- In existential instantiation the variable \( v \) is replaced by a Skolem constant \( k \) a symbol that does not appear elsewhere in the KB:
  \[
  \frac{\exists v : a}{a((v/k))}
  \]
- E.g., from the sentence \( \exists x : \text{Father}(John) = x \) we can infer the instantiation \( \text{Father}(John) = F_1 \), where \( F_1 \) is a new constant.
- Eliminating existential quantifiers by replacing the variables with Skolem constants and universal quantifiers with the set of all possible instantiations turns the KB essentially propositional.
- Ground atomic sentences such as \( \text{Student}(John) \) and \( \text{Mother}(John, Mary) \) must be viewed as proposition symbols.
- Therefore, we can apply any complete propositional inference algorithm to obtain first-order conclusions.
• When the KB includes a function symbol, the set of possible ground term substitutions is infinite
  \[ \text{Father(Father(Father(John)))} \]
• By the famous theorem due to Herbrand (1930) if a sentence is entailed by the original, first-order KB, then there is a proof involving just a finite subset of the propositionalized knowledge base
• Thus, nested functions can be handled in the order of increasing depth without losing the possibility to prove any entailed sentence
• Inference is hence complete
• Analogy with the halting problem for Turingin machines however shows that the problem is undecidable
  More exactly: the problem is semidecidable, the sketched approach comes up with a proof for entailed sentences

9.2 Unification and Lifting

• Inference in propositional logic is obviously too inefficient
• Writing out 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):
  \[ p'_1, \ldots, p'_n, (p_1 \land \ldots \land p_n \land q) \Rightarrow q(\theta) \]
• For example, from the fact \( \text{Student}(John) \) and sentences
  \( \forall x: \text{EagerToLearn}(x) \) and
  \( \forall y: \text{Student}(y) \land \text{EagerToLearn}(y) \Rightarrow \text{Thesis}_2011(y) \)
we can infer \( \text{Thesis}_2011(John) \) 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 Unify takes two sentences and returns a unifier for them if one exists:
\[
\text{Unify}(p, q) = \emptyset, \text{ s.t. } p(\emptyset) = q(\emptyset)
\]
otherwise unification fails

\[
\text{Unify( Knows(John, x), Knows(John, Jane) ) = } \{ x/Jane \}
\]
\[
\text{Unify( Knows(John, x), Knows(y, Bill) )= } \{ x/Bill, y/John \}
\]
\[
\text{Unify( Knows(John, x), Knows(y, Mother(y)) ) = }
\{ y/John, x/Mother(John) \}
\]
\[
\text{Unify( Knows(John, x), Knows(x, Eliza) ) = fail}
\]

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.

\[
\text{Unify( Knows(John, x), Knows(y, z) )}
\begin{align*}
\{ y/John, z/x \} \\
\{ y/John, x/John, z/John \}
\end{align*}
\]