

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 $A \vee B$
- However, we can use resolution to answer the question whether $A \vee B$ is true
- To show that $KB \models \alpha$, we show that $(KB \wedge \neg\alpha)$ is unsatisfiable
- First $(KB \wedge \neg\alpha)$ 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)

$$(l_{1,1} \vee \dots \vee l_{1,k}) \wedge \dots \wedge (l_{n,1} \vee \dots \vee l_{n,k})$$
- Every sentence of propositional logic is logically equivalent to a CNF formula



- Let us convert the sentence $R_2: B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ into CNF
- Eliminate \Leftrightarrow :

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$$
- Eliminate \Rightarrow :

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$$
- Move \neg inwards:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$$
- Apply distributivity law to yield R_2 finally in CNF:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$



A resolution algorithm

- First $(KB \wedge \neg\alpha)$ 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 α , or
 - an application of the resolution rule derives the empty clause (= F), in which case $KB \models \alpha$
- Resolution rule applied to a contradiction $P \wedge \neg P$ 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 $(L_{1,1} \wedge \text{Breeze}) \Rightarrow B_{1,1}$
- 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 $P_1 \wedge \dots \wedge P_n \Rightarrow Q$ is logically equivalent with disjunction $(\neg P_1 \vee \dots \vee \neg P_n \vee Q)$
- Thus, one allows that at most one of the literals is a positive proposition



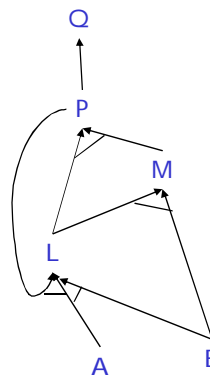
7.5.4 Forward and backward chaining

- 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

$P \Rightarrow Q$
 $L \wedge M \Rightarrow P$
 $B \wedge L \Rightarrow M$
 $A \wedge P \Rightarrow L$
 $A \wedge B \Rightarrow L$
 A
 B





- 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)

$$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$$

- There is at least one wumpus:

$$W_{1,1} \vee W_{1,2} \vee \dots \vee W_{4,3} \vee W_{4,4}$$

and, on the other hand, at most one wumpus, which can be expressed by sentences such as

$$\neg W_{1,1} \vee \neg W_{1,2}$$





- 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 ,
 $Mother(John, Mary)$,
 and properties are unary relations,
 $Student(John)$
- 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
 $\forall y: Student(y) \Rightarrow EagerToLearn(y)$
 $\exists x: Father(John) = x$
- 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 $f(t_1, \dots, t_n)$ refers to the object that is the value of function F applied to objects d_1, \dots, d_n , where F is the interpretation of f and d_1, \dots, d_n are the objects that the argument terms t_1, \dots, t_n refer to
- Atomic sentence $P(t_1, \dots, t_n)$ is true if the relation referred to by the predicate symbol P holds among the objects referred to by the arguments t_1, \dots, 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 φ 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 φ 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

$$\begin{aligned} \forall x: \neg\varphi(x) &\Leftrightarrow \neg\exists x: \varphi(x) \\ \neg\forall x: \varphi(x) &\Leftrightarrow \exists x: \neg\varphi(x) \\ \forall x: \varphi(x) &\Leftrightarrow \neg\exists x: \neg\varphi(x) \\ \neg\forall x: \neg\varphi(x) &\Leftrightarrow \exists x: \varphi(x) \end{aligned}$$

- 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

$$\forall m, c: \text{Mother}(c) = m \Leftrightarrow \text{Female}(m) \wedge \text{Parent}(m, c).$$

$$\forall w, h: \text{Husband}(h, w) \Leftrightarrow \text{Male}(h) \wedge \text{Spouse}(h, w).$$

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

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

$$\forall g, c: \text{Grandparent}(g, c) \Leftrightarrow \exists p: \text{Parent}(g, p) \wedge \text{Parent}(p, c).$$

$$\forall x, y: \text{Sibling}(x, y) \Leftrightarrow x \neq y \wedge \exists p: \text{Parent}(p, x) \wedge \text{Parent}(p, y).$$



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 $0, S(0), S(S(0)), \dots$
- 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:

$$\forall m: \text{NatNum}(m) \Rightarrow +(m, 0) = m.$$

$$\forall m, n: \text{NatNum}(m) \wedge \text{NatNum}(n) \Rightarrow +(S(m), n) = S(+(m, n))$$



Axioms for sets

- To define sets we use:
 - Constant symbol \emptyset , which refers to the empty set
 - Unary predicate 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:

$$\forall s: \text{Set}(s) \Leftrightarrow s = \emptyset \vee \exists x, s_2: \text{Set}(s_2) \wedge s = \{x|s_2\}.$$
- 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:

$$\forall x, s: x \in s \Leftrightarrow s = \{x|s\}.$$
- The only members of a set are those elements that were adjoined into it:

$$\forall x, s: x \in s \Leftrightarrow [\exists y, s_2: (s = \{y|s_2\} \wedge (x = y \vee x \in s_2))].$$
- Set inclusion:

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

$$\forall s_1, s_2: (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1).$$

$$\forall x, s_1, s_2: x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2).$$

$$\forall x, s_1, s_2: x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2).$$



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 $\forall v: \alpha$ with any *ground term*, a term without variables
- **Substitution** θ is a binding list – set of variable/term pairs – in which each variable is given the ground term replacing it
- Let $\alpha(\theta)$ denote the result of applying the substitution θ to the sentence α
- Universal instantiation to eliminate the quantifier is the inference

$$\frac{\forall v: \alpha}{\alpha(\{v/g\})}$$

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

$$\frac{\exists v: \alpha}{\alpha(\{v/k\})}$$

- E.g., from the sentence $\exists x: \text{Father}(\text{John}) = x$ we can infer the *instantiation* $\text{Father}(\text{John}) = F_1$, 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 $\text{Student}(\text{John})$ and $\text{Mother}(\text{John}, \text{Mary})$ 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

$Father(Father(Father(John)))$

- 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 Turing 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 θ 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):

$$\frac{p'_1, \dots, p'_n, (p_1 \wedge \dots \wedge p_n \Rightarrow q)}{q(\theta)}$$

- For example, from the fact $Student(John)$ and sentences
 $\forall x: EagerToLearn(x)$ and
 $\forall y: Student(y) \wedge EagerToLearn(y) \Rightarrow Thesis_2012(y)$
 we can infer $Thesis_2012(John)$ 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:

$$\text{Unify}(p, q) = \theta, \text{ s.t. } p(\theta) = q(\theta)$$
 otherwise unification **fails**



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

- The last unification fails because **x** cannot take on the values **John** and **Eliza** simultaneously
- Because variables are universally quantified, **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

Student(John)
 EagerToLearn(x)
 $Student(y) \wedge EagerToLearn(y) \Rightarrow Thesis_2012(y)$

- 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

Likes(x, Candy)

Likes(y, 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

