Constraint satisfaction problems

Factor graph - (aka Markov random field) a set of variables $X = X_1, \dots, X_n$ where $X_i \in \text{Domain}_i$ and factors $f_1, \dots, f_m$, with each $f_j(X) \geq 0$. Each factor is implemented as checking a solution rather than computing the solution.

Domain - possible values to be assigned to a variable.

Scope of a factor f_j - the set of variables f_j depends on. **Arity -** the size of this set. "Unary factors" (arity 1); "Binary factors" (arity 2). "Constraints" (factors that return 0 or 1).

Assignment weight - each assignment $x = (x_1, \dots, x_n)$ yields a $\text{Weight}(x)$ defined as being the product of all factors f_j applied to that assignment.

$\boxed{\text{Weight}(x) = \Pi_{j=q}^{m} f_j(x)}$ (x in its entirety is passed in to each f_j for simplicity of this notation, though in reality only a subset of x would be needed for f_j)

CSP - a factor graph where all factors are binary.

$\boxed{\text{For } j = 1, \cdots,\ f_j(x) \in \{0, 1\}}$ (the constraint j with assignment x is said to be satisfied iff $f_j(x) = 1$.)

Consistent assignment x of a CSP - iff $\text{Weight}(x) = 1$ (i.e., all constrains are satisfied.)

Dependent factors $D(x, X_i)$ - a set of factors depending on X_i but not on unassigned variables.

Backtracking search - find maximum weight assignment of a factor graph.

Backtrack($x, w,$ Domains)

1. choose an unassigned **variable** X_i (MCV)
2. order **values** of X_i's Domain (LCV)
3. for each value v in the order:
   (a) $\delta \leftarrow \Pi_{f_j \in D(x, X_i)} f_j(x \cup \{X_i : v\})$
   (b) if $\delta = 0$: continue
   (c) Domains$'$ ← Domains via **lookahead** (forward checking)
   (d) if $\text{Domains}_i'$ is empty: continue
   (e) **Backtrack**($x \cup \{X_i : v\}, w\delta,$ **Domains$'$**)

- Strategy: extends partial assignments
- Optimality: exact
- Time: exponential

Forward checking - one-step lookahead heuristic that preemptively removes inconsistent values from the domains of neighboring variables.

- After assigning a variable X_i, it eliminates inconsistent values from the domains of all its neighbors.
- If any of these domains become empty, stop the local backtracking search.
- if we unassign a variable X_i, have to restore the domain of its neighbors.

Most constrained variable - selects the next unassigned variable that has the fewest consistent values: fail early, prune early. Useful when some factors are constraints.

Least constrained value - assigns the next value that yields the highest number of consistent values of neighboring variables: prefers the value that is most likely to work. Useful when all factors are constraints. If some factors get arbitrary non-negative values, we need to try all consistent values anyway.

Arc consistency of variable X_i - w.r.t. X_j is enforced when for each $x_i \in \text{Domain}_i$, there exists $x_j \in \text{Domain}_j$ such that any factos between X_i and X_j is non-zero. Arc consistency is a ***definition***, not an algorithm: enforce all the way.

AC-3 - a multi-step lookahead heuristic that applies forward checking to all relevant variables. After a given assignment, it performs forward checking and then successively enforces arc consistency w.r.t. the neighbors of variables for which the domain change during the process. AC-3 only looks locally at the graph for nothing blatantly wrong; it can't detect when there are no consistent assignments.

Beam search - extends partial assignments of n variables of branching factor $b = |\text{Domain}|$ by exploring the K top paths at each step. The beam size $1 \geq K \geq b^n$ controls the tradeoff between efficiency and accuracy. Runtime is Linear to n: $O(n\underbrace{Kb\log(Kb)}_{\text{sorting top K}})$. $K = 1$: greedy search ($O(nb)$ time); $K \to +\infty$: BFS ($O(b^n)$ time).

- Strategy: extends partial assignments
- Optimality: approximate
  - Global optimality is only guaranteed by unbounded beam size
  - Increasing k from 1 to 2 doesn't guarantee increasing weight assignment
- Time: linear

Local search (iterated conditional modes) - modifies the assignment of a factor graph one variable at a time until convergence. At step i, assign to X_i the value v that maximizes the product of all factors connected to that variable.

- Initiualize x to a random complete assignment (not partial)
- Loop through $i = 1, \dots, n$ until convergence:
  - Compute weight of $x_v = x \cup X_i : v$ for each v
  - $x \leftarrow x_v$ with highest weight

ICM may get stuck in local optima; adding randomness may help.

- Strategy: modify complete assignments
- Optimality: approximate
- Time: linear

CSP tricks -

- If it's possible that a variable doesn't get an assignment, consider extending the domain to inclue a $\varnothing$ value.
- max weight assignment may be skewed towards arbitrarily high assignment if there exists unbounded non-negative factor value like a "preference". Bounding it to say 1 through 10 may help.

Markov Networks

CSPs	Markov networks
variables	random variables
weights	probabilities
max weight assignment	marginal probabilities

Capture the uncertainty over assignment using the language of probability.

Markov Network - a factor graph which defines a joint distribution over random variables $X = (X_1, \dots, X_n)$: $\boxed{\mathbb{P}(X = x) = \frac{\text{Weight}(x)}{Z}}$

$Z = \sum_{x'}$ **Weight**(x') **-** sum all the possible assignments' weights (normalization constant)

Marginal probability - the probability of when one particular variable X_i is assigned with a particular value v: sum $\mathbb{P}$ when $X_i = v$

$\boxed{\mathbb{P}(X_i = v) = \sum_{x : x_i = v} \mathbb{P}(X = x)}$

Gibbs sampling - approximately computes marginal probabilities. Follows the tempalte of local search: change one variable at a time. But unlike ICM, Gibbs is a randomized algo .

- Initialize x to a random complete assignment.
- Loop through $i = 1, \dots, n$ until convergence:
  - Set $x_i = v$ with probability $\mathbb{P}(X_i = v | \underbrace{X_{-i}}_{\text{all vars except} X_i} = x_{-i})$
  - Increment $\text{count}_i(x_i)$ (how often this assignment is encountered. can just track particular vars we're interested in.)
- Estimate $\hat{\mathbb{P}}(X_i = x_i) = \frac{\text{count}_i(x_i)}{\sum_v \text{count}_i(v)}$

ICM	Gibbs sampling
max weight assignment	marginal probabilities
choose best value	sample a value
converges to local optimum	marginals converge to correct answer

Bayesian Networks

$$\boxed{P(A|B,C) = \frac{P(B|A,C)P(A|C)}{P(B|C)}}$$

Explaining away - suppose two causes positively influence an effect. Conditioned on the effect, further conditioning on one causes reduces the probability of the other cause.

Bayesian network - a directed acyclic graph that specifies a joint distribution over random variables $X = (X_1, \dots, X_n)$ as a product of local conditional distributions, one for each node:

$$\boxed{\mathbb{P}(X_1 = x_1, \dots, X_n = x_n) = \prod_{i=1}^{n} p(x_i | x_{\text{Parents}(i)})}$$

Probabilistic program - randomizes variable assignment such that we can write down complex Bayesian networks that generates assignments without having to explicitly specify associated probabilities. Unlike normal classification (e.g., neural nets), Bayesian networks provide a different paradigm where we think about going from output to the input.

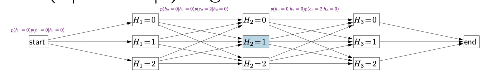
Probabilistic inference strategy - to compute the probability $P(Q|E = e)$ of query Q given evidence $E = e$:

1. Remove vars that aren't ancestors of the query Q or the evidence E by marginalization
2. Convert Bayesian network to factor graph
3. Condition on the evidence $E = e$
4. Remove nodes disconnected from the query Q by marginalization
5. Run probabilistic inference algorithm

Filtering question - asks for the distribution of some hidden variable H_i conditioned on only the evidence up until that point. Useful for real-time object tracking as the future can't be seen.

Smoothing question - asks for the distribution of some hidden variable H_i conditioned on on the evidence including the future. Useful when all the data have been collected and we want to retrospectively go and figure out what the hidden state H_i was.

Lattice representation - $O(n|\text{Domain}|)$ nodes and $O(n|\text{Domain}|^2)$ edges.


$p(h_2 = 0|h_1 = 0)p(e_2 = 2|h_2 = 0)$
$p(h_3 = 0|h_2 = 0)p(e_3 = 2|h_3 = 0)$
$p(h_1 = 0)p(e_1 = 0|h_1 = 0)$
start
$H_1 = 0$
$H_1 = 1$
$H_1 = 2$
$H_2 = 0$
$H_2 = 1$
$H_2 = 2$
$H_3 = 0$
$H_3 = 1$
$H_3 = 2$
end

- Edge $\boxed{\text{start}} \Rightarrow \boxed{H_1 = h_1}$ has weight $p(h_1)p(e_1 \mid h_1)$
- Edge $\boxed{H_{i-1} = h_{i-1}} \Rightarrow \boxed{H_i = h_i}$ has weight $p(h_i \mid h_{i-1})p(e_i \mid h_i)$
- Each path from $\boxed{\text{start}}$ to $\boxed{\text{end}}$ is an assignment with weight equal to the product of edge weights

Key: $\mathbb{P}(H_i = h_i \mid E = e)$ is the weighted fraction of paths through $\boxed{H_i = h_i}$

A path's weight is also the joint probability $P(H = h, E = e)$ (all variables assigned!)

Forward-backward algorithm - computes the exact value of $P(H = h_k|E = e)$ a smoothing query) for any $k \in \{1, \dots, L\}$ in the case of an HHM of size L. Running time: $O(n|\text{Domain}|^2)$.

1. for $i \in \{1, \dots, L\}$, compute $F_i(h_i) =$
$$\boxed{\begin{cases} p(h_1)p(e_1|h_1) & i = 1 \\ \sum_{h_{i-1}} F_{i-1}(h_{i-1})p(h_i|h_{i-1})p(e_i|h_i) & \text{otherwise} \end{cases}}$$
2. for $i \in \{L, \dots, 1\}$, compute $B_i(h_i) =$
$$\boxed{\begin{cases} 1 & \text{i =} \\ \sum_{h_{i+1}} B_{i+1}(h_{i+1})p(h_{i+1}|h_i)p(e_{i+1}|h_{i+1}) & \text{othe} \end{cases}}$$
3. for $i \in \{1, \dots, L\}$, compute $S_i(h_i) = \frac{F_i(h_i)B_i(h_i)}{\sum_{h_i} F_i(h_i)B_i(h_i)}$

with the convention $F_0 = B_{L+1}$. We get $\boxed{P(H = h_k|E = e) = S_k(h_k)}$

Particle (partial assignment) filtering - approximates the posterior density of state variables given the evidence of observation variables by keeping track of K particles at a time. Initialize $C \leftarrow [\{\}]$. For $t = 1, \dots, n$:

1. **proposal:** for each old particle $x_{t-1} \in C$, sample x from the transition probability distribution $p(x|x_{t-1})$ and add x to a set C'.
2. **weighting:** weight each x of the set C' by $w(x) = p(e_t|x)$ (e_t is the evidence observed at time t).
3. **resampling:** sample K elements from the set C' using the probability distribution induced by w and store them in C: these are current particles x_t.

Maximum likelihood - if we don't know the local conditional distributions, we can learn them using max likelihood.

$\boxed{\max_\theta \prod_{x \in D_{\text{train}}} p(X = x; \theta)}$ HMM parameters:

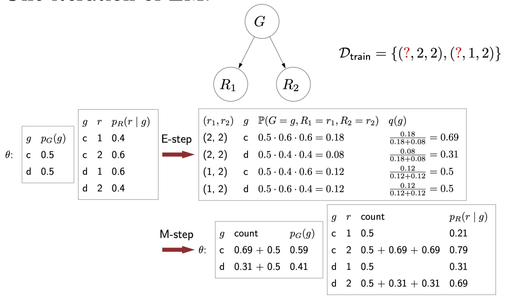
$\theta = (p_{\text{start}}, p_{\text{trans}}, p_{\text{emit}})$.

Laplace smoothing - for each distribution d and partial assignment $(x_{\text{Parents}(i)}, x_i)$, add λ (a constant) to count$_d(x_{\text{Parents}(i)}, x_i)$, then normalize to get probability estimates. Hallucinate λ occurrences of each local assignment. Larger $\lambda \Rightarrow$ more smoothing $\Rightarrow$ probabilities closer to uniform.

Expectation-Maximization (EM) - a parameter learning algorithm that estimates the parameter θ; H is hidden, $E = e$ is observed:

- Initialize θ randomly
- Repeat until convergence:
  - E-step:
    - Compute $\boxed{q(h) = P(H = h | E = e; \theta)}$
    - Create fully-observed weighted examples: (h, e) with weight $q(h)$
  - M-step: Maximum likelihood (count and normalize) on weighted examples to get θ

One iteration of EM:



g	$p_G(g)$
c	0.5
d	0.5

g	r	$p_R(r \mid g)$
c	1	0.4
c	2	0.6
d	1	0.6
d	2	0.4

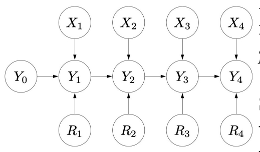
(r_1, r_2)	g	$\mathbb{P}(G = g, R_1 = r_1, R_2 = r_2)$	$q(g)$
(2, 2)	c	$0.5 \cdot 0.6 \cdot 0.6 = 0.18$	$\frac{0.18}{0.18+0.08} = 0.69$
(2, 2)	d	$0.5 \cdot 0.4 \cdot 0.4 = 0.08$	$\frac{0.08}{0.18+0.08} = 0.31$
(1, 2)	c	$0.5 \cdot 0.4 \cdot 0.6 = 0.12$	$\frac{0.12}{0.12+0.12} = 0.5$
(1, 2)	d	$0.5 \cdot 0.6 \cdot 0.4 = 0.12$	$\frac{0.12}{0.12+0.12} = 0.5$

g	count	$p_G(g)$
c	0.69 + 0.5	0.59
d	0.31 + 0.5	0.41

g	r	count	$p_R(r \mid g)$
c	1	0.5	0.21
c	2	0.5 + 0.69 + 0.69	0.79
d	1	0.5	0.31
d	2	0.5 + 0.31 + 0.31	0.69

Like K-means, EM converges to a local optima.

EM can handle partial data.



$Y_0 = 0$
X_i chosen uniformly from $\{1, 2, 3, 4\}$.
$p(R_i = 1) = (1 - \epsilon)$ (remember)
Suppose we observe that $Y_2 = 3$. What do we believe about X_2? First note that $Y_1 = X_1$ deterministically, which has a uniform distribution over $\{1, 2, 3, 4\}$. Next, let us write out the conditional distribution:
$P(X_2 = x_2 | Y_2 = 3) \propto P(X_2 = x_2, Y_2 = 3) = \sum_{y_1, r_2} p(y_1) p(r_2) p(x_2) p(y_2 = 3 | y_1, x_2, r_2)$ Now comute RHS for each value of x_2

x_2	$\mathbb{P}(X_2 = x_2, Y_2 = 3)$	$\mathbb{P}(X_2 = x_2 \mid Y_2 = 3)$
1	$\frac{1}{4} \cdot (1 - \epsilon) \cdot \frac{1}{4} \cdot 1$	$\frac{1-\epsilon}{2+2\epsilon}$
2	$\frac{1}{4} \cdot (1 - \epsilon) \cdot \frac{1}{4} \cdot 1$	$\frac{1-\epsilon}{2+2\epsilon}$
3	$1 \cdot \epsilon \cdot \frac{1}{4} \cdot 1$	$\frac{4\epsilon}{2+2\epsilon}$
4	0	0

For $X_2 \in \{1, 2\}$, we must have had remembered $(R_2 = 1)$, which forces $Y_1 = Y_2 - X_2$ (has probability $\frac{1}{4}$).

For $X_2 = 3$, we must have forgotten $(R_2 = 0)$, and Y_1 is free to be anything (has probability 1).

Logic Basics

Syntax - representation of a meaning: what are valid expressions in the language?

Semantics - what do these expressions mean?

Inference rules - what does a formula imply?

Propositional symbols - (atomic formulas): A, B, C

Model w - denotes an assignment of truth values to propositional symbols. Example: 3 propositional symbols yield $2^3 = 8$ possible models.

Interpretation function $\mathcal{I}(f, w)$ - outputs whether model w satisfies formula f:

$$\boxed{\mathcal{I}(f, w) = \begin{cases} 1 & w \text{ satisfies } f \\ 0 & \text{otherwise} \end{cases}}$$

Set of models $\mathcal{M}(f)$ - denotes the set of models w that satisfy formula f:

$$\boxed{\forall w \in \mathcal{M}(f), \mathcal{I}(f, w) = 1}$$

Knowledge base KB - a set of formulas representing their conjunction:

$\boxed{\mathcal{M}(KB) = \cap_{f \in KB} \mathcal{M}(f)}$ KB specifies constraints on the world. $\mathcal{M}(KB)$ is the set of all worlds satisfying those constraints. Adding more formulas to the KB shrinks the set of models (more constrained).

Entailment $KB \models f$ - KB entails f iff $\mathcal{M}(KB) \subseteq \mathcal{M}(f)$ f doesn't bring any new info.

Contradiction $KB \models \neg f$ - KB contradicts f iff $\mathcal{M}(KB) \cap \mathcal{M}(f) = \varnothing$ No model satisfies the constraints after adding f.

Contigency - $\varnothing \subsetneq \mathcal{M}(KB) \cap \mathcal{M}(f) \subsetneq \mathcal{M}(KB)$. f adds a non-trivial amount of info to KB and doesn't contradict KB.

Probabilistic interpretation - the probability that query f is evaluated to 1 can be seen as the proportion of models w of the KB that satisfy f:

$$\boxed{P(f|KB) = \frac{\sum_{w \in \mathcal{M}(KB) \cap \mathcal{M}(f)} P(W = w)}{\sum_{w \in \mathcal{M}(KB)} P(W = w)}}$$

Satisfiability - the KB is said to be satisfiable if at least one model w satisfies all its constraints, i.e. $\boxed{\mathcal{M}(KB) \neq \varnothing}$.

- Is $KB \cup \{\neg f\}$ satisfiable?
  - no: **entailment**
  - yes: Is $KB \cup \{f\}$ satisfiable?
    - no: **contradiction**
    - yes: **contingent**

Model checking - input: a KB and output: whether the KB is satisfiable or not.

Model checking	**CSP**
propositional symbol	variable
formula	constraint
model	assignment

Inference rule - of premises $f_1, \ldots, f_k$ and conclusion g is written: $\boxed{\frac{f_1, \ldots, f_k}{g}}$ where fs and g are formulas. Rules operate directly on syntax, not on semantics.

Modus ponens inference rule - for any propositional symbols p and q: $\boxed{\frac{p, p \to q}{q}}$. Example: $\frac{\text{Rain}, \text{Rain} \to \text{Wet}}{\text{Wet}}$

- Premises
  - It's raining
  - If it's raining, then it's wet
- Conclusion: therefore, it's wet

Forward inference - a model checking algorithm. Input: set of inference rules Rules. Repeat until no changes to KB:

- Choose set of formulas $f_1, \ldots, f_k \in KB$
- If matching rule $\frac{f_1, \ldots, f_k}{g}$ exists:
  - Add g to KB

Derivation $KB \vdash f$ - KB **derives/proves** f iff f eventually gets added to KB.

Soundness - a set of inference rules Rules is sound if: $\boxed{\{f : KB \vdash f\} \subseteq \{f : KB \models f\}}$.

- Inferred formulas are entailed by KB
- Can be checked one rule at a time
- *Nothing but the truth*

Examples: Modus ponens such as $\frac{\text{Rain}, \text{Rain} \to \text{Wet}}{\text{Wet}}$ is sound. $\frac{\text{Rain}, \text{Rain} \to \text{Wet}}{\text{Rain}}$ is not sound.

Completeness - a set of inference rules Rules is complete if: $\boxed{\{f : KB \vdash f\} \supseteq \{f : KB \models f\}}$.

- Formulas entailing KB are either already in the knowledge base or inferred from it
- *The whole truth*

Modus ponens is incomplete.

Propositional logic

Horn clause - is either:

- a definite clause: $\boxed{(p_1 \wedge \cdots \wedge p_k) \to q}$
- a goal clause: $\boxed{(p_1 \wedge \cdots \wedge p_k) \to \text{false}}$

Interpreted as the negation of the conjunction.

Modus ponens inference rule -

$\boxed{\frac{p_1, \ldots, p_k, (p_1 \wedge \cdots \wedge p_k) \to q}{q}}$ Modus ponens is **complete** w.r.t. Horn clauses:

- if we suppose that KB contains only Horn clauses and q is an entailed propositional symbol.
- then applying modus ponens will derive q.

Conjunctive normal form (CNF) - $\wedge$ (and) or $\vee$ (or-s). Equivalent: KB where each formula is a clause.

Convert into CNFs -

$f \to g$	$\neg f \vee g$
$f \leftrightarrow g$	$(f \to g) \wedge (g \to f)$
$\neg(f \wedge g)$	$\neg f \vee \neg g$
$\neg(f \vee g)$	$\neg f \wedge \neg g$
$f \vee (g \wedge h)$	$(f \vee g) \wedge (f \vee h)$

Use parentheses around $(f \to g)$ to "beat" $\wedge$

Resolution inference rule -

$\boxed{\frac{f_1 \vee \cdots \vee f_k \vee p, \neg p \vee g_1 \vee \cdots \vee g_m}{f_1 \vee \cdots \vee f_k \vee g_1 \vee \cdots \vee g_m}}$ Example:

$\frac{\text{Rain} \vee \text{Snow}, \neg \text{Snow} \vee \text{Traffic}}{\text{Rain} \vee \text{Traffic}}$

Resolution-based inference - recall $KB \models f \leftrightarrow KB \cup \{\neg f\}$ is unsatisfiable.

- Add $\neg f$ to KB
- Convert all formulas into CNF.
- Repeatedly apply resolution rule.
- Return entailment iff False is derived.

First-order logic

Terms - expressions refer to objects:

- Constant symbol: Alice
- Variable: x
- Function of terms: $sum(x, 3)$

Formulas - refer to truth values:

- Atomic formula (predicate applied to terms): Knows(Bob, Arith)
- Connectives applied to formulas
- Quantifiers applied to formulas
- see "every" thinks $\forall$ and $\to$:
  - Every student knows arithmetic: $\forall x \text{Student}(x) \to \text{Knows}(x, \text{arithmetic})$
- see "some" thinks $\exists$ and $\wedge$:
  - Some student knows arithmetic: $\exists x \text{Student}(x) \wedge \text{Knows}(x, \text{arithmetic})$
- $\neg \forall x P(x)$ equivalent to $\exists x \neg P(x)$

Model w - in first-order logic maps:

- constant symbols to objects: e.g. $w(\text{alice}) = o_1, w(\text{bob}) = o_2, w(\text{arithmetic}) = o_3$
- predicate symbols to tuple of objects: e.g. $w(\text{Knows}) = \{(o_1, o_3), (o_2, o_3), \ldots\}$

Two assumptions to simplify things:

- Unique names assumption: each object has **at most one** constant symbol.
- Domain closure: each object has **at least one** constant symbol.

Together: 1-to-1 mapping between constant symbols in syntax-land and objects in semantics-land.

Then: First-order logic is syntactic sugar for propositional logic → use any inference algorithm for propositional logic. **Propositionalization -** example:

$\text{Stu(A)} \wedge \text{Stu(A)}$	$\text{StuB} \wedge \text{StuB}$
$\forall x \text{Stu}(x) \to \text{Per}(x)$	$(\text{StuA} \to \text{PerA}) \wedge (\text{StuB} \to \text{PerB})$
$\exists x \text{Stu}(x) \wedge \text{Creative}(x)$	$(\text{StuA} \wedge \text{CreativeA}) \vee (\text{StuB} \to \text{CreativeB})$

Horn clause - $\boxed{\forall x_1, \ldots, \forall x_n, (a_1 \wedge \cdots \wedge a_k) \to b}$

$x_1, \ldots x_n$ are variables and $a_1, \ldots a_k, b$ are atomic formulas which contain those variables.

Substitution - a substitution θ maps variables to terms and Subst$[\theta, f]$ denotes the result of substitution θ on f.

Unification - takes two formulas f and g and returns a substitution θ which is the most general substitution θ that makes them equal:

$\boxed{\text{Unify}[f, g] = \theta \text{ such that } \text{Subst}[\theta, f] = \text{Subst}[\theta, g]}$.

Unify$[f, g]$ returns Fail if no such θ exists.

Modus ponens - by calling $\theta = \text{Unify}[a'_1 \wedge \cdots \wedge a'_k, a_1 \wedge \cdots \wedge a_k]$,

$$\boxed{\frac{a'_1 \wedge \cdots \wedge a'_k, \forall x_1, \ldots, \forall x_n (a_1 \wedge \cdots \wedge a_k) \to b}{\text{Subst}[\theta, b]}}$$

Completeness - Modus ponens is complete for first-order logic with only Horn clauses.

semi-decidability - first-order logic (even restricted to only Horn clauses) is semi-deciable.

- if $KB \models f$, forward inference on complete inference rules will prove f in finite time
- if $KB \not\models f$, no algorithm can show this in finite time

Resolution rule - by calling $\theta = \text{Unify}[p, q]$,

$$\boxed{\frac{f_1 \vee \cdots \vee f_k \vee p, \neg q \vee g_1 \vee \cdots \vee g_m}{\text{Subst}[\theta, f_1 \vee \cdots \vee f_k \vee g_1 \vee \cdots \vee g_m]}}$$