Forward Chaining, Backward Chaining, and Unification

Adapted from Marie desJardins

**GENERALIZED MODUS PONENS (GMP)**

- Apply modus ponens reasoning to generalized rules
- Combines And-Introduction, Universal-Elimination, and Modus Ponens
  - From \( P(c) \) and \( Q(c) \) and \( (\forall x)(P(x) \land Q(x)) \rightarrow R(x) \) derive \( R(c) \)
- General case: **Given**
  - atomic sentences \( P_1, P_2, ..., P_N \)
  - implication sentence \( (Q_1 \land Q_2 \land ... \land Q_N) \rightarrow R \)
  - \( Q_1, ..., Q_N \) and \( R \) are atomic sentences
  - substitution \( \text{subst}(\theta, P_i) = \text{subst}(\theta, Q_i) \) for \( i=1, ..., N \)
  - Derive new sentence: \( \text{subst}(\theta, R) \)
- Substitutions
  - \( \text{subst}(\theta, a) \) denotes the result of applying a set of substitutions defined by \( \theta \) to the sentence \( a \)
  - A substitution list \( \theta = \{v_1/t_1, v_2/t_2, ..., v_n/t_n\} \) means to replace all occurrences of variable symbol \( v_i \) by term \( t_i \)
  - Substitutions are made in left-to-right order in the list
  - \( \text{subst}(\{x/IceCream, y/Ziggy\}, \text{eats}(y,x)) = \text{eats}(Ziggy, IceCream) \)
**Forward Chaining**

- Proofs start with the given axioms/premises in KB, deriving new sentences using GMP until the goal/query sentence is derived
- This defines a *forward-chaining* inference procedure because it moves “forward” from the KB to the goal [eventually]

**Forward Chaining Example**

- **KB:**
  - allergies(X) → sneeze(X)
  - cat(Y) ∧ allergic-to-cats(X) → allergies(X)
  - cat(Felix)
  - allergic-to-cats(Lise)
- **Goal:**
  - sneeze(Lise)
Applying Chaining Depends on Unification

- Unification is a “pattern-matching” procedure
  - Takes two atomic sentences, called literals, as input
  - Returns “Failure” if they do not match and a substitution list, θ, if they do
- That is, \( \text{unify}(p,q) = \theta \) means \( \text{subst}(\theta, p) = \text{subst}(\theta, q) \) for two atomic sentences, \( p \) and \( q \)
- \( \theta \) is called the most general unifier (mgu)
- All variables in the given two literals are implicitly universally quantified
- To make literals match, replace (universally quantified) variables by terms

Unification Examples

- Example:
  - \( \text{parents}(x, \text{father}(x), \text{mother}(\text{Bill})) \)
  - \( \text{parents}(\text{Bill}, \text{father}(\text{Bill}), y) \)
  - \( \{x/\text{Bill}, y/\text{mother}(\text{Bill})\} \)

- Example:
  - \( \text{parents}(x, \text{father}(x), \text{mother}(\text{Bill})) \)
  - \( \text{parents}(\text{Bill}, \text{father}(y), z) \)
  - \( \{x/\text{Bill}, y/\text{Bill}, z/\text{mother}(\text{Bill})\} \)

- Example:
  - \( \text{parents}(x, \text{father}(x), \text{mother}(\text{Jane})) \)
  - \( \text{parents}(\text{Bill}, \text{father}(y), \text{mother}(y)) \)
  - Failure
UNIFICATION ALGORITHM

procedure unify(p, q, θ)
    Scan p and q left-to-right and find the first corresponding terms where p and q “disagree” (i.e., p and q not equal)
    If there is no disagreement, return θ (success!)
    Let r and s be the terms in p and q, respectively, where disagreement first occurs
    If variable(r) then {
        Let θ = union(θ, {r/s})
        Return unify(subst(θ, p), subst(θ, q), θ)
    } else if variable(s) then {
        Let θ = union(θ, {s/r})
        Return unify(subst(θ, p), subst(θ, q), θ)
    } else return “Failure”
end

UNIFICATION: REMARKS

- *Unify* is a linear-time algorithm that returns the most general unifier (mgu), i.e., the shortest-length substitution list that makes the two literals match.
- In general, there is not a unique minimum-length substitution list, but unify returns one of minimum length
- A variable can never be replaced by a term containing that variable
  - Example: x/f(x) is illegal.
- This “occurs check” should be done in the above pseudo-code before making the recursive calls
FORWARD CHAINING ALGORITHM

procedure \textsc{forward-chain}(KB, p)
\begin{itemize}
  \item if there is a sentence in \( KB \) that is a renaming of \( p \) then return
  \item Add \( p \) to \( KB \)
  \item for each \((p_1 \land \ldots \land p_n \Rightarrow q)\) in \( KB \) such that for some \( i \), UNIFY\((p_i, p) = \theta\) succeeds do
    \item Find-And-Infer\((KB, [p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n], q, \theta)\)
  \item end
\end{itemize}

procedure \textsc{find-and-infer}(KB, premises, conclusion, \( \theta \))
\begin{itemize}
  \item if premises = [] then
    \item Forward-Chain\((KB, \text{Subst}(\theta, \text{conclusion}))\)
  \item else for each \( p' \) in \( KB \) such that \( \text{UNIFY}(p', \text{Subst}(\theta, \text{First(premises)})) = \theta \) do
    \item Find-And-Infer\((KB, \text{Rest}(\text{premises}), \text{conclusion}, \text{Compose}(\theta, \theta_2))\)
  \item end
\end{itemize}

BACKWARD CHAINING
\begin{itemize}
  \item \textbf{Backward-chaining} deduction using GMP is also \textbf{complete} for KBs containing \textbf{only} Horn clauses
  \item Proofs start with the goal query, find rules with that conclusion, and then prove each of the antecedents in the implication
  \item Keep going until you reach premises
  \item Avoid loops: check if new subgoal is already on the goal stack
  \item Avoid repeated work: check if new subgoal
    \begin{itemize}
      \item Has already been proved true
      \item Has already failed
    \end{itemize}
\end{itemize}
**BACKWARD CHAINING EXAMPLE**

- **KB:**
  - allergies(X) $\rightarrow$ sneeze(X)
  - cat(Y) $\land$ allergic-to-cats(X) $\rightarrow$ allergies(X)
  - cat(Felix)
  - allergic-to-cats(Lise)

- **Goal:**
  - sneeze(Lise)

---

**BACKWARD CHAINING ALGORITHM**

```
function BACK-CHAIN(KB, q) returns a set of substitutions
  BACK-CHAIN-LIST(KB, [q], {})

function BACK-CHAIN-LIST(KB, qlist, $\theta$) returns a set of substitutions
  inputs: KB, a knowledge base
  qlist, a list of conjuncts forming a query ($\theta$ already applied)
  $\theta$, the current substitution
  static: answers, a set of substitutions, initially empty

  if qlist is empty then return $\{\theta\}$
  q ← FIRST(qlist)
  for each $q'_i$ in KB such that $\theta_i$ ← UNIFY(q, $q'_i$) succeeds do
    Add COMPOSE($\theta$, $\theta_i$) to answers
  end
  for each sentence $(p_1 \land \ldots \land p_n \Rightarrow q'_i)$ in KB such that $\theta_i$ ← UNIFY(q, $q'_i$) succeeds do
    answers ← BACK-CHAIN-LIST(KB, SUBST($\theta_i$, [p_1 \ldots p_n]), $\theta$) union answers
  end
  return the union of BACK-CHAIN-LIST(KB, REST(qlist), $\theta$) for each $\theta \in$ answers
```
FORWARD VS. BACKWARD CHAINING

• FC is data-driven
  • Automatic, unconscious processing
  • E.g., object recognition, routine decisions
  • May do lots of work that is irrelevant to the goal

• BC is goal-driven, appropriate for problem-solving
  • Where are my keys? How do I get to my next class?
  • Complexity of BC can be much less than linear in the size of the KB

AUTOMATING FOL INFERENCE WITH RESOLUTION
**Resolution**
- Resolution is a *sound* and *complete* inference procedure for FOL.
- Reminder: Resolution rule for propositional logic:
  - $P_1 \vee P_2 \vee ... \vee P_n$
  - $\neg P_1 \vee Q_2 \vee ... \vee Q_m$
  - Resolvent: $P_2 \vee ... \vee P_n \vee Q_2 \vee ... \vee Q_m$
- Examples
  - $P$ and $\neg P \vee Q$: derive $Q$ (Modus Ponens)
  - $(\neg P \vee Q)$ and $(\neg Q \vee R)$: derive $\neg P \vee R$
  - $P$ and $\neg P$: derive $\text{False}$ [contradiction!]
  - $(P \vee Q)$ and $(\neg P \vee \neg Q)$: derive $\text{True}$

**Resolution in First-Order Logic**
- Given sentences:
  - $P_1 \vee ... \vee P_n$
  - $Q_1 \vee ... \vee Q_m$
- in *conjunctive normal form*:
  - each $P_i$ and $Q_i$ is a literal, i.e., a positive or negated predicate symbol with its terms,
- if $P_j$ and $\neg Q_k$ unify with substitution list $\theta$, then derive the resolvent sentence:
  - $\text{subst}(\theta, P_1 \vee ... \vee P_{j-1} \vee P_{j+1} ... P_n \vee Q_1 \vee ... Q_{k-1} \vee Q_{k+1} \vee ... \vee Q_m)$
- Example
  - from clause $P(x, f(a)) \vee P(x, f(y)) \vee Q(y)$
  - and clause $\neg P(z, f(a)) \vee \neg Q(z)$
  - derive resolvent $P(z, f(y)) \vee Q(y) \vee \neg Q(z)$
  - using $\theta = \{x/z\}$
**Resolution Refutation**

- Given a consistent set of axioms KB and goal sentence Q, show that KB |= Q.
- **Proof by contradiction**: Add ¬Q to KB and try to prove false.
  - i.e., (KB |- Q) <-> (KB ∧ ¬Q |- False)
- Resolution is **refutation complete**: it can establish that a given sentence Q is entailed by KB, but can't (in general) be used to generate all logical consequences of a set of sentences.
- Also, it cannot be used to prove that Q is **not entailed** by KB.
- Resolution **won't always give an answer** since entailment is only semidecidable.
  - And you can't just run two proofs in parallel, one trying to prove Q and the other trying to prove ¬Q, since KB might not entail either one.