FORWARD CHAINING

Oksana Emerson

Inference Chaining

- Generalized Modus Ponens gives us a natural, intuitive, and reasonably powerful tool to use for inference.
- There are two types of inference that differ simply in direction.
- Forward Chaining starts with new premises and tries to generate all new conclusions.
- Backward Chaining begins from a desired conclusion and attempts to realize the necessary implications and premises required to arrive at it.

Inference Procedures

Forward Chaining

- Data-driven
- Triggered by adding a new fact
- Premises $\Rightarrow$ Consequent(s)
- Renaming
- ***Composition***:

$$\text{Subst}(\text{Compose}(\theta_1, \theta_2), p) = \text{Subst}(\theta_2, \text{Subst}(\theta_1, p))$$

Backward Chaining

- Goal-driven
- Triggered by a query , i.e. Ask
- Premises $\Leftarrow$ Consequent
- Build up the unifier as it goes

Which method to use?

FORWARD CHAINING

If the 'average' rule has more conditions than conclusions, that is the typical hypothesis or goal (the conclusions) can lead to many more questions (the conditions)

BACKWARD CHAINING

The average rule has more conclusions than conditions such that each fact may fan out into a large number of new facts or actions

Forward Chaining Algorithm

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

  repeat until *new* is empty
    $new \leftarrow \{\}$
    for each sentence r **in** KB **do**
      $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
      for each θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
          for some $p'_1, \ldots, p'_n$ in KB
       $q' \leftarrow$ SUBST(θ, q)
       if q' is not a renaming of a sentence already in KB or *new* **then do**
         add q' to *new*
         $\phi \leftarrow$ UNIFY(q', α)
         if ϕ is not *fail* **then return** ϕ
    add *new* to KB
  return *false*

... it is a crime for an American to sell weapons to hostile nations:

$American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$

Nono ... has some missiles, i.e., $\exists x\; Owns(Nono, x) \land Missile(x)$:

$Owns(Nono, M_1)$ and $Missile(M_1)$

... all of its missiles were sold to it by Colonel West

$\forall x\;\; Missile(x) \land Owns(Nono, x) \;\Rightarrow\; Sells(West, x, Nono)$

Missiles are weapons:

$Missile(x) \Rightarrow Weapon(x)$

An enemy of America counts as "hostile":

$Enemy(x, America) \;\Rightarrow\; Hostile(x)$

West, who is American ...

$American(West)$

The country Nono, an enemy of America ...

$Enemy(Nono, America)$

Properties of forward chaining

Sound and complete for first-order definite clauses
(proof similar to propositional proof)

Datalog = first-order definite clauses + *no functions* (e.g., crime KB)
FC terminates for Datalog in poly iterations: at most $p \cdot n^k$ literals

May not terminate in general if α is not entailed

This is unavoidable: entailment with definite clauses is semidecidable

Efficiency of forward chaining

Simple observation: no need to match a rule on iteration k
if a premise wasn't added on iteration $k-1$

$\Rightarrow$ match each rule whose premise contains a newly added literal

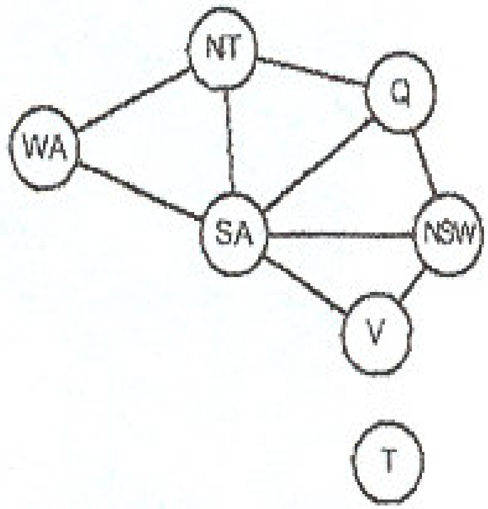
Matching itself can be expensive

Database indexing allows $O(1)$ retrieval of known facts
e.g., query $Missile(x)$ retrieves $Missile(M_1)$

Matching conjunctive premises against known facts is NP-hard

Forward chaining is widely used in deductive databases

Hard matching example



$$\begin{aligned}
&\mathit{Diff}(wa, nt) \land \mathit{Diff}(wa, sa) \land \\
&\quad \mathit{Diff}(nt, q) \mathit{Diff}(nt, sa) \land \\
&\quad \mathit{Diff}(q, nsw) \land \mathit{Diff}(q, sa) \land \\
&\quad \mathit{Diff}(nsw, v) \land \mathit{Diff}(nsw, sa) \land \\
&\quad \mathit{Diff}(v, sa) \Rightarrow \mathit{Colorable}()
\end{aligned}$$

$\mathit{Diff}(Red, Blue)$ $\mathit{Diff}(Red, Green)$
$\mathit{Diff}(Green, Red)$ $\mathit{Diff}(Green, Blue)$
$\mathit{Diff}(Blue, Red)$ $\mathit{Diff}(Blue, Green)$

$\mathit{Colorable}()$ is inferred iff the CSP has a solution

CSPs include 3SAT as a special case, hence matching is NP-hard