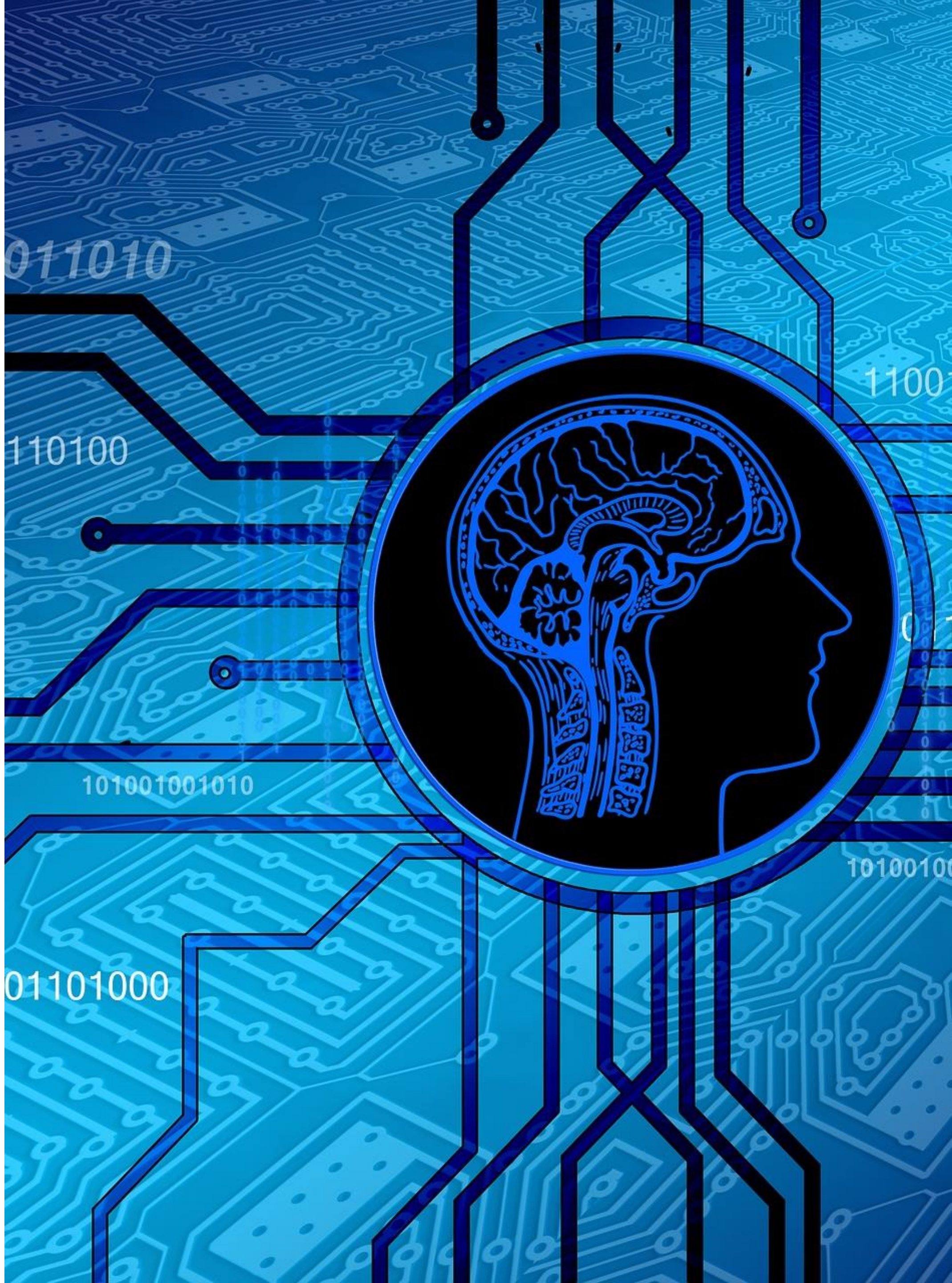
Resolution-based Inference

Petros Papapanagiotou

Informatics 2D: Reasoning and Agents
Lecture 13



Forward chaining

'Winnie-the-Pooh' Knowledge Base

$VeryFondOfFood(x) \wedge Treat(y) \wedge Friend(z) \wedge Gives(x, y, z) \Rightarrow Generous(x)$

$Owns(Eeyore, J) \wedge Hunny(J)$

$Hunny(x) \wedge Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

$Hunny(x) \Rightarrow Treat(x)$

$Resident(x, HAW) \Rightarrow Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$

Forward chaining proof

$VeryFondOfFood(x) \wedge Treat(y) \wedge Friend(z) \wedge Gives(x, y, z) \Rightarrow Generous(x)$

$Owns(Eeyore, J) \wedge Hunny(J)$

$Hunny(x) \wedge Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

$Hunny(x) \Rightarrow Treat(x)$

$Resident(x, HAW) \Rightarrow Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$

Forward chaining proof

$VeryFondOfFood(x) \wedge Treat(y) \wedge Friend(z) \wedge Gives(x, y, z) \Rightarrow Generous(x)$

$Owns(Eeyore, J) \wedge Hunny(J)$

$Hunny(x) \wedge Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

$Hunny(x) \Rightarrow Treat(x)$

$Resident(x, HAW) \Rightarrow Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$

VeryFondOfFood(Pooh) | Hunny(J) | Owns(Eeyore,J) | Resident(Eeyore,HAW)

Forward chaining proof

$VeryFondOfFood(x) \land Treat(y) \land Friend(z) \land Gives(x, y, z) \Rightarrow Generous(x)$

$Owns(Eeyore, J) \land Hunny(J)$

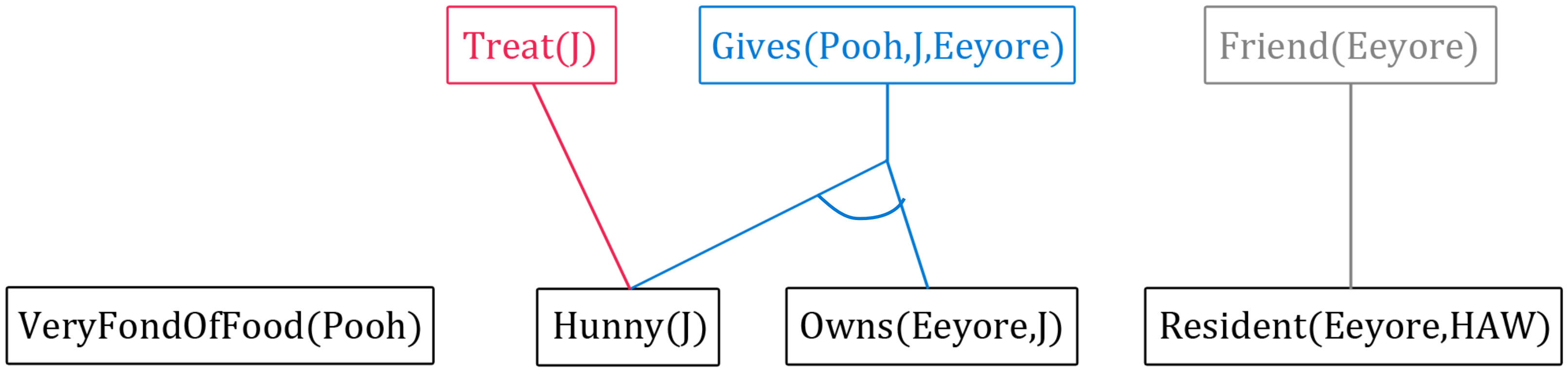
$Hunny(x) \land Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

$Hunny(x) \Rightarrow Treat(x)$

$Resident(x, HAW) \Rightarrow Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Forward chaining proof

$VeryFondOfFood(x) \land Treat(y) \land Friend(z) \land Gives(x, y, z) \Rightarrow Generous(x)$

$Owns(Eeyore, J) \land Hunny(J)$

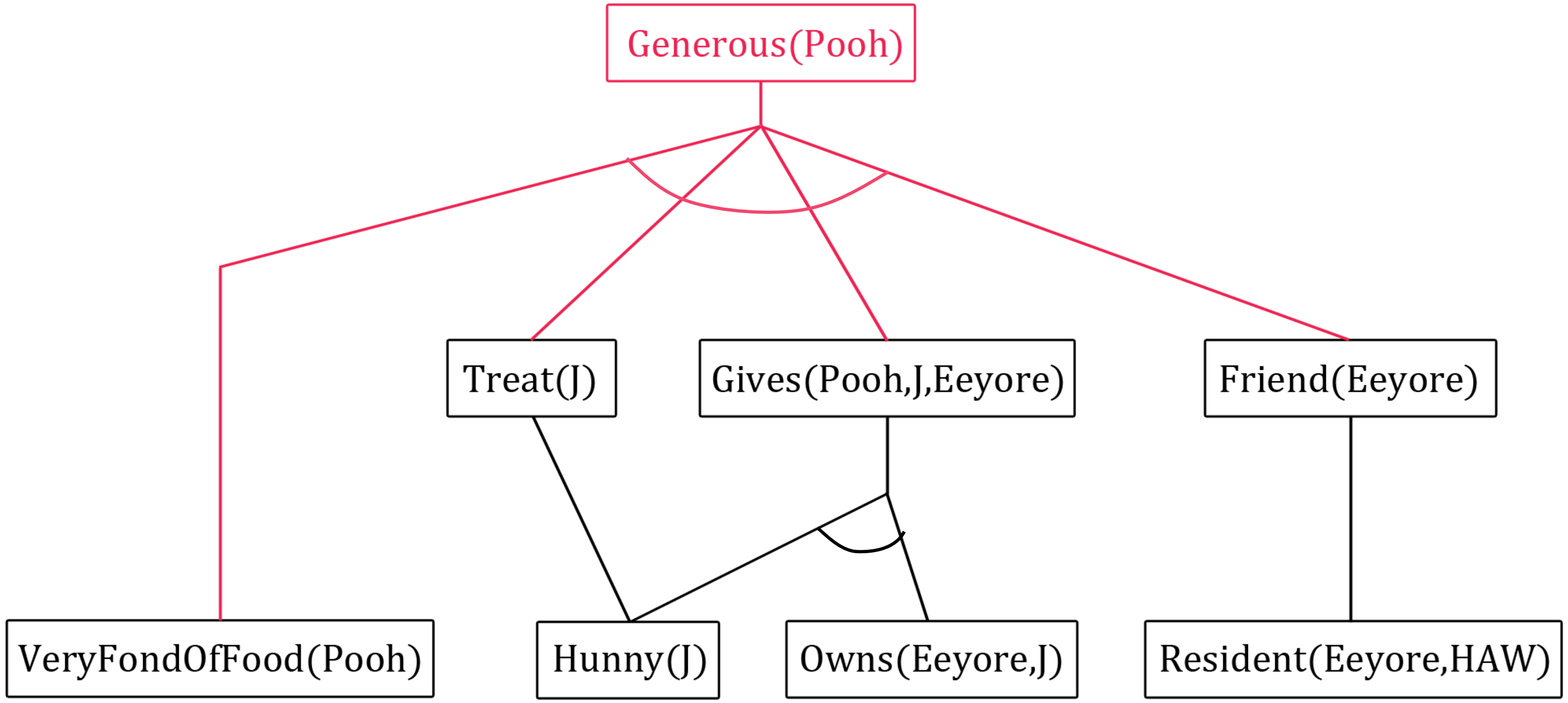
$Hunny(x) \land Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

$Hunny(x) \Rightarrow Treat(x)$

$Resident(x, HAW) \Rightarrow Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Forward chaining algorithm

```
function FOL-FC-ASK(KB, α) returns a substitution or false
  inputs: KB, the knowledge base, a set of first-order definite clauses
          α, the query, an atomic sentence
  local variables: new, the new sentences inferred on each iteration

  repeat until new is empty
     new ← { }
     for each rule in KB do
        (p1 ∧ ... ∧ pn ⇒ q) ← STANDARDIZE-VARIABLES(rule)
        for each θ such that SUBST(θ, p1 ∧ ... ∧ pn) = SUBST(θ, p'1 ∧ ... ∧ p'n)
                 for some p'1, ..., p'n in KB
           q' ← SUBST(θ, q)
           if q' does not unify with some sentence already in KB or new then
              add q' to new
              φ ← UNIFY(q', α)
              if φ is not fail then return φ
     add new to KB
  return false
```

Pattern-matching

Facts irrelevant to the goal can be generated

Properties of forward chaining

Sound and complete for first-order definite clauses

- Definite clause = exactly one positive literal.

Datalog = first-order definite clauses + no functions

- FC terminates for Datalog in finite number of iterations

May not terminate in general if α is not entailed

This is unavoidable: entailment with definite clauses is semi-decidable

Efficiency of forward chaining

Incremental forward chaining: 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 positive literal

Matching itself can be expensive:

Database indexing allows O(1) retrieval of known facts

- e.g. query *Hunny(x)* retrieves *Hunny(J)*

Forward chaining is widely used in deductive databases

Efficiency of forward chaining II

$$\textbf{for each } \theta \text{ such that } \text{SUBST}(\theta, p_1 \land \ldots \land p_n) = \text{SUBST}(\theta, p'_1 \land \ldots \land p'_n) \text{ for some } p'_1, \ldots, p'_n \text{ in } KB$$

- Finding all possible unifiers can be very expensive

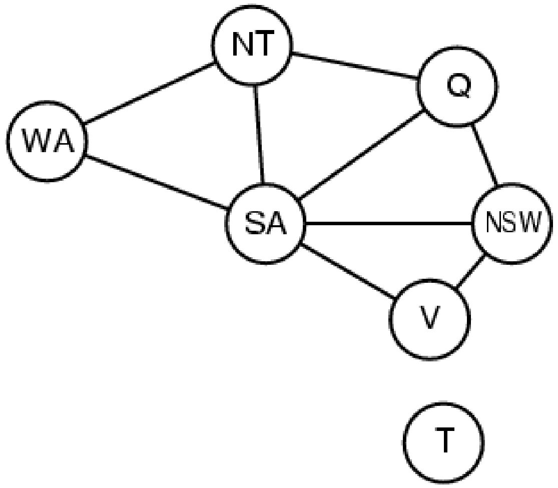
Example:

$$Hunny(x) \land Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$$

- Can find each object owned by Eeyore in constant time and then check if it is a jar of hunny.
- *But* what if Eeyore owns many objects but very few jars?
- **Conjunct Ordering**: Better (cost-wise) to find all jars first and then check whether they are owned by Eeyore.

- Optimal ordering is NP-hard. Heuristics available: e.g. MRV from CSP if each conjunct is viewed as a constraint on its variables.

Hard matching example



$Diff(WA, NT) \wedge Diff(WA, SA) \wedge Diff(NT, Q) \wedge$
$Diff(NT, SA) \wedge Diff(Q, NSW) \wedge Diff(Q, SA) \wedge$
$Diff(NSW, V) \wedge Diff(NSW, SA) \wedge Diff(V, SA)$
$\Rightarrow$ *Colourable*

Diff(Red, Blue) Diff (Red, Green)
Diff(Green, Red) Diff(Green, Blue)
Diff(Blue, Red) Diff(Blue, Green)

Every finite domain CSP can be expressed as a single definite clause + ground facts

Colourable is inferred iff the CSP has a solution

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

Backward chaining

Backward chaining proof

$VeryFondOfFood(x) \wedge Treat(y) \wedge Friend(z) \wedge Gives(x, y, z) \Rightarrow Generous(x)$

$Owns(Eeyore, J) \wedge Hunny(J)$

$Hunny(x) \wedge Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

$Hunny(x) \Rightarrow Treat(x)$

$Resident(x, HAW) \Rightarrow Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$

$\boxed{\text{Generous(Pooh)}}$

Backward chaining proof

$VeryFondOfFood(x) \wedge Treat(y) \wedge Friend(z) \wedge Gives(x, y, z) \Rightarrow Generous(x)$

$Owns(Eeyore, J) \wedge Hunny(J)$

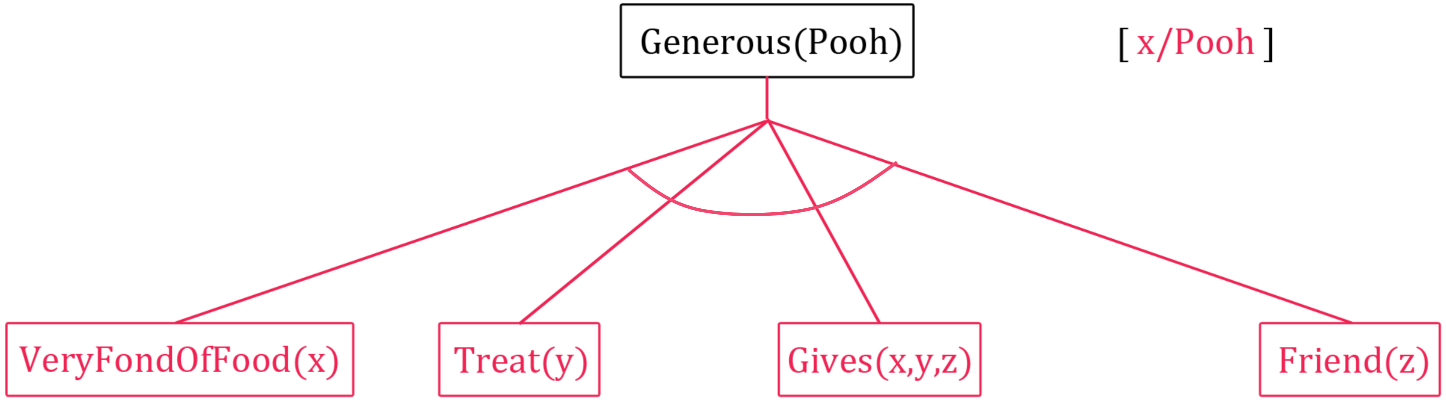
$Hunny(x) \wedge Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

$Hunny(x) \Rightarrow Treat(x)$

$Resident(x, HAW) \Rightarrow Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Backward chaining proof

$VeryFondOfFood(x) \wedge Treat(y) \wedge Friend(z) \wedge Gives(x, y, z) \Rightarrow Generous(x)$

$Owns(Eeyore, J) \wedge Hunny(J)$

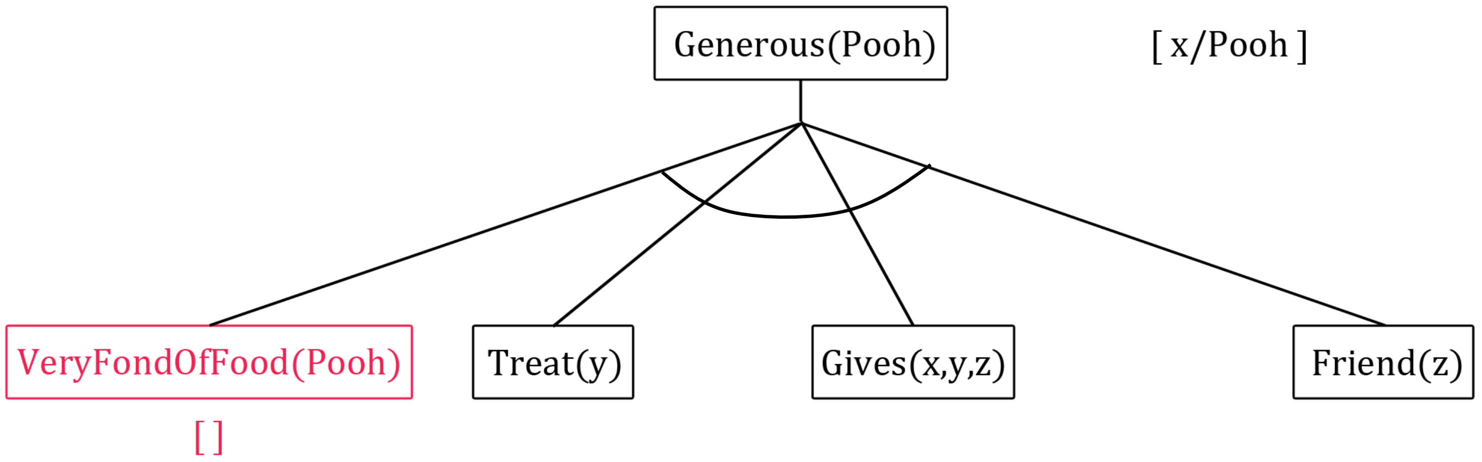
$Hunny(x) \wedge Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

$Hunny(x) \Rightarrow Treat(x)$

$Resident(x, HAW) \Rightarrow Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Backward chaining proof

$VeryFondOfFood(x) \land Treat(y) \land Friend(z) \land Gives(x, y, z) \Rightarrow Generous(x)$

$Owns(Eeyore, J) \land Hunny(J)$

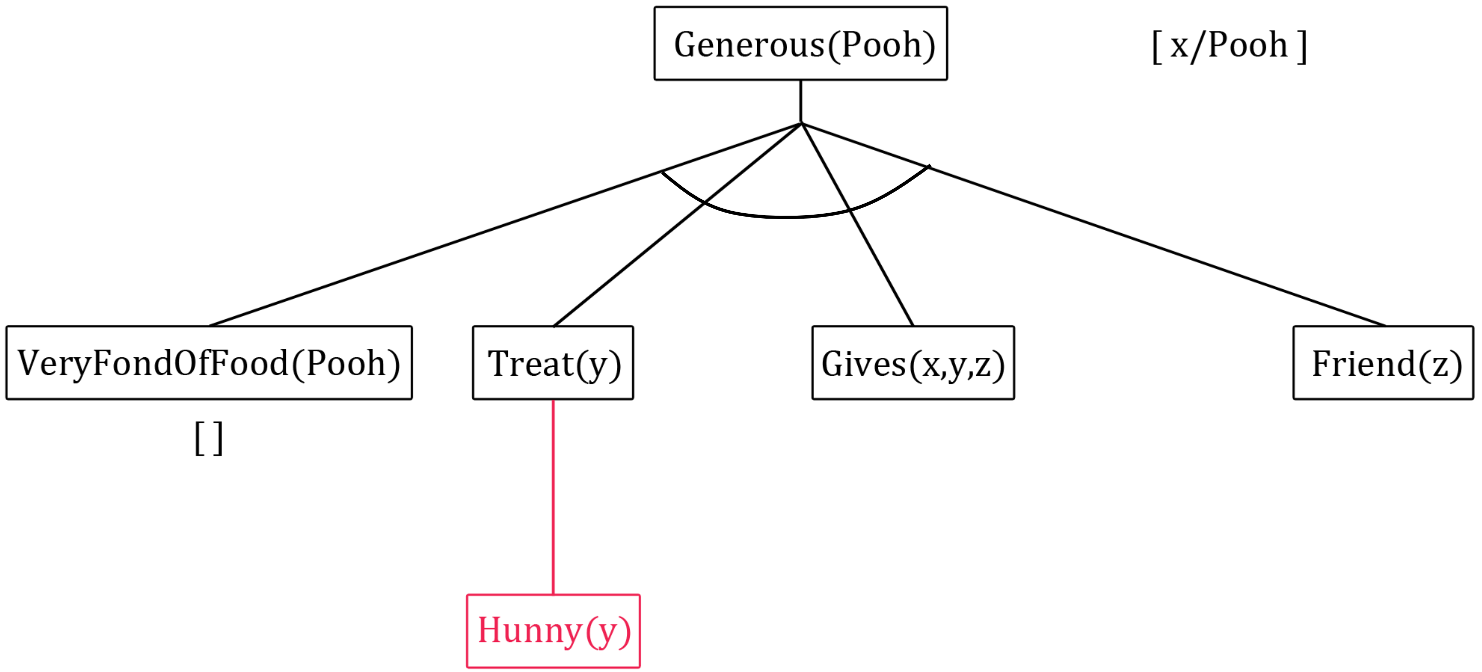
$Hunny(x) \land Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

$Hunny(x) \Rightarrow Treat(x)$

$Resident(x, HAW) \Rightarrow Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Backward chaining proof

$VeryFondOfFood(x) \wedge Treat(y) \wedge Friend(z) \wedge Gives(x, y, z) \Rightarrow Generous(x)$

$Owns(Eeyore, J) \wedge Hunny(J)$

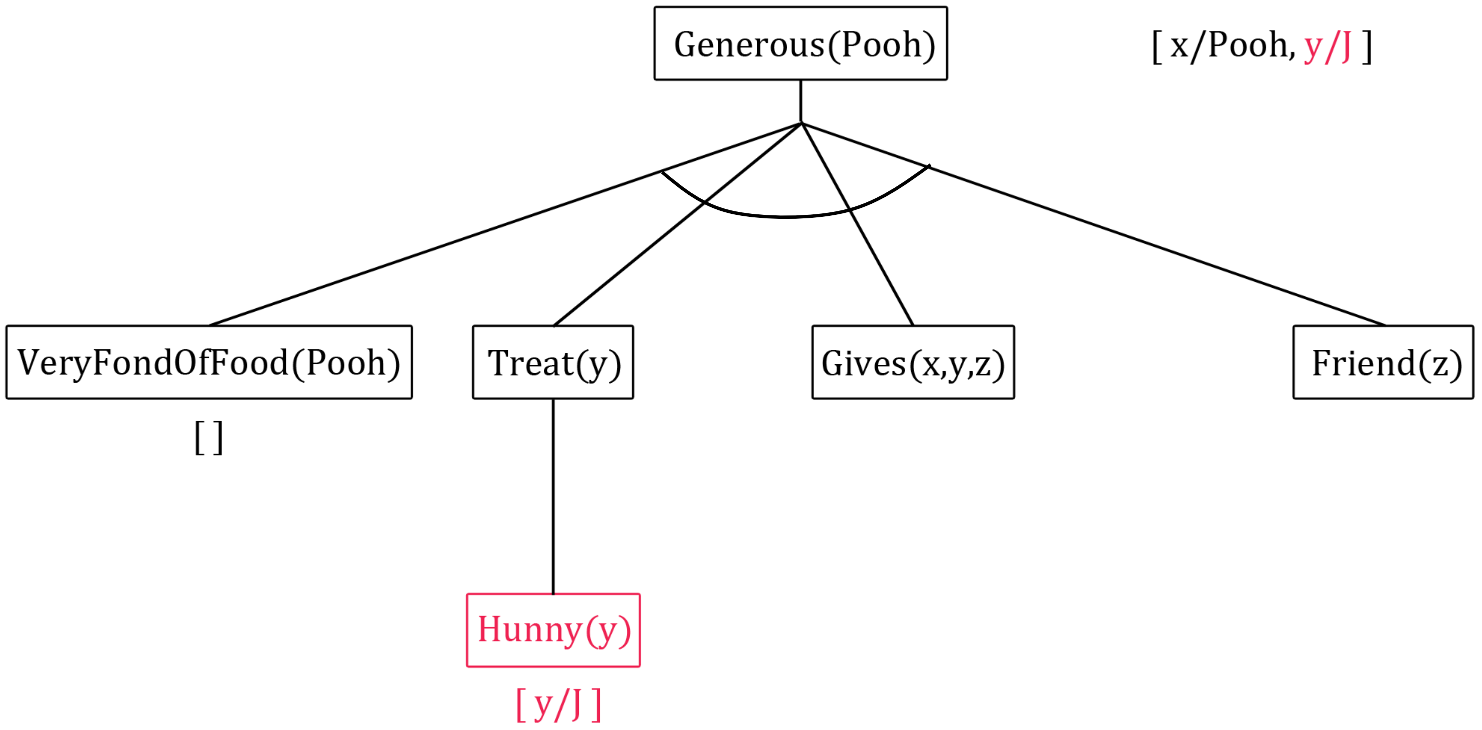
$Hunny(x) \wedge Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

$Hunny(x) \Rightarrow Treat(x)$

$Resident(x, HAW) \Rightarrow Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Backward chaining proof

$VeryFondOfFood(x) \land Treat(y) \land Friend(z) \land Gives(x, y, z) \Rightarrow Generous(x)$

$Owns(Eeyore, J) \land Hunny(J)$

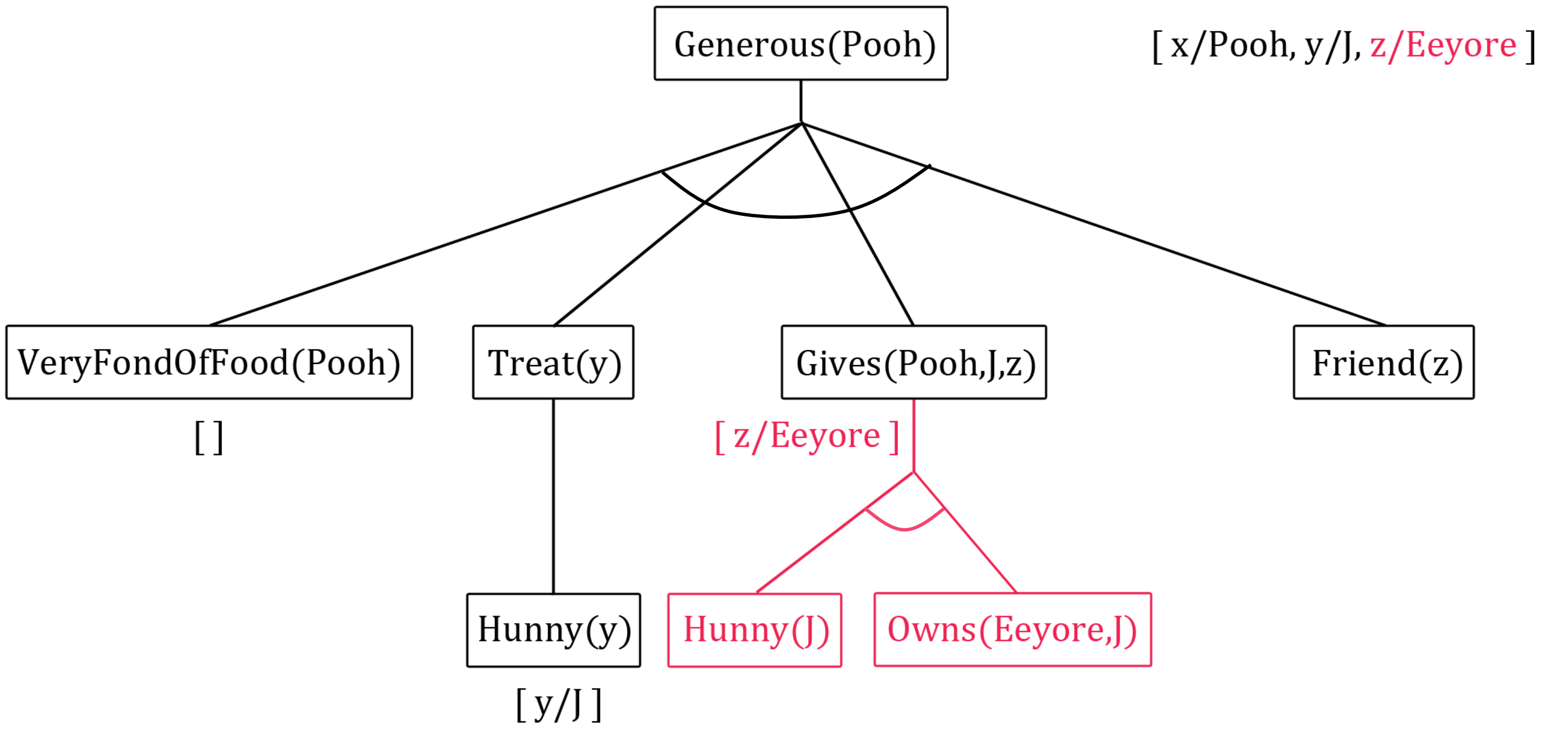
$Hunny(x) \land Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

$Hunny(x) \Rightarrow Treat(x)$

$Resident(x, HAW) \Rightarrow Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Backward chaining proof

$VeryFondOfFood(x) \land Treat(y) \land$
$Friend(z) \land Gives(x,y,z) \Rightarrow Generous(x)$

$Owns(Eeyore, J) \land Hunny(J)$

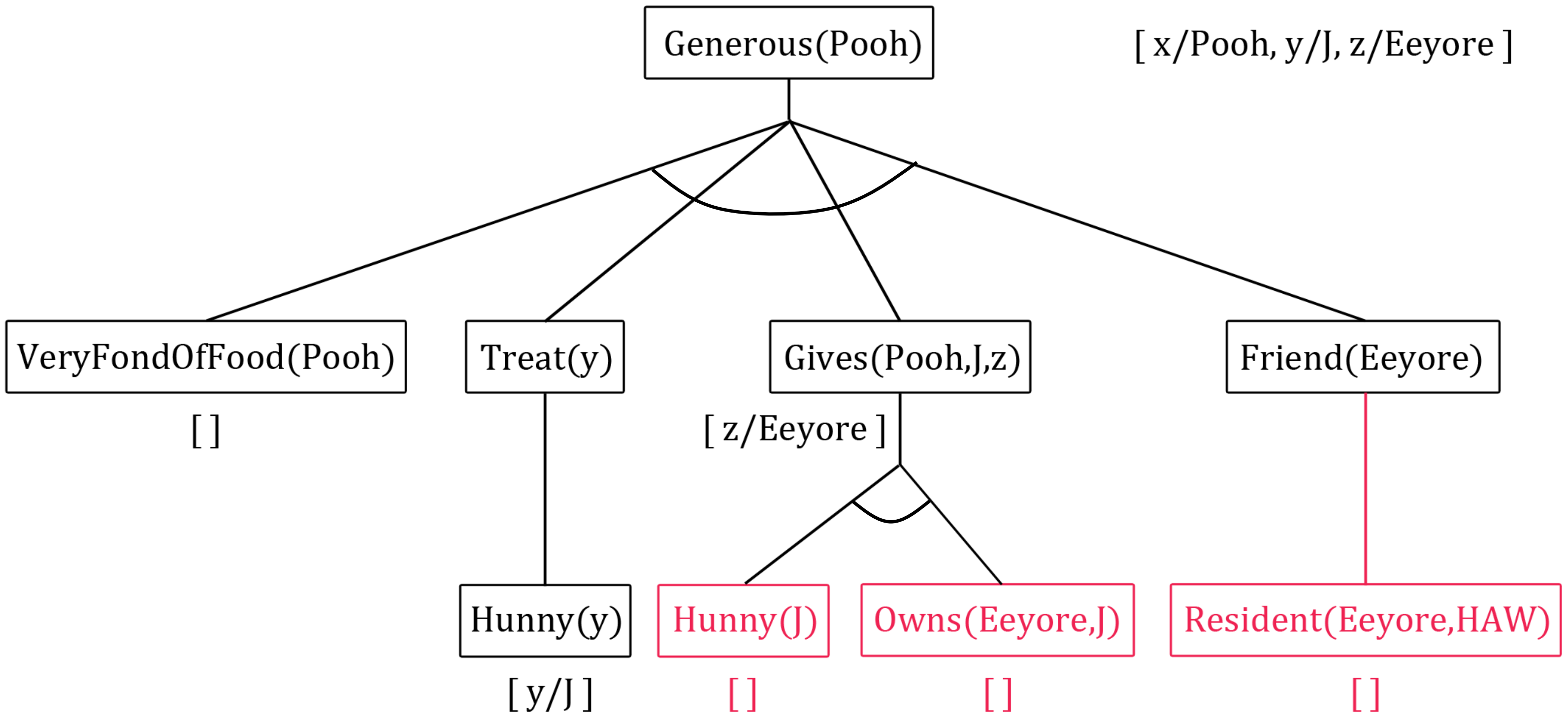
$Hunny(x) \land Owns(Eeyore, x) \Rightarrow$
$Gives(Pooh, x, Eeyore)$

$Hunny(x) \Rightarrow Treat(x)$

$Resident(x, HAW) \Rightarrow Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Backward chaining algorithm

Fetch rules that might unify

```
function FOL-BC-ASK(KB, query) returns a gene
   return FOL-BC-OR(KB, query, { })

generator FOL-BC-OR(KB, goal, θ) yields a substitution
   for each rule (lhs ⇒ rhs) in FETCH-RULES-FOR-GOAL(KB, goal) do
      (lhs, rhs) ← STANDARDIZE-VARIABLES((lhs, rhs))
      for each θ′ in FOL-BC-AND(KB, lhs, UNIFY(rhs, goal, θ)) do
         yield θ′

generator FOL-BC-AND(KB, goals, θ) yields a substitution
   if θ = failure then return
   else if LENGTH(goals) = 0 then yield θ
   else do
      first,rest ← FIRST(goals), REST(goals)
      for each θ′ in FOL-BC-OR(KB, SUBST(θ, first), θ) do
         for each θ′′ in FOL-BC-AND(KB, rest, θ′) do
            yield θ′′
```

A function that returns multiple times, each time giving one possible result

Properties of backward chaining

Depth-first recursive proof search: space is linear in size of proof

Incomplete due to infinite loops

- partial fix by checking current goal against every goal on stack

Inefficient due to repeated subgoals (both success and failure)

- fix using caching of previous results (extra space)

Widely used for logic programming.

Resolution

Ground Binary Resolution

$$\frac{C \vee P \qquad D \vee \neg P}{C \vee D}$$

Soundness:

$C \vee P$ iff $\neg C \Rightarrow P$

$D \vee \neg P$ iff $P \Rightarrow D$

- Therefore, $\neg C \Rightarrow D$
- Which is equivalent to $C \vee D$

Note: if both C and D are empty then resolution deduces the *empty clause*, i.e. **false**.

Non-Ground Binary Resolution

$$\frac{C \lor P \qquad D \lor \neg P'}{(C \lor D)\theta}$$

where θ is the mgu of P and P'

- The two clauses are assumed to be standardized apart so that they share no variables.

Soundness: apply θ to premises then appeal to ground binary resolution.

$$\frac{C\theta \lor P\theta \qquad D\theta \lor \neg P\theta}{C\theta \lor D\theta}$$

Example

$$\frac{\neg HasHunny(x) \vee Happy(x) \qquad HasHunny(Pooh)}{Happy(Pooh)}$$

with $\theta = \{x/Pooh\}$

Factoring

$$\boxed{\frac{C \vee P_1 \vee \cdots \vee P_m}{(C \vee P_1)\theta} \quad \text{where } \theta \text{ is the mgu of the } P_i}$$

Soundness: by universal instantiation and deletion of duplicates.

Full Resolution

$$\frac{C \vee P_1 \vee \cdots \vee P_m \qquad D \vee \neg P_1' \vee \cdots \vee \neg P_n'}{(C \vee D)\theta}$$

where ϑ is mgu of all P_i and P_i'

Soundness: by combination of factoring and binary resolution.

To prove α: apply resolution steps to CNF($KB \wedge \neg\alpha$);

- complete for FOL, if full resolution or binary resolution + factoring is used

Conversion to CNF (1/2)

$$\forall x. \left(\forall y. Animal(y) \Rightarrow Loves(x, y)\right) \Rightarrow (\exists y. Loves(y, x))$$

Eliminate ⇔, ⇒ : replace $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$ and $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$

- $\forall x. \neg\left(\forall y. \neg Animal(y) \vee Loves(x, y)\right) \vee (\exists y. Loves(y, x))$

Move ¬ inwards : use de Morgan's rules, $\neg\neg\alpha = \alpha$, $\neg\forall x. P \equiv \exists x. \neg P, \neg\exists x. P \equiv \forall x. \neg P$

- $\forall x. (\exists y. \neg(\neg Animal(y) \vee Loves(x, y))) \vee (\exists y. Loves(y, x))$
- $\forall x. \left(\exists y. \neg\neg Animal(y) \wedge \neg Loves(x, y)\right) \vee (\exists y. Loves(y, x))$
- $\forall x. \left(\exists y. Animal(y) \wedge \neg Loves(x, y)\right) \vee (\exists y. Loves(y, x))$

Standardize variables apart: each quantifier should use a different one

- $\forall x. \left(\exists y. Animal(y) \wedge \neg Loves(x, y)\right) \vee (\exists z. Loves(z, x))$

Conversion to CNF (2/2)

$$\forall x. \left(\exists y. Animal(y) \land \neg Loves(x, y)\right) \lor (\exists z. Loves(z, x))$$

Skolemize: a more general form of existential instantiation

- Each existential variable is replaced by a **Skolem function** of the enclosing universally quantified variables.
- $\forall x. \left(Animal(F(x)) \land \neg Loves(x, F(x))\right) \lor Loves(G(x), x)$

Drop universal quantifiers ∀

- $\left(Animal(F(x)) \land \neg Loves(x, F(x))\right) \lor Loves(G(x), x)$

Create clauses: apply distributivity law (∨ over ∧) and flatten

- $\left(Animal\big(F(x)\big) \lor Loves(G(x), x)\right) \land \left(\neg Loves\big(x, F(x)\big) \lor Loves(G(x), x)\right)$

Resolution algorithm

```
function PL-RESOLUTION(KB, α) returns true or false
  inputs: KB, the knowledge base, a sentence in propositional logic
          α, the query, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
  new ← { }
  loop do
     for each pair of clauses Ci, Cj in clauses do
        resolvents ← PL-RESOLVE(Ci, Cj)
        if resolvents contains the empty clause then return true
        new ← new ∪ resolvents
     if new ⊆ clauses then return false
     clauses ← clauses ∪ new
```

returns the set of all possible clauses obtained by resolving its two inputs

‘Winnie-the-Pooh’ Knowledge Base

$VeryFondOfFood(x) \wedge Treat(y) \wedge Friend(z) \wedge Gives(x, y, z) \Rightarrow Generous(x)$

$Owns(Eeyore, J) \wedge Hunny(J)$

$Hunny(x) \wedge Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

$Hunny(x) \Rightarrow Treat(x)$

$Resident(x, HAW) \Rightarrow Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$

'Winnie-the-Pooh' Knowledge Base

$\neg VeryFondOfFood(x) \vee \neg Treat(y) \vee \neg Friend(z) \vee \neg Gives(x, y, z) \vee Generous(x)$

$Owns(Eeyore, J) \qquad Hunny(J)$

$\neg Hunny(x) \vee \neg Owns(Eeyore, x) \vee Gives(Pooh, x, Eeyore)$

$\neg Hunny(x) \vee Treat(x)$

$\neg Resident(x, HAW) \vee Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$

Resolution proof

$\neg VeryFondOfFood(x) \vee \neg Treat(y) \vee \neg Friend(z) \vee \neg Gives(x, y, z) \vee Generous(x)$

$Owns(Eeyore, J)$ $Hunny(J)$

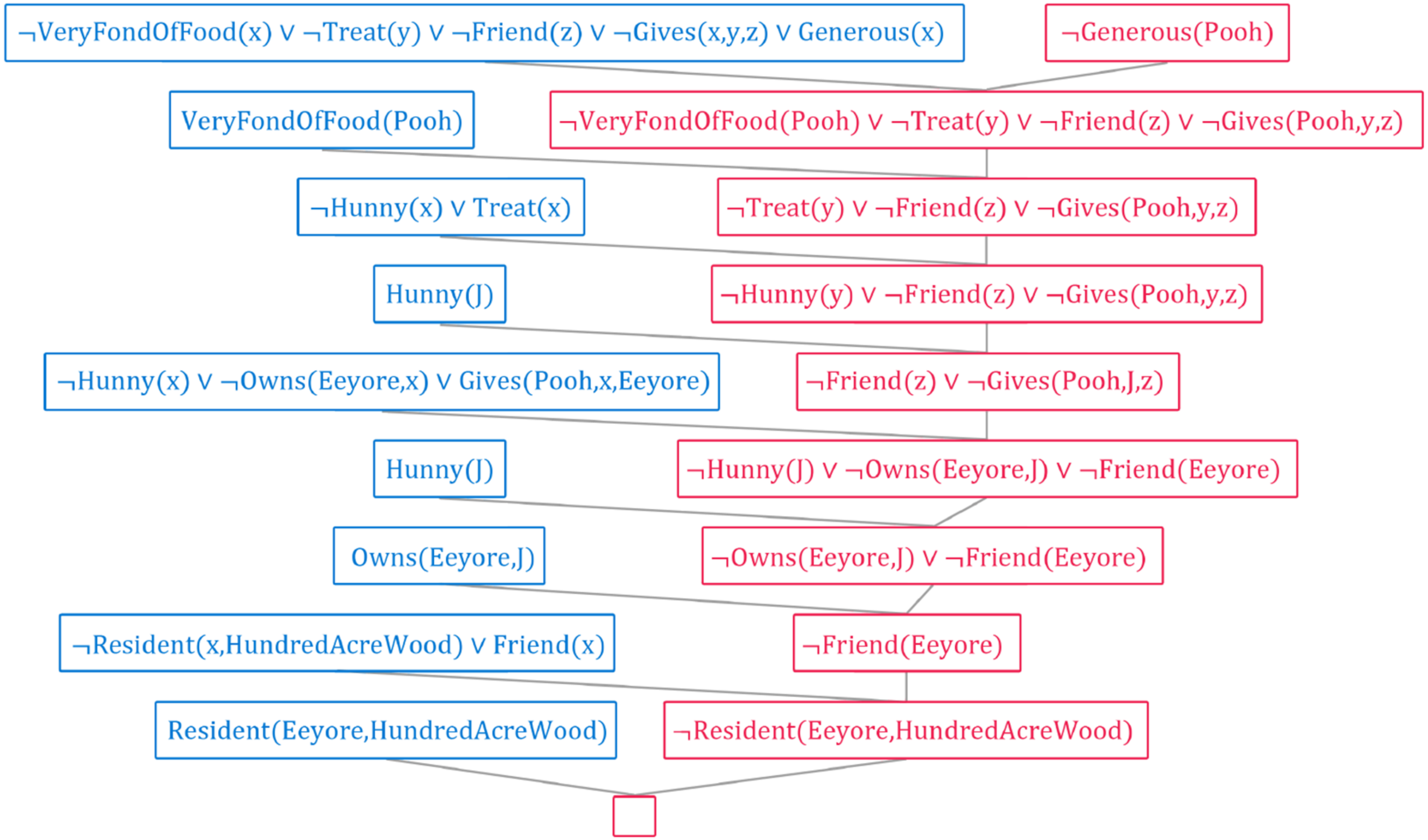
$\neg Hunny(x) \vee \neg Owns(Eeyore, x) \vee Gives(Pooh, x, Eeyore)$

$\neg Hunny(x) \vee Treat(x)$

$\neg Resident(x, HAW) \vee Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Why?

Fundamentals of reasoning in FOL.

Automated logic-based reasoning.

Proof search.

Applications discussed in Lecture 11.