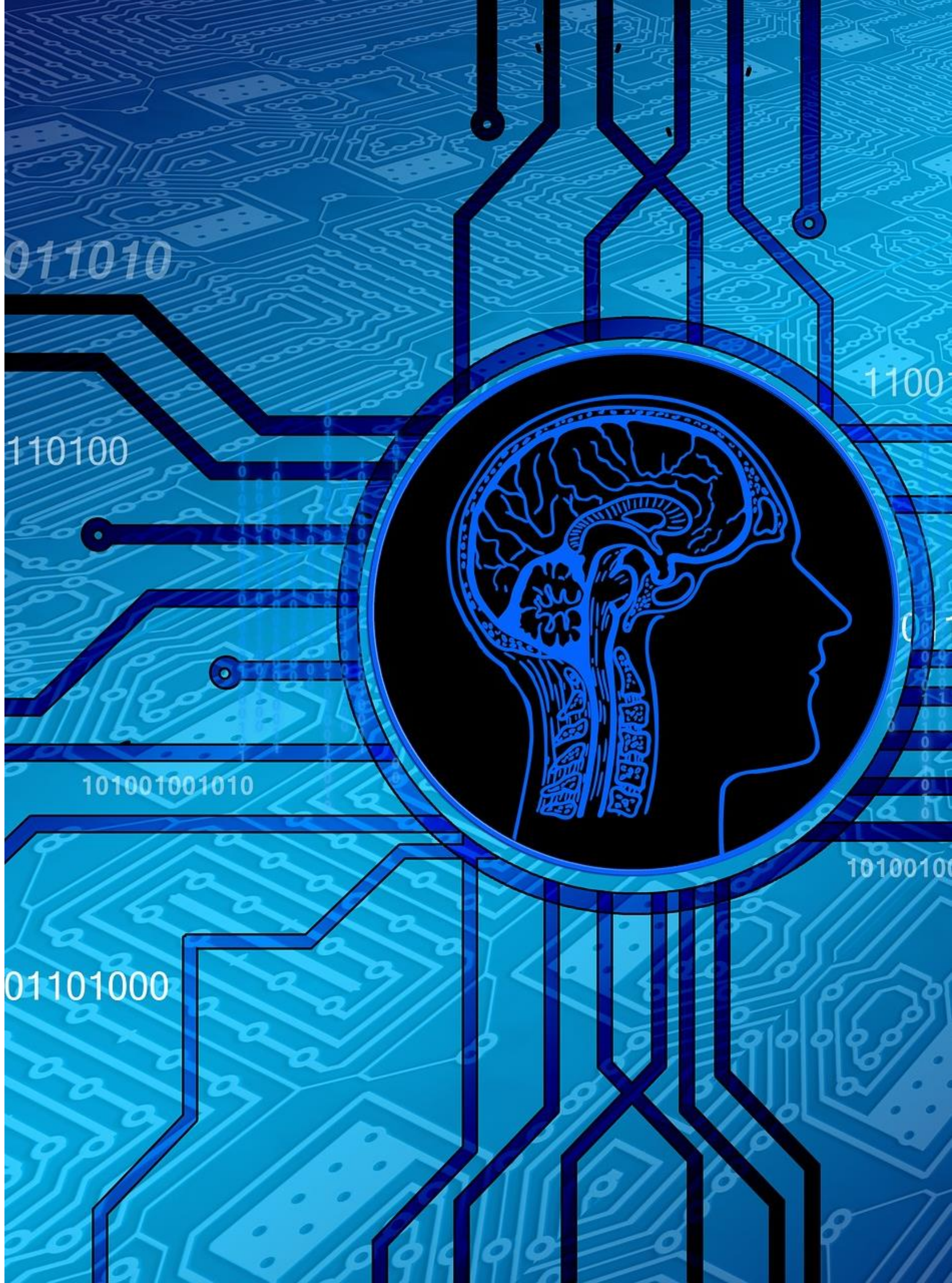
Resolution-based Inference

Informatics 2D: Reasoning and Agents

Adapted from slides provided by Dr Petros Papapanagiotou



Propositionalization · Generalized Modus Ponens · Unification

Outline

Forward Chaining · Backward Chaining · Resolution

Winnie-the-Pooh: A generous teddy bear

It is known in The Hundred-Acre Wood that if someone who is very fond of food gives a treat to one of their friends, they are really generous.

Eeyore, the sad donkey, has some hunny that he has received for his birthday from Winnie-the-Pooh, who, as we know, is very fond of food.

Prove that Winnie-the-Pooh is generous.

A Formalisation in First-order Logic

if someone who is very fond of food gives a treat to one of their friends, they are really generous

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

Eeyore (…) has some hunny

- $\exists x. Owns(Eeyore, x) \wedge Hunny(x)$ or after EI: $Owns(Eeyore, H_1) \wedge Hunny(H_1)$

that he has received for his birthday from Winnie-the-Pooh

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

Hunny is a treat.

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

Residents of the the Hundred-Acre Wood are friends.

- $Resident(x, HundredAcreWood) \Rightarrow Friend(x)$

Eeyore is a resident of the the Hundred-Acre Wood.

- $Resident(Eeyore, HundredAcreWood)$

Pooh is very fond of food.

- $VeryFondOfFood(Pooh)$

Forward chaining

'Winnie-the-Pooh' Knowledge Base

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

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

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

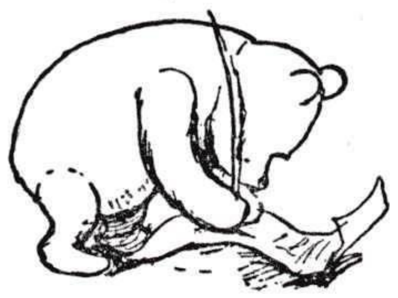
$Hunny(x) \Rightarrow Treat(x)$

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

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$

Prove that Winnie-the-Pooh is generous



Forward chaining proof

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

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

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

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

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

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$

Prove that Winnie-the-Pooh is generous

Forward chaining proof

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

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

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

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

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

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$

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

Prove that Winnie-the-Pooh is generous

Forward chaining proof

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

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

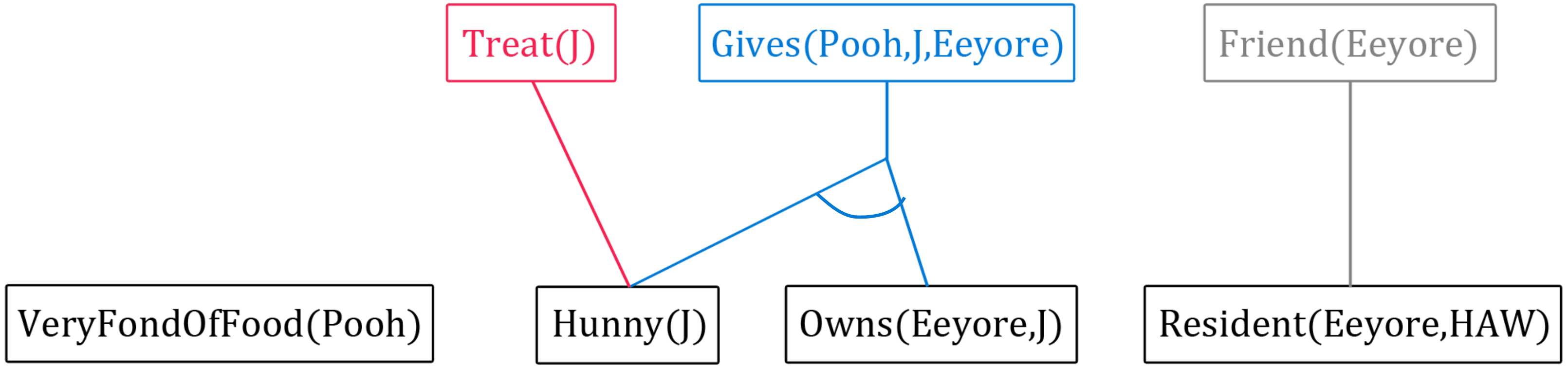
$Hunny(x) \wedge Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

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

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

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Prove that Winnie-the-Pooh is generous

Forward chaining proof

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

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

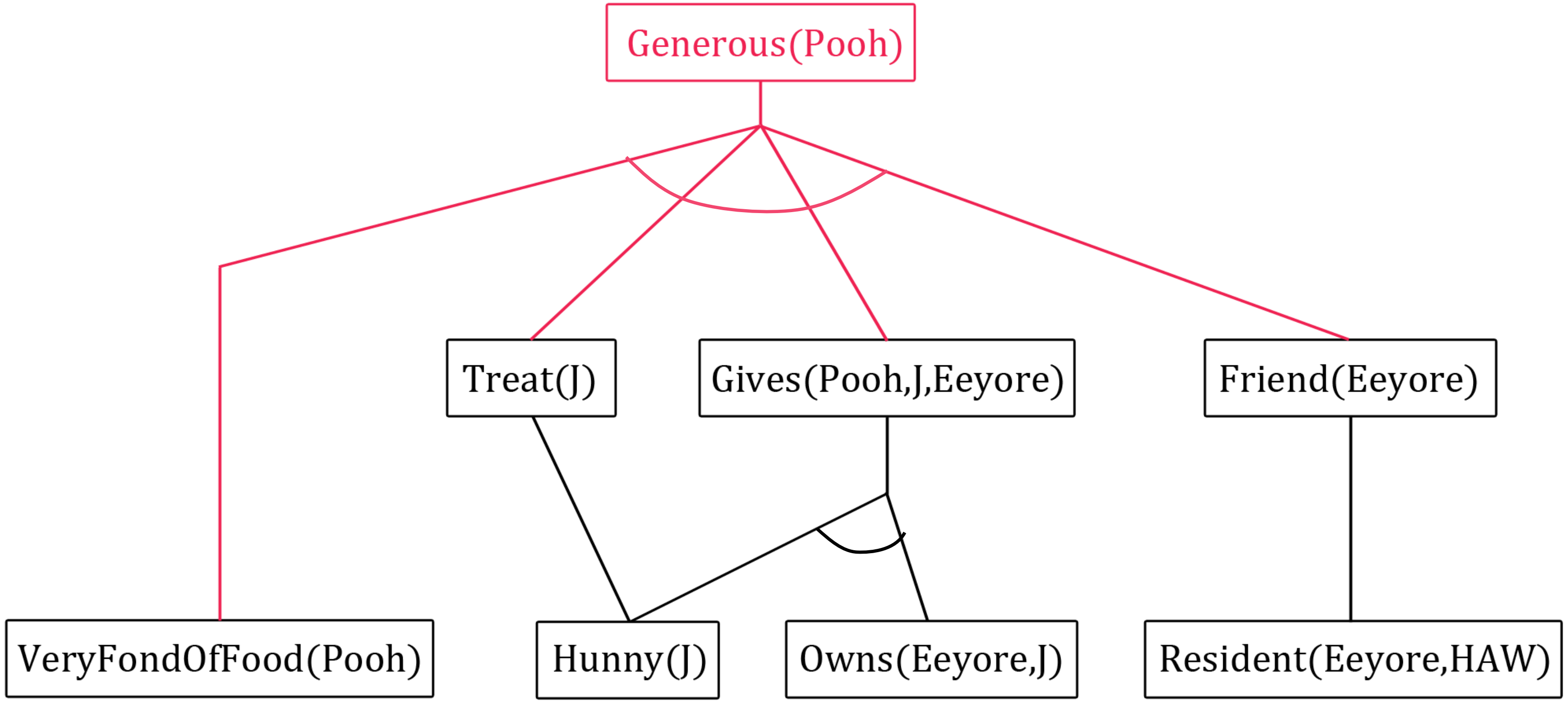
$Hunny(x) \wedge Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

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

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

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Forward chaining 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
    $new \leftarrow \{\}$
    for each $rule$ **in** KB **do**
      $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES($rule$)
      for each θ such that $\text{SUBST}(\theta, p_1 \wedge \ldots \wedge p_n) = \text{SUBST}(\theta, p'_1 \wedge \ldots \wedge p'_n)$
          for some $p'_1, \ldots, p'_n$ in KB ← Pattern-matching
        $q' \leftarrow \text{SUBST}(\theta, q)$
        if q' does not unify with some sentence already in KB or new **then**
          add q' to new ← Facts irrelevant to the goal can be generated
          $\phi \leftarrow \text{UNIFY}(q', \alpha)$
          if ϕ is not *fail* **then return** ϕ
    add new to KB
  return *false*

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
- 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*

  ⇒ 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

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

- Finding all possible unifiers can be very expensive

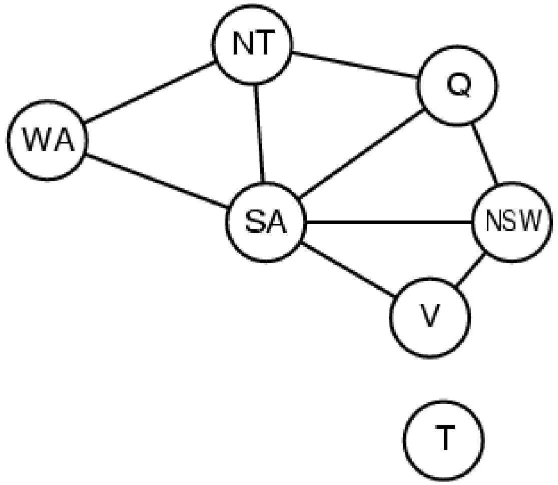
Example:

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

- 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.

Pattern matching and CSPs



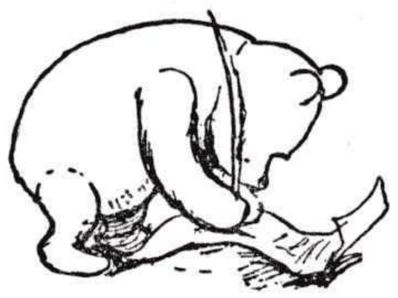
Diff(WA, NT) ∧ Diff(WA, SA) ∧ Diff(NT, Q) ∧
Diff(NT, SA) ∧ Diff(Q, NSW) ∧ Diff(Q, SA) ∧
Diff(NSW, V) ∧ Diff(NSW, SA) ∧ Diff(V, SA)
⇒ *Colorable*

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
- *Colorable* is inferred iff the CSP has a solution
- CSPs include 3SAT as a special case, hence matching is NP-hard

Backward chaining

Prove that Winnie-the-Pooh is generous



Backward chaining proof

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

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

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

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

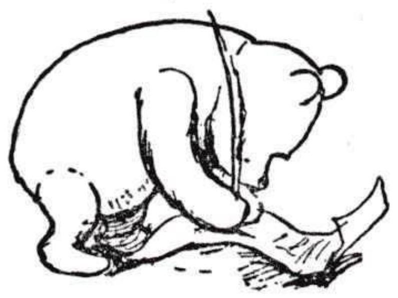
$Resident(x, HAW) \Rightarrow Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$

Generous(Pooh)

Prove that Winnie-the-Pooh is generous



Backward chaining proof

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

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

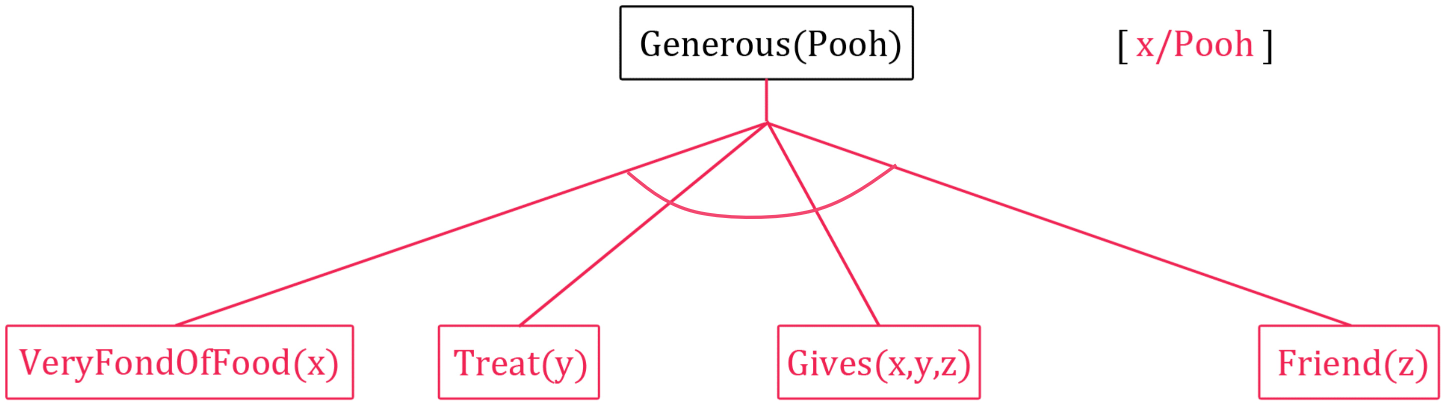
$Hunny(x) \land Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

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

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

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Prove that Winnie-the-Pooh is generous

Backward chaining proof

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

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

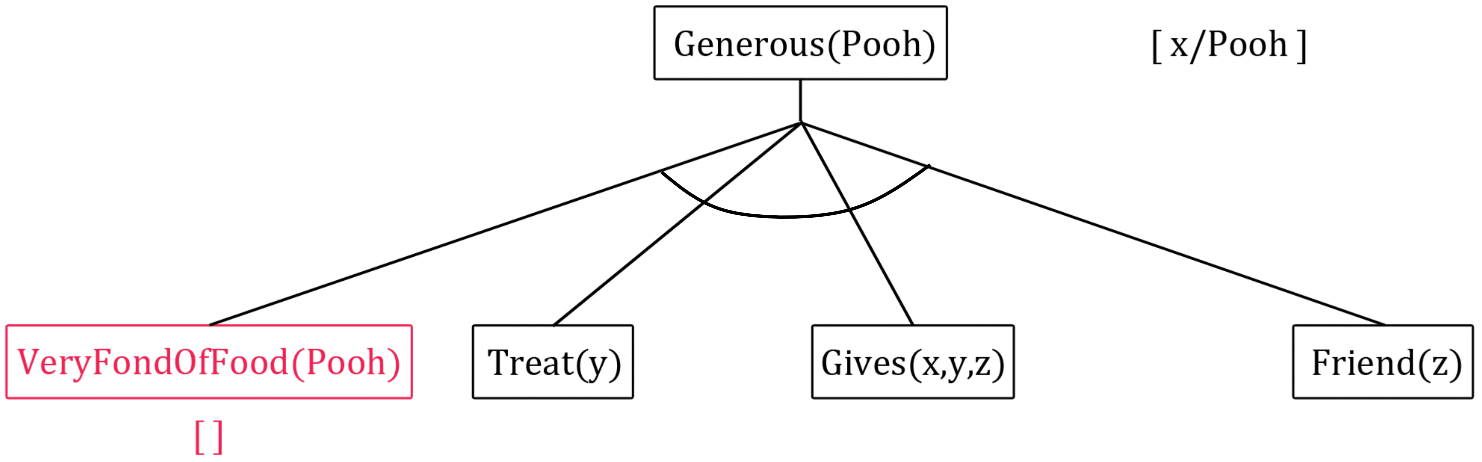
$Hunny(x) \wedge Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

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

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

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Prove that Winnie-the-Pooh is generous

Backward chaining proof

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

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

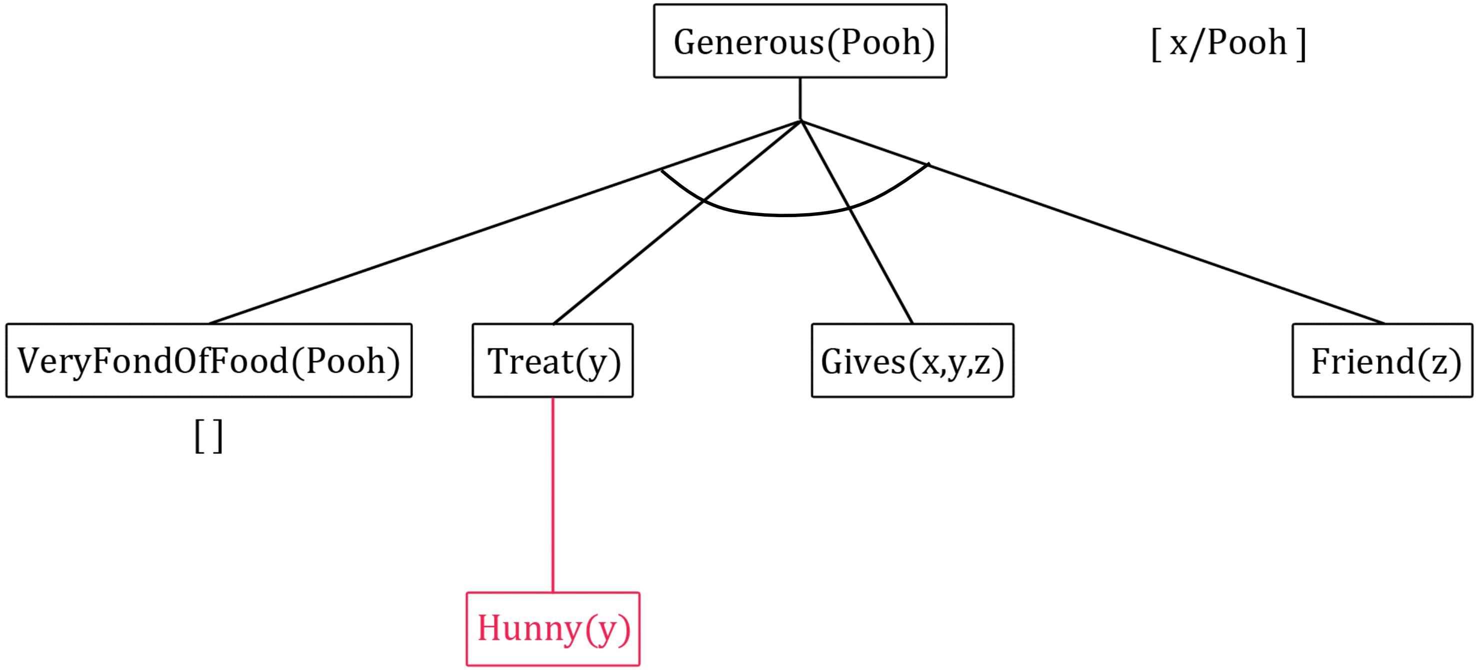
$Hunny(x) \land Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

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

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

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Prove that Winnie-the-Pooh is generous

Backward chaining proof

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

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

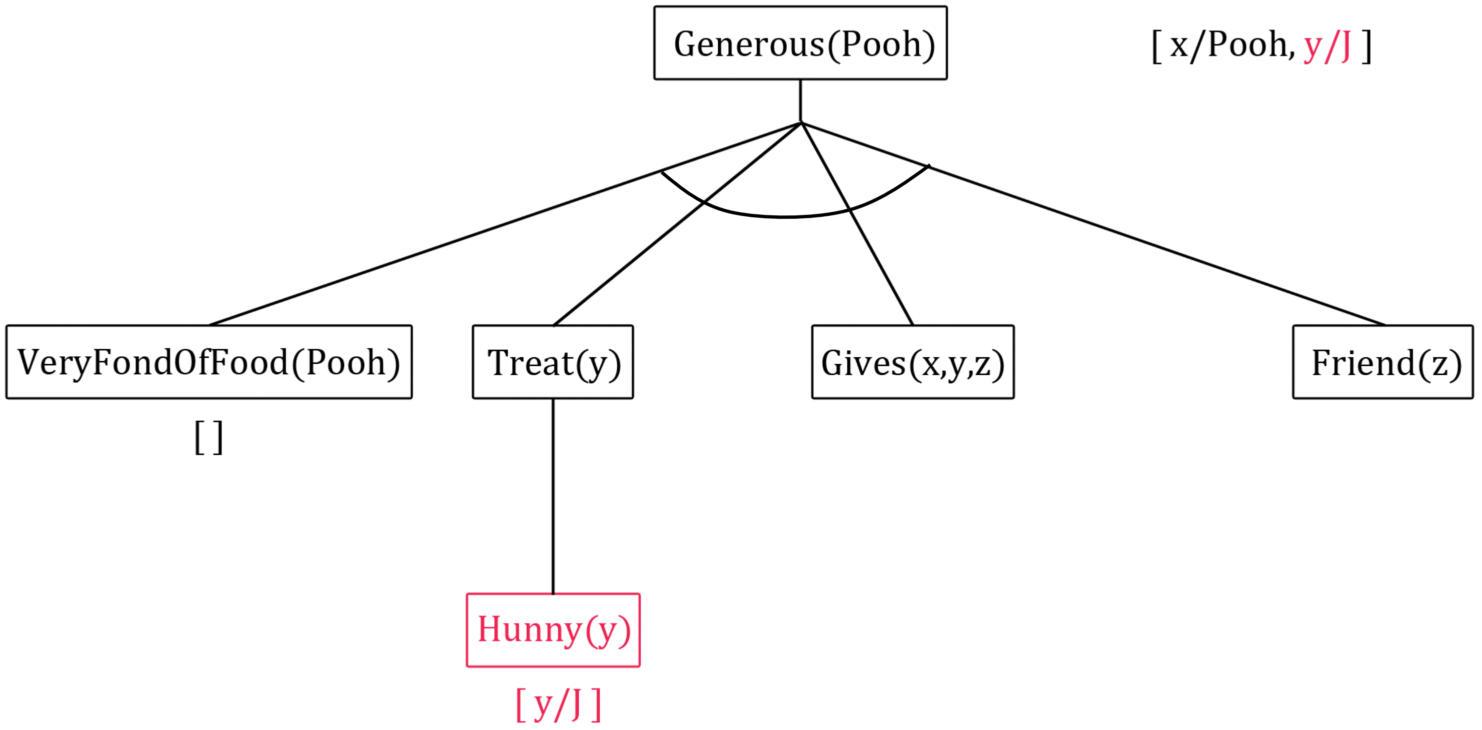
$Hunny(x) \land Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

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

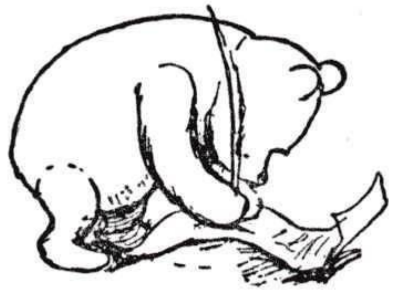
$Resident(x, HAW) \Rightarrow Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Prove that Winnie-the-Pooh is generous



Backward chaining proof

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

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

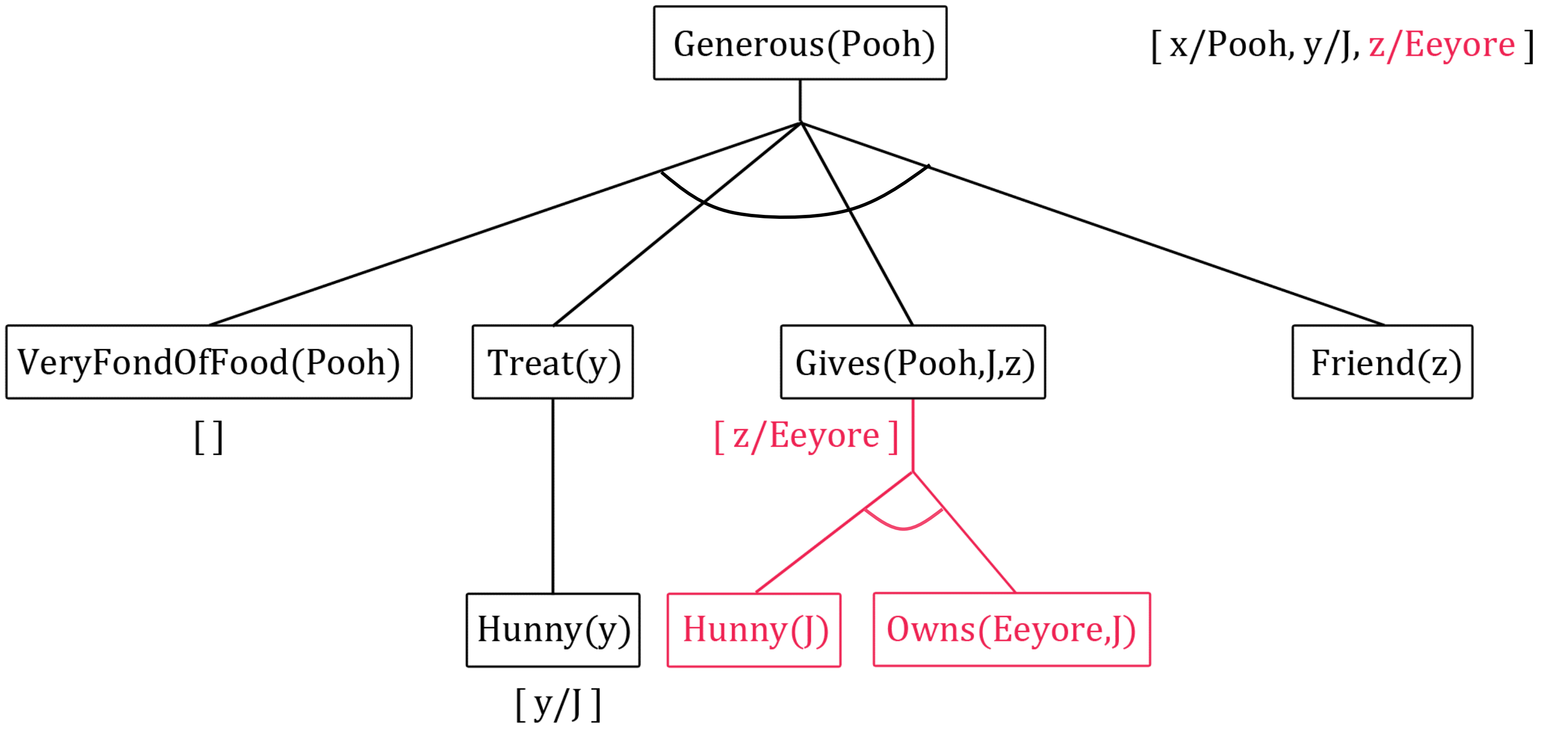
$Hunny(x) \wedge Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

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

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

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Prove that Winnie-the-Pooh is generous

Backward chaining proof

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

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

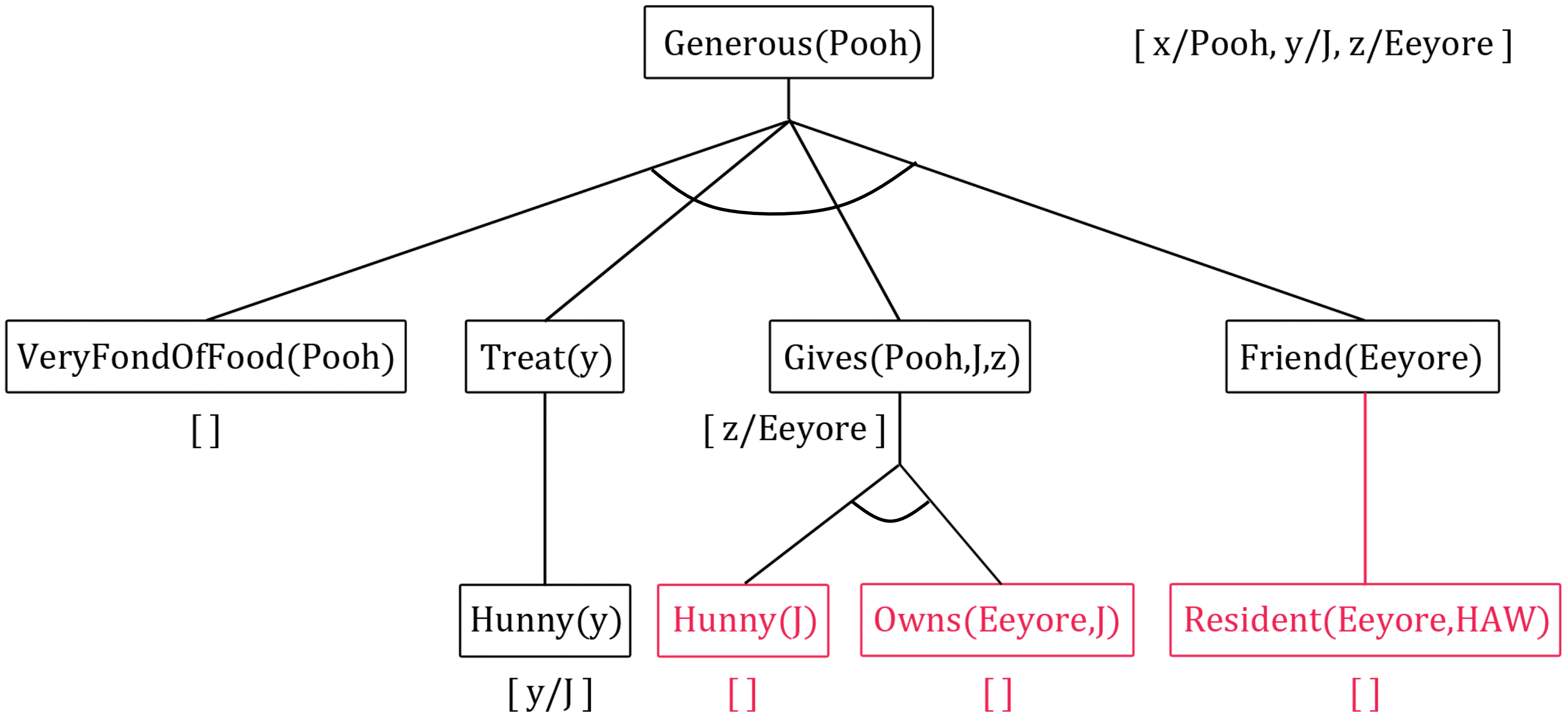
$Hunny(x) \land Owns(Eeyore, x) \Rightarrow Gives(Pooh, x, Eeyore)$

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

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

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$



Backward chaining algorithm

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

Fetch rules that might unify

```
function FOL-BC-ASK(KB, query) returns a gene… …s
   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''
```

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

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

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.

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

Example

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

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

Factoring

$$\frac{C \vee P_1 \vee \cdots \vee P_m}{(C \vee P_1)\theta}$$

where θ 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)

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

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

- $\forall x. \neg\big(\forall y. \neg Animal(y) \lor Loves(x, y)\big) \lor (\exists y. Loves(y, x))$

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

- $\forall x. (\exists y. \neg(\neg Animal(y) \lor Loves(x, y))) \lor (\exists y. Loves(y, x))$
- $\forall x. \big(\exists y. \neg\neg Animal(y) \land \neg Loves(x, y)\big) \lor (\exists y. Loves(y, x))$
- $\forall x. \big(\exists y. Animal(y) \land \neg Loves(x, y)\big) \lor (\exists y. Loves(y, x))$

Standardize variables apart: each quantifier should use a different one

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

Conversion to CNF (2/2)

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

Skolemize: a more general form of existential instantiation

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

Drop universal quantifiers ∀

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

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

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

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

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \land \neg\alpha$
  $new \leftarrow \{\}$
  loop do
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      $resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      $new \leftarrow new \cup resolvents$
    if $new \subseteq clauses$ **then return** *false*
    $clauses \leftarrow clauses \cup new$

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

'Winnie-the-Pooh' Knowledge Base

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

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

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

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

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

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$

‘Winnie-the-Pooh’ Knowledge Base

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

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

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

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

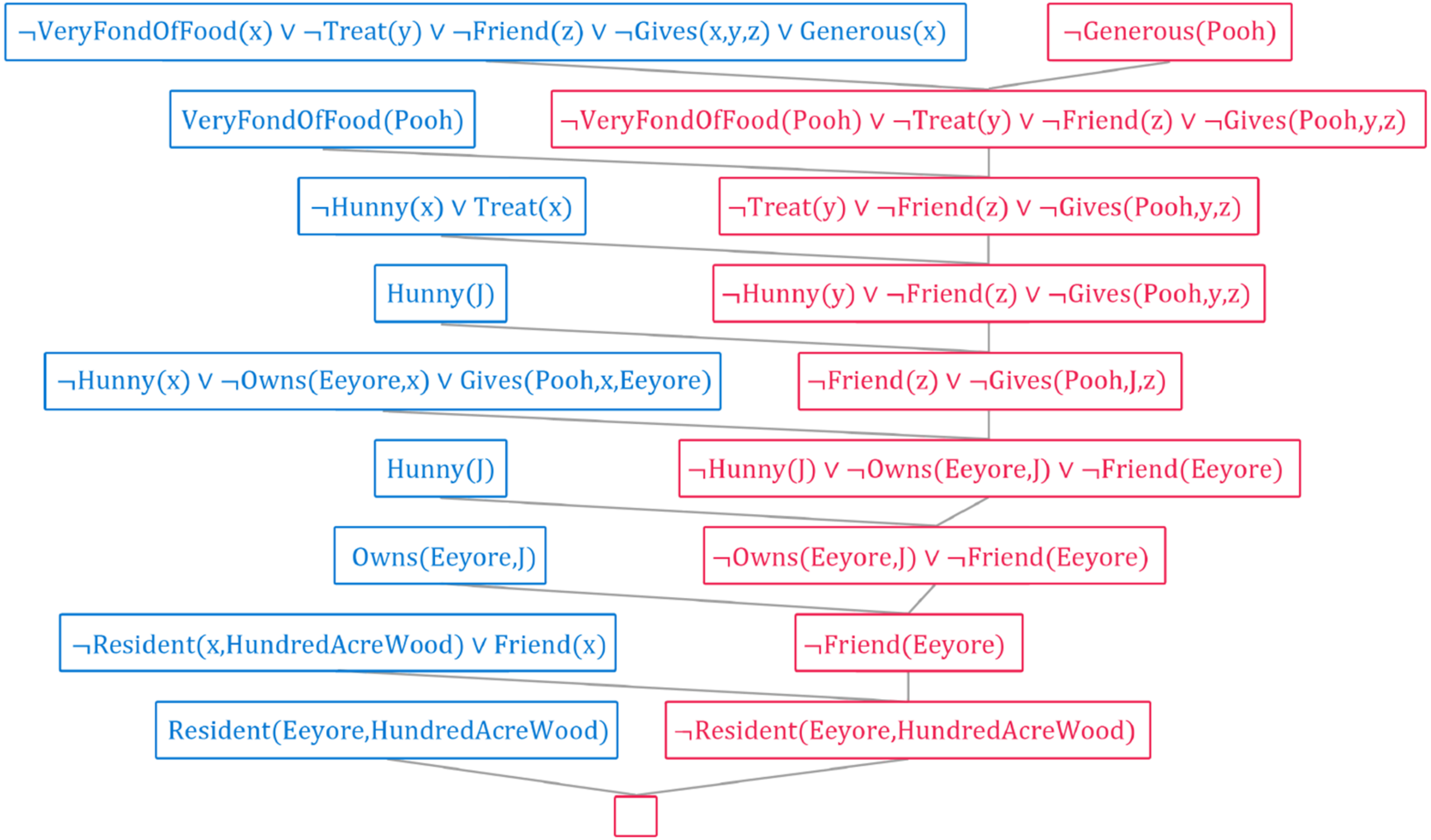
$\neg Resident(x, HAW) \vee Friend(x)$

$Resident(Eeyore, HAW)$

$VeryFondOfFood(Pooh)$

Prove that Winnie-the-Pooh is generous

Resolution proof



Why?

- Winnie-the-Pooh is generous!
- Fundamentals of reasoning in FOL
- Automated logic-based reasoning
- Proof search