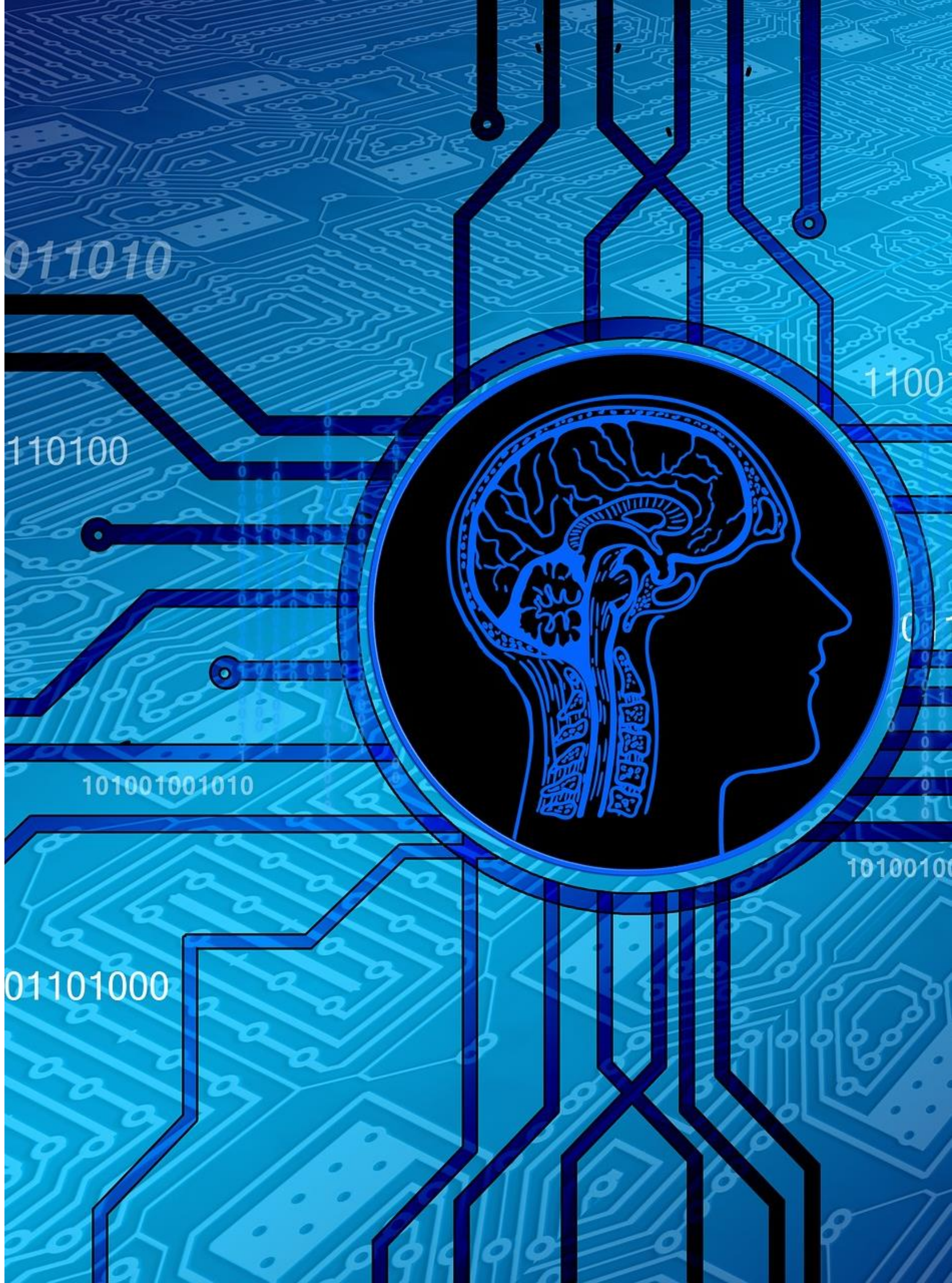
# Unification & Generalised Modus Ponens

Informatics 2D: Reasoning and Agents

**Lecture 11**

***Adapted from slides provided by Dr Petros Papapanagiotou***



# Propositional vs First-Order Inference

- So far, we know how to formulate simple inference rules in FOL.
- Goal: Enabling first-order inference
- Idea:
  - Convert the KB to propositional logic and use propositional inference
- Better idea:
  - Use inference methods to work with first-order sentences directly

# Universal instantiation (UI)

- Infer any sentence by substituting a ground term for the variable

$$\frac{\forall v \quad \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$

Example: $\forall x.\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ yields:

- $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$
- $King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$
- $King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$

# Existential instantiation (EI)

- Replace the variable by a single new constant symbol

$$\frac{\exists v \;\; \alpha}{\textsc{Subst}(\{v/k\}, \alpha)}$$

Example. $\exists x.\ Crown(x) \land OnHead(x, John)$ yields:

- $Crown(C_1) \land OnHead(C_1, John)$

provided $C_1$ is a new constant symbol, called a Skolem constant

# Inferential Equivalence

- UI can be applied many times to produce many different outcomes
- EI can be applied once, then the existentially quantified sentence could be discarded.
- The new knowledge base (KB') is inferentially equivalent to the old KB

# Reduction to propositional inference

- Suppose the KB contains just the following:

$\forall x.\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ $\quad King(John)$ $\quad Greedy(John)$ $\quad Brother(Richard, John)$

- Instantiating the universal sentence in all possible ways, we have:
  - $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$
  - $\underline{King(Richard)} \wedge \underline{Greedy(Richard)} \Rightarrow Evil(Richard)$
  - $King(John)$
  - $Greedy(John)$
  - $Brother(Richard, John)$

# Reduction to propositional inference

- Suppose the KB contains just the following:

$\forall x.\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ $\quad King(John)$ $\quad Greedy(John)$ $\quad Brother(Richard, John)$

- Instantiating the universal sentence in all possible ways, we have:
  - $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$
  - $\underline{King(Richard)} \wedge \underline{Greedy(Richard)} \Rightarrow Evil(Richard)$
  - $King(John)$
  - $Greedy(John)$
  - $Brother(Richard, John)$

KB': The new KB will then include extra propositional symbols

# Propositionalization

- Every FOL KB can be propositionalized so as to preserve entailment
  - A ground sentence is entailed by new KB iff entailed by original KB

- Idea: propositionalize KB and query, apply DPLL (or some other complete propositional method), return result

- Problem: with function symbols, there are infinitely many ground terms,
  - e.g., *Father*(*Father*(*Father*(*John*)))

### Theorem: Herbrand (1930)

- If a sentence α is entailed by a FOL KB, it is entailed by a finite subset of the propositionalized KB

**Idea**: For $n$ = 0 to ∞ do
- create a propositional KB by instantiating with depth-$n$ terms
- see if α is entailed by this KB

**Problem**: works if α is entailed, loops forever if α is not entailed

### Theorem: Turing (1936), Church (1936).

- Entailment for FOL is **semi-decidable**
(i.e., algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every non-entailed sentence.)

A first-order inference rule

# Problems with Propositionalization

$$\forall x.\ King(x) \wedge Greedy(x) \Rightarrow Evil(x) \qquad King(John)$$

$$\forall y.\ Greedy(y) \qquad Brother(Richard, John)$$

- It seems obvious that *Evil*(*John*), but propositionalization produces lots of facts such as *Greedy*(*Richard*) that are irrelevant.

➢ With $p$ $k$-ary predicates and $n$ constants, there are $p \cdot n^k$ instantiations.

➢ We want to find a substitution both for the variables in the implication sentence and for the variables in the sentences in the KB (e.g., *x/John*, *y/John*).

# Modus Ponens (Propositional Logic)

Latin for *"method of putting by placing"* – *"way that affirms by affirming"*

$$\boxed{\dfrac{P \qquad P \Rightarrow Q}{Q}}$$

$$P, \qquad P \Rightarrow Q \vdash Q$$

A first-order inference rule → Generalized Modus Ponens

# Generalized Modus Ponens (GMP)

such that $\text{SUBST}(\theta, p_i{}') = \text{SUBST}(\theta, p_i)$, for all $i$,

$$\frac{p_1{}',\ \ p_2{}',\ \ \ldots,\ \ p_n{}',\ \ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

## KB

$\forall x\ \ King(x) \wedge Greedy(x)\ \Rightarrow\ Evil(x)$
$King(John)$
$\forall y\ \ Greedy(y)$

## Applying GMP to KB

| | |
|---|---|
| $p_1{}'$ is $King(John)$ | $p_1$ is $King(x)$ |
| $p_2{}'$ is $Greedy(y)$ | $p_2$ is $Greedy(x)$ |
| $\theta$ is $\{x/John, y/John\}$ | $q$ is $Evil(x)$ |
| $\text{SUBST}(\theta, q)$ is $Evil(John)$ | |

- GMP is a sound inference rule.

# Unification

MAKE DIFFERENT LOGICAL EXPRESSIONS LOOK IDENTICAL

# Unification

- The `UNIFY` algorithm takes two sentences and returns a unifier for them if one exists.

$$\mathrm{UNIFY}(p, q) = \theta \text{ where } \mathrm{SUBST}(\theta, p) = \mathrm{SUBST}(\theta, q)$$

# Unification examples

| $\alpha$ | $\beta$ | $\theta$ |
|---|---|---|
| *Knows(John, x)* | *Knows(John, Jane)* | |
| *Knows(John, x)* | *Knows(y, OJ)* | |
| *Knows(John, x)* | *Knows(y, Mother(y))* | |
| *Knows(John, x)* | *Knows(x, Richard)* | |

# Unification examples

| $\alpha$ | $\beta$ | $\theta$ |
|---|---|---|
| *Knows(John, x)* | *Knows(John, Jane)* | *{x/Jane}* |
| *Knows(John, x)* | *Knows(y, OJ)* | |
| *Knows(John, x)* | *Knows(y, Mother(y))* | |
| *Knows(John, x)* | *Knows(x, Richard)* | |

# Unification examples

| $\alpha$ | $\beta$ | $\theta$ |
|---|---|---|
| *Knows(John, x)* | *Knows(John, Jane)* | *{x/Jane}* |
| *Knows(John, x)* | *Knows(y, OJ)* | *{x/OJ, y/John}* |
| *Knows(John, x)* | *Knows(y, Mother(y))* | |
| *Knows(John, x)* | *Knows(x, Richard)* | |

# Unification examples

| $\alpha$ | $\beta$ | $\theta$ |
|---|---|---|
| *Knows(John, x)* | *Knows(John, Jane)* | {*x/Jane*} |
| *Knows(John, x)* | *Knows(y, OJ)* | {*x/OJ, y/John*} |
| *Knows(John, x)* | *Knows(y, Mother(y))* | {*y/John, x/Mother(John)*} |
| *Knows(John, x)* | *Knows(x, Richard)* | |

# Unification examples

| $\alpha$ | $\beta$ | $\theta$ |
|---|---|---|
| *Knows(John, x)* | *Knows(John, Jane)* | *{x/Jane}* |
| *Knows(John, x)* | *Knows(y, OJ)* | *{x/OJ, y/John}* |
| *Knows(John, x)* | *Knows(y, Mother(y))* | *{y/John, x/Mother(John)}* |
| *Knows(John, x)* | *Knows(x, Richard)* | **Fail!** |

# Unification examples

| $\alpha$ | $\beta$ | $\theta$ |
|---|---|---|
| *Knows(John, x)* | *Knows(John, Jane)* | {*x/Jane*} |
| *Knows(John, x)* | *Knows(y, OJ)* | {*x/OJ, y/John*} |
| *Knows(John, x)* | *Knows(y, Mother(y))* | {*y/John, x/Mother(John)*} |
| *Knows(John, x)* | *Knows(x, Richard)* | **Fail!** |

Standardizing variables apart eliminates overlap of variables

e.g. change *Knows(x, Richard)* to *Knows*($z_{17}$, *Richard)* and then we succeed the last case with $\theta$ = {$z_{17}$/*John, x/Richard*}

# Most General Unifier (MGU)

Unifying *Knows(John, x)* and *Knows(y, z)*

$$\theta = \{y/John, x/z\} \quad \text{or} \quad \theta = \{y/John, x/John, z/John\}$$

The first unifier is more general than the second.

FOL: There is a **single** most general unifier (MGU) that is unique up to renaming of variables.

$$\text{MGU} = \{y/John, x/z\}$$

Can be viewed as an equation solving problem.

- *i.e. solve* $Knows(John, x) \stackrel{?}{=} Knows(y, z)$

# MGU Examples

| | MGU |
|---|---|
| $Loves(John, x) \stackrel{?}{=} Loves(y, Mother(y))$ | |
| $Loves(John, Mother(y)) \stackrel{?}{=} Loves(y, y)$ | |

# MGU Examples

| | MGU |
|---|---|
| $Loves(John, x) \stackrel{?}{=} Loves(y, Mother(y))$ | $\{x/Mother(John), y/John\}$ |
| $Loves(John, Mother(y)) \stackrel{?}{=} Loves(y, y)$ | |

# MGU Examples

| | MGU |
|---|---|
| $Loves(John, x) \stackrel{?}{=} Loves(y, Mother(y))$ | $\{x/Mother(John), y/John\}$ |
| $Loves(John, Mother(y)) \stackrel{?}{=} Loves(y, y)$ | Fail! |

# Finding the MGU

Can be broken-down into a series of steps

- Decomposition
- Conflict
- Eliminate
- Delete
- Switch
- Coalesce
- Occurs Check

Other presentations of algorithm are possible (see R&N)

# Decomposition

**Given**

$$f(s_1, \ldots, s_n) \stackrel{?}{=} f(t_1, \ldots, t_n)$$

**Replace with**

$$s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n$$

Example

**Given**

$$Knows(John, x) \stackrel{?}{=} Knows(y, z)$$

**Replace with**

$$John \stackrel{?}{=} y, \quad x \stackrel{?}{=} z$$

Given

$f(s_1, \ldots, s_n) \stackrel{?}{=} g(t_1, \ldots, t_n)$ where $f \neq g$

Fail!

# Conflict

Example

Given

$Knows(John, x) \stackrel{?}{=} Greedy(y)$

fail

# Eliminate

**Given**

$P, \quad x \stackrel{?}{=} t$ where $x$ occurs in $P$ but not in $t$, and $t$ is not a variable

**Replace with**

$P\{x/t\}$ and $x \stackrel{?}{=} t$

**Example**

**Given**

$Knows(John, x) \stackrel{?}{=} Knows(y, z), \quad z \stackrel{?}{=} Richard$

**Replace with**

$Knows(John, x) \stackrel{?}{=} Knows(y, Richard), \quad z \stackrel{?}{=} Richard$

Given

$P, \; s \stackrel{?}{=} s$

Replace with

$P$

# Delete

Example

Given

$z \stackrel{?}{=} Richard, \quad Greedy(John) \stackrel{?}{=} Greedy(John)$

Replace with

$z \stackrel{?}{=} Richard$

# Switch

Given

$P, \ s \overset{?}{=} x$ *where* $x$ *is a variable and* $s$ *is not*

Replace with

$P$ *and* $x \overset{?}{=} s$

Example

Given

$Knows(John, \ x) \overset{?}{=} Knows(y, \ z), \quad Richard \overset{?}{=} z$

Replace with

$Knows(John, \ x) \overset{?}{=} Knows(y, \ z), \quad z \overset{?}{=} Richard$

# Coalesce

**Given**

$P, \; x \stackrel{?}{=} y$ *where* $x, y$ *variables occurring in* $P$

**Replace with**

$P\{x/y\}$ *and* $x \stackrel{?}{=} y$

Example

**Given**

$Knows(John, \; x) \stackrel{?}{=} Knows(y, \; z), \quad y \stackrel{?}{=} z$

**Replace with**

$Knows(John, \; x) \stackrel{?}{=} Knows(z, \; z), \quad y \stackrel{?}{=} z$

# Occurs Check

Given

$x \stackrel{?}{=} s$ *where* $x$ ***occurs*** *in* $s$ *and* $s$ *not a variable*

*Fail!*

Example

Given

$P(x), \quad x \stackrel{?}{=} Father(x)$

*Fail (else Eliminate will loop)*

$P(Father(Father(Father(\ldots))))$

# Example

$$Loves(John, x) \stackrel{?}{=} Loves(y, Mother(y))$$

Decompose

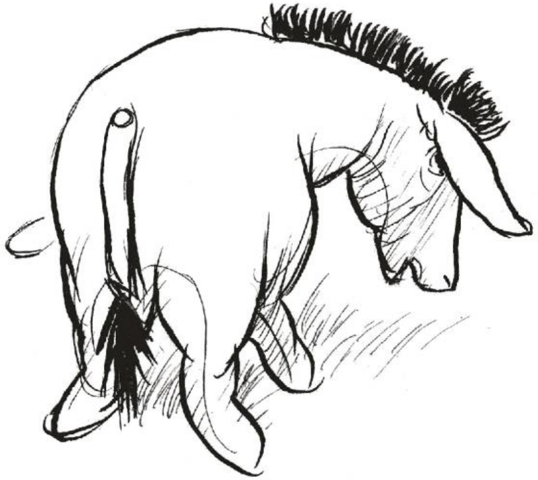
$$John \stackrel{?}{=} y,\ \ x \stackrel{?}{=} Mother(y)$$

Switch

$$y \stackrel{?}{=} John,\ \ x \stackrel{?}{=} Mother(y)$$

Eliminate

$$y \stackrel{?}{=} John,\ \ x \stackrel{?}{=} Mother(John)$$



# New Example KB

# Example Knowledge Base

*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.*

# Formalisation

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

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

*Eeyore (…) has some hunny*

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

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

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

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.

- $VeryFondOfFood(Pooh)$

# Why?

- Setting the scene for inference & resolution.
- Linked to logic programming.

*…but more in the next lecture!*