4CCS1ELA ELEMENTARY LOGIC WITH APPLICATIONS

LECTURE 9:
PREDICATE DEFINITE CLAUSE PROGRAMMING

Objectives for the day

- Consolidate Conjunctive Normal Form and Programming with Propositional Definite Clause Rules (last week)

- Learn about and work on Prenex Normal Form
- Learn about and work on **Predicate** Definite Clause Rules
- Learn about and work on Programming with Predicate Definite Clause Rules

Revision of last week

Conjunctive Normal Form (CNF)

- CNF is a **conjunction** of **one** or more formulas, each of which is a **disjunction** of **one** or more literals

$$(A \lor B) \land (C \lor D)$$

- $A \lor B$ is in CNF
  - Because… "CNF is a **conjunction** of **one** or more **formulas**,…"
    - So this one has **one formula** with two literals
- $A \land B$ is in CNF
  - Because it is "… a **disjunction** of **one** or more **literals** ".
    - So this one has two disjunctions of **one literal**
- A is in CNF
  - Because it is a disjunction of **one literal** in a conjunction that has **one formula**.

Clause

- A **clause** is a disjunction of one or more literals.

- So let's rephrase the CNF definition:
  - CNF is a **conjunction** of one or more formulas, each of which is a **disjunction** of one or more literals. Or:
- CNF is a **conjunction** of one or more **clauses**!

Let's count the clauses in our KB

CNF is a **conjunction** of one or more **clauses** (disjunctions of 1 or more literals)

KB1

1. P
2. $(\neg P \vee Q) \wedge (\neg P \vee R)$
3. $\neg Q \vee S$
4. $\neg S \vee \neg R \vee T$

- CNF 1., 3., and 4. have one clause
- CNF 2. has 2 clauses:
  $(\neg P \vee Q)$ and $(\neg P \vee R)$

Horn clauses and Definite clauses

1. P	1 clause
2. $(\neg P \vee Q) \wedge (\neg P \vee R)$	2 clauses
3. $\neg Q \vee S$	1 clause
4. $\neg S \vee \neg R \vee T$	1 clause

- **A Horn clause is a clause with no more than one positive literal**
  - 0 or 1 positive literals plus 0 to many negatives
- **A definite clause is a Horn clause with exactly one positive literal.**
  - exactly1 positive literal plus 0 to many negatives

How many definite clauses do you see in those CNFs?

1) $P \lor Q$
2) $P \land Q$
3) P

How many definite clauses do you see in those CNFs?

1) $P \vee Q$
2) $P \wedge Q$
3) P

CNF 1 has one clause
CNF 2 has two clauses
CNF 3 has one clause

- Are all these four clauses definite clauses?
- $P \vee Q$ is not definite (two positive literals!)
- The rest are definite clauses!

From definite clauses to **DEFINITE RULES**

- A definite clause of the form

$\neg X_1 \vee \ldots \vee \neg X_m \vee X$

can be represented as the equivalent:

$X_1 \wedge \ldots \wedge X_m \rightarrow X$

we replace the $\wedge$ with **,**

$X_1 , \ldots , X_m \rightarrow X$

How to transform propositional logic to definite rules to programs

1. Transform to CNF, using equivalences
2. Identify definite clauses (not all clauses are definite!)
3. Represent the definite clauses as definite rules

Question [3.1] from LGT worksheet 6 (last week)

$\neg P \rightarrow (\neg Q \land R)$

1. transform it into CNF using equivalence laws (see next slide).

$\neg\neg P \lor (\neg Q \land R)$ (we applied law 1)

$P \lor (\neg Q \land R)$ (we applied law 5)

$(P \lor \neg Q) \land (P \lor R)$ (we applied law 6)

2. identify definite clauses. Not all clauses are definite clauses (a definite clause has exactly one positive literal)!

$(P \lor \neg Q) \land (P \lor R)$

$(P \lor \neg Q)$ is a definite clause

$(P \lor R)$ is not (two positive literals!)

3. we represent the definite clause as definite rules.

$(P \lor \neg Q)$

can be represented as rule **$Q \rightarrow P$**

Question [4] from LGT worksheet 6 (last week)

$\neg(R \rightarrow \neg T)$

1. transform it into CNF using equivalences.

$\neg(\neg R \vee \neg T)$

$\neg\neg R \wedge \neg\neg T$

$R \wedge T$

2. identify definite clauses.

R and *T* are two definite clauses.

3. represent the definite clauses as definite rules.

R can be represented as the rule $\rightarrow$***R***

T can be represented as the rule $\rightarrow$***T***

Equivalences

- Use equivalence rules until you get to CNF

1) $F \rightarrow G \equiv \neg F \vee G$
2) $F \leftrightarrow G \equiv (F \rightarrow G) \wedge (G \rightarrow F)$
3) $\neg(F \vee G) \equiv \neg F \wedge \neg G$
4) $\neg(F \wedge G) \equiv \neg F \vee \neg G$
5) $\neg\neg F \equiv F$
6) $F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$
7) $(F \wedge G) \vee H \equiv (F \vee H) \wedge (G \vee H)$

(F,G and H are propositional formulas and not literals)

Predicate Logic recap

Predicate logic notation

- Upper case P, R, Q … denote **predicates**

- Lower case at end of alphabet - t,u,v,w,x,y,z – denote **variables**
- a,b,c,d and lower case *strings* of more than one letter (e.g., $lela$, $mary$) denote **constants**.

- A **term** is either a **variable**, or a **constant**, or a **function** symbol applied to arguments that are terms

Predicate logic notation

- An **atomic formula**, or atom, is a predicate symbol applied to arguments that are terms.
  - e.g., $P(x)$, $P(a)$, $Loves(mary,y)$, $Q(f(x),a)$
  - Sometimes we write "$loves(mary,y)$" instead of "$Loves(mary,y)$" – obvious to see that although "loves" starts with a lower case letter it is a predicate symbol
- Sometimes we use F, G, H to refer to any predicate formula e.g., F denotes $P(x)$, $P(x) \rightarrow Q(y)$, $\neg \forall x\, R(x, y)$ …..

Prenex Normal Form

Prenex Normal Form

Definition

- A formula is in **prenex normal form** if it starts with **0 or more quantifiers** followed by a **formula with no quantifiers**.

  $$Q_1 x_1 \dots Q_n x_n \, F$$

  where Q_i (i = 0,…n) is $\forall$ or $\exists$ and the formula F has no quantifiers.

- $Q_1 x_1 \dots Q_n x_n$ is called the *prefix* and it may even be empty!
- F is called the *matrix*

- Any first order (predicate) wff can be transformed to *prenex normal form (PNF)*

- We will then see that some *PNF* formulas give us *first order definite clauses/rules* that we can program with in a simple way

Exercise

- Which of the following are in Prenex Normal Form, and why?

1) $\forall x\, P(x) \vee \forall x\, Q(x)$
2) $\forall x \forall y\, \neg(P(x) \rightarrow Q(y))$
3) $\forall x \exists y\, R(x, y)$
4) $R(x, y)$
5) $\neg \forall x\, R(x, y)$
6) $loves(mary, john)$

A formula is in **prenex normal form** if it starts with **0 or more quantifiers** followed by a **formula with no quantifiers**.

Exercise

- Which of the following are in **Prenex Normal Form**, and why?

1) $\forall x\, P(x) \lor \forall x\, Q(x)$ *(quantifier… formula with quantifier)*
2) $\forall x \forall y\, \neg(P(x) \rightarrow Q(y))$ *(quantifier… f. with* ***no*** *quantifier)*
3) $\forall x \exists y\, R(x, y)$ *(quantifier… f. with* ***no*** *quantifier)*
4) $R(x, y)$ *(no quantifier… f. with* ***no*** *quantifier)*
5) $\neg \forall x\, R(x, y)$ *(f. with quantifier)*
6) $loves(mary, john)$ *(no quantifier… f. with* ***no*** *quantifier)*

A formula is in **prenex normal form** if it starts with **0 or more quantifiers** followed by a **formula with no quantifiers**.

Algorithm to **transform a formula into PNF**

1. Remove $\rightarrow$ and $\leftrightarrow$
2. Move negations inward

   - such that, in the end, negations only appear in front of atoms.
3. Rename variables so that variables of each quantifier are unique, if necessary (this is called standardisation).
4. Move quantifiers to the front of the formula

   - preserving order of quantifiers.

Steps 1 and 2

1. Remove $\rightarrow$ and $\leftrightarrow$
2. Move negations inward

To perform steps 1 and 2, use equivalences!

1. To remove $\rightarrow$ and $\leftrightarrow$ use equivalences:

$F \rightarrow G \equiv \neg F \vee G$

$F \leftrightarrow G \equiv (\neg F \vee G) \wedge (F \vee \neg G)$

$F \leftrightarrow G \equiv (F \wedge G) \vee (\neg F \wedge \neg G)$

2. To move $\neg$ inwards use equivalences:

$\neg(F \vee G) \equiv \neg F \wedge \neg G$

$\neg(F \wedge G) \equiv \neg F \vee \neg G$

$\neg\neg F \equiv F$

$\neg \exists x F \equiv \forall x \neg F$

$\neg \forall x F \equiv \exists x \neg F$

Step 3

3. Rename variables so that variables of each quantifier are unique, if necessary (this is called standardisation).

- Example:

$$\forall x \, (P(x) \rightarrow Q(x)) \wedge \exists x \, R(x)$$

Step 3

3. Rename variables so that variables of each quantifier are unique, if necessary (this is called standardisation).

- Example:

$\forall \boldsymbol{x}(P(\boldsymbol{x}) \rightarrow Q(\boldsymbol{x})) \wedge \exists \boldsymbol{x}\, R(\boldsymbol{x})$

- Rename $\boldsymbol{x}$ in $\exists \boldsymbol{x}\, R(\boldsymbol{x})$ to something else:

$\forall \boldsymbol{x}(P(\boldsymbol{x}) \rightarrow Q(\boldsymbol{x})) \wedge \exists \boldsymbol{y} R(\boldsymbol{y})$

Step 3

- Example:

$$\forall x\, P(x) \rightarrow Q(x) \wedge R(x)$$

- Do you need to rename any variable?
- No! The x in $P(x) \rightarrow Q(x)$ and $R(x)$ are not unique variables as they are in the scope of the same quantifier $\forall x$

Exercise

- Rename variables in:

$$\forall z\,((P(z) \to Q(z)) \land R(z)) \land \exists z\, S(z)$$

$$\forall z\,((P(z) \to Q(z)) \land R(z)) \land \exists z\, S(z)$$

$$\forall \boldsymbol{z}\,((P(\boldsymbol{z}) \to Q(\boldsymbol{z})) \land R(\boldsymbol{z})) \land \exists \boldsymbol{z}\, S(\boldsymbol{z})$$

$$\forall \boldsymbol{z}\,((P(\boldsymbol{z}) \to Q(\boldsymbol{z})) \land R(\boldsymbol{z})) \land \exists \boldsymbol{y}\, S(\boldsymbol{y})$$

Step 4

4. Move quantifiers to the front of the formula

To perform step 4, we use equivalences again!

- $F \wedge \exists x G \equiv \exists x\, (F \wedge G)$
- $F \wedge \forall x G \equiv \forall x\, (F \wedge G)$
- $F \vee \exists x G \equiv \exists x\, (F \vee G)$
- $F \vee \forall x G \equiv \forall x\, (F \vee G)$

  x not occurring in F

Step 4

- More equivalences:

$$Q_1 x\, F \wedge Q_2 y\, G \equiv Q_1 x\, Q_2 y\, (F \wedge G)$$

x not occurring in G

y not occurring in F

$$Q_1 x\, F \vee Q_2 y\, G \equiv Q_1 x\, Q_2 y\, (F \vee G)$$

x not occurring in G

y not occurring in F

, where Q_1 Q_2 $\in \{\forall, \exists\}$.

Step 4

- More equivalences:

$Q_1 x\, F \wedge Q_2 y\, G \equiv Q_1 x\, Q_2 y\, (F \wedge G)$

 x not occurring in G

 y not occurring in F

$Q_1 x\, F \vee Q_2 y\, G \equiv Q_1 x\, Q_2 y\, (F \vee G)$

 x not occurring in G

 y not occurring in F

, where $Q_1\ Q_2 \in \{\forall, \exists\}$.

- For example, if $Q_1 = \forall$ and $Q_2 = \exists$:

$\forall x\, A(x) \wedge \exists y\, B(y) \equiv$

$\forall x \exists y\, A(x) \wedge B(y)$

Note about PNF vs transforming to PNF

- A PNF formula is any formula that starts with 0 or more quantifiers followed by a **formula with no quantifiers**
- That formula may contain $\rightarrow$
- *Example:*

  $\forall x \, \forall y \, \neg(P(x) \rightarrow Q(y))$ - from slide 21

- But converting a first order formula involves removing $\rightarrow$ so the PNF formulas obtained by transformation will not contain $\rightarrow$

Algorithm to **transform a formula into PNF**

1. Remove $\rightarrow$ and $\leftrightarrow$
2. Move negations inward

   - such that, in the end, negations only appear in front of atoms.
3. Rename variables so that variables of each quantifier are unique, if necessary (this is called standardisation).
4. Move quantifiers to the front of the formula

   - preserving order of quantifiers as they appear in formula.

Example: transforming to PNF

$\forall x\ ($

$\exists y\ R(x, y) \wedge \exists z\ \neg\ S(x, z) \longrightarrow \neg \exists y\ P(x, y)$

$)$

Steps 1 and 2

1. Remove $\rightarrow$ and $\leftrightarrow$
2. Move negations inward

To perform steps 1 and 2, use equivalences!

1. To remove $\rightarrow$ and $\leftrightarrow$ use equivalences:
   1) $F \rightarrow G \equiv \neg F \vee G$
   2) $F \leftrightarrow G \equiv (\neg F \vee G) \wedge (F \vee \neg G)$
   3) $F \leftrightarrow G \equiv (F \wedge G) \vee (\neg F \wedge \neg G)$

2. To move $\neg$ inwards use equivalences:
   1) $\neg(F \vee G) \equiv \neg F \wedge \neg G$
   2) $\neg(F \wedge G) \equiv \neg F \vee \neg G$
   3) $\neg\neg F \equiv F$
   4) $\neg \exists x F \equiv \forall x \neg F$
   5) $\neg \forall x F \equiv \exists x \neg F$

Step 1

1. Remove $\rightarrow$ and $\leftrightarrow$

Use equivalence:

$F \rightarrow G \equiv \neg F \vee G$

Remember: $\forall x$ quantifies over the whole formula.

$\forall x$

$(\exists y\, R(x, y) \wedge \exists z\, \neg\, S(x, z) \rightarrow \neg \exists y\, P(x, y))$

$\equiv$

$\forall x$

$\neg(\exists y\, R(x, y) \wedge \exists z\, \neg S(x, z)) \vee (\neg \exists y\, P(x, y))$

Step 2

2. Move negations inward

$\neg(F \wedge G) \equiv \neg F \vee \neg G$

$\forall x$

$\neg(\exists y\, R(x, y) \wedge \exists z\, \neg S(x, z)) \vee (\neg \exists y\, P(x, y))$

$\equiv$

$\forall x$

$(\neg \exists y\, R(x, y) \vee \neg \exists z\, \neg S(x, z)) \vee (\neg \exists y\, P(x, y))$

Step 2

2. Move negations inward

$\neg \exists x F \equiv \forall x \neg F$

$\forall x$

$(\neg \exists y\ R(x, y) \lor \neg \exists z\ \neg S(x, z)) \lor (\neg \exists y\ P(x, y))$

$\equiv$

$\forall x$

$(\forall y\ \neg R(x, y) \lor \forall z\ \neg\neg S(x, z)) \lor (\forall y\ \neg P(x, y))$

Step 2

2. Move negations inward

$$\neg\neg F \equiv F$$

$$\forall x$$

$$(\forall y\ \neg R(x, y) \lor \forall z\ \neg\neg S(x, z)) \lor (\forall y\ \neg P(x, y))$$

$$\equiv$$

$$\forall x$$

$$(\forall y\ \neg R(x, y) \lor \forall z\ S(x, z)) \lor (\forall y\ \neg P(x, y))$$

Step 3

3. Rename variables so that variables of each quantifier are unique, if necessary (this is called standardisation).

$\forall x$

$(\forall y\ \neg R(x, y) \lor \forall z\ S(x, z)) \lor (\forall y\ \neg P(x, y))$

Step 3

3. Rename variables so that variables of each quantifier are unique, if necessary (this is called standardisation).

$\forall x$

$(\forall y\ \neg R(x, y) \vee \forall z\ S(x, z)) \vee (\forall \mathbf{y}\ \neg P(x, \mathbf{y}))$

$\equiv$

$\forall x$

$(\forall y\ \neg R(x, y) \vee \forall z\ S(x, z)) \vee (\forall \mathbf{w}\ \neg P(x, \mathbf{w}))$

Step 4

4. Move quantifiers to the front of the formula – preserve the order

$\forall x$

$(\forall y\ \neg R(x, y) \vee \forall z\ S(x, z)) \vee (\forall w\ \neg P(x, w))$

$\equiv$

$\forall x\ \forall y\ \forall z\ \forall w$

$\neg R(x, y) \vee S(x, z) \vee \neg P(x, w)$

In PNF!

PREFIX

$\forall x\ \forall y\ \forall z\ \forall w$

$\neg R(x, y) \vee S(x, z) \vee \neg P(x, w)$

MATRIX

Does the matrix remind you of anything?

Predicate Horn and Definite Clauses

Horn clause

- A clause in the *matrix* of a PNF formula is a first order (predicate) **Horn** clause if:
    1) the *prefix* consists only of **universal** quantifiers quantifying over **all** variables in the clause
    2) the clause consists of a finite disjunction of positive or negative atoms (a negative atom is an atom preceded by ¬), **with no more than one positive atom**

$\forall x\ \forall y\ \forall z\ \forall w$ *(1) is met)*

$\neg R(x, y) \lor S(x, z) \lor \neg P(x, w)$ *(2) is met)*

- This is a Horn clause

Definite clause

- A first order Definite clause is a Horn clause with **exactly one positive** atom and 0 or more negative atoms

$\forall x\ \forall y\ \forall z\ \forall w$

$\neg R(x, y) \vee S(x, z) \vee \neg P(x, w)$ *(exactly one positive)*

- This is also a definite clause!

Definite Clauses to **Definite RULES**

- A first order (FO) **definite clause** is of the form:

$$\forall x_1,\ldots,\forall x_n \;\; \neg\alpha_1 \vee \ldots \vee \neg\alpha_m \vee \alpha$$

  - $m \geq 0$
  - each α_i is an *atom*.
  - $x_1,\ldots, x_n$ are all the variables in $\neg\alpha_1 \vee \ldots \vee \neg\alpha_m \vee \alpha$

- A FO definite clause can be represented as a FO **definite rule**:

$$\forall x_1,\ldots,\forall x_n \;\; \alpha_1 \wedge \ldots \wedge \alpha_m \rightarrow \alpha$$

- and then dropping $\wedge$ as we did with propositional definite rules:

$$\forall x_1,\ldots,\forall x_n \;\; \alpha_1 ,\ldots , \alpha_m \rightarrow \alpha$$

Definite Rules

Transforming PNF formulas to Definite Rules

- The matrix of a PNF formula can be positive or negative atoms joined by the connectives $\wedge$, $\vee$

- Like in our example:

$$\forall x\ \forall y\ \forall z\ \forall w \quad \neg R(x, y) \vee S(x, z) \vee \neg P(x, w)$$

- But we could also have:

$$\forall x\ \forall y\ \forall z\ \neg R(x, y) \vee (S(x, z) \wedge P(x,z))$$

Transforming PNF formulas to Definite Rules

- So first transform matrix of a PNF formula into first order **Conjunctive Normal Form**

- Since the matrix of the PNF formula is already in the form of positive or negative atoms joined by the connectives ∧ , ∨ , we can just use rules for **distributing** over **∧ , ∨** …
- …Just like when we transform a propositional formula into conjunctive normal form

1) $F \lor (G \land H) \equiv (F \lor G) \land (F \lor H)$
2) $(F \land G) \lor H \equiv (F \lor H) \land (G \lor H)$

Transform into CNF!

$$F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$$

$\forall x\ \forall y\ \forall z$

$\neg R(x, y) \vee (S(x, z) \wedge P(x,z))$

$\equiv$

$\forall x\ \forall y\ \forall z$

$(\neg R(x, y) \vee S(x, z)) \wedge (\neg R(x, y) \vee P(x,z))$

Procedure of transformation of predicate logic formulas to definite rules

1) Transform predicate formula into **PNF** formula F

2) Transform matrix of F to **CNF**

3) If

   i) every quantifier in prefix is **universal**

   ii) every clause in CNF of matrix contains **exactly one positive atom**, **0 or more negative atoms** and all variables in the clause are in the scope of a universal quantifier in the prefix (which means that each clause is a **definite clause**)

Then represent each clause as a **definite rule**

Let us continue with our example

$$\underbrace{\forall x\ \forall y\ \forall z\ \forall w}_{\text{Prefix}}\quad \underbrace{\neg R(x, y) \vee S(x, z) \vee \neg P(x, w)}_{\text{Matrix}}$$

- *Remember: PNF → CNF → Definite Clauses → Definite Rules*

- The formula is in PNF
- Matrix is in CNF

Definite Clauses to **Definite RULES**

- **A definite clause:**

$$\forall x_1, \ldots, \forall x_n \;\; \neg \alpha_1 \vee \ldots \vee \neg \alpha_m \vee \alpha$$

- **Represented as definite rule:**

$$\forall x_1, \ldots, \forall x_n \;\; \alpha_1 \wedge \ldots \wedge \alpha_m \rightarrow \alpha$$

- Now drop $\wedge$:

$$\forall x_1, \ldots, \forall x_n \;\; \alpha_1 , \ldots , \alpha_m \rightarrow \alpha$$

Let us continue with our example

$\forall x\ \forall y\ \forall z\ \forall w \quad \neg R(x, y) \lor S(x, z) \lor \neg P(x, w)$

- *Remember: PNF → CNF → Definite Clauses → Definite Rules*

- The matrix has one definite clause:

$\forall x\ \forall y\ \forall z\ \forall w \quad \neg R(x, y) \lor \neg P(x, w) \lor S(x, z)$

- The definite clause can be represented as:

$\forall x\ \forall y\ \forall z\ \forall w \quad R(x, y) \land P(x, w) \rightarrow S(x, z)$

- Drop $\land$

$\forall x\ \forall y\ \forall z\ \forall w \quad R(x, y)\ ,\ P(x, w) \rightarrow S(x, z)$

Another example

The example we worked on slide 48

- *Remember: PNF → CNF → Definite Clauses → Definite Rules*

$$\underbrace{\forall x\ \forall y\ \forall z}_{\text{Prefix}} \quad \underbrace{\neg R(x, y) \lor (S(x, z) \land P(x, z))}_{\text{Matrix}}$$

- PNF but matrix not in CNF

Another example

The example we worked on slide 48

- *Remember: PNF → CNF → Definite Clauses → Definite Rules*

$$\forall x\ \forall y\ \forall z\ \neg \mathbf{R(x, y)} \vee (\mathbf{S(x, z)} \wedge \mathbf{P(x, z)})$$

$$\equiv$$

$$\forall x\ \forall y\ \forall z\ (\neg \mathbf{R(x, y)} \vee \mathbf{S(x, z)}) \wedge (\neg \mathbf{R(x, y)} \vee \mathbf{P(x, z)})$$

Now matrix in CNF *(applied distributivity law)*

Another example

- *Remember: PNF → CNF → Definite Clauses → Definite Rules*

$\forall x\ \forall y\ \forall z\ \ (\neg R(x, y) \vee S(x, z)) \wedge (\neg R(x, y) \vee P(x, z))$

- The matrix has two definite clauses:

$\forall x\ \forall y\ \forall z\ \ (\neg R(x, y) \vee S(x, z)) \wedge (\neg R(x, y) \vee P(x, z))$

- We represent each as a definite rule:

$\forall x\ \forall y\ \forall z\ \ R(x, y) \rightarrow S(x, z)$

and

$\forall x\ \forall y\ \forall z\ \ R(x, y) \rightarrow P(x, z)$

First Order (Predicate)Definite Clause Programming

First order definite clause programs

- A first order (predicate) definite clause program is a set of **first order definite clauses**
- These can be represented as their **equivalent definite rules** and we can program with these rules

First order definite clause programs

- A first order (predicate) definite clause program is a set of **first order definite clauses**
- These can be represented as their **equivalent definite rules** and we can **program** with these rules

definite clauses

1. $loves(mary, john)$
2. $engaged(mary, john)$
3. $\forall x\ \forall y\ \neg loves(x, y) \vee \neg engaged(x, y) \vee marries(x, y)$

definite rules

1. $\rightarrow loves(mary, john)$
2. $\rightarrow engaged(mary, john)$
3. $\forall x\ \forall y\ loves(x, y), engaged(x, y) \rightarrow marries(x, y)$

Querying a predicate logic program

- Let P be a **program** of first order definite rules
- A **query** to P is a PNF formula

$$\exists x_1 \ldots \exists x_n\, F$$

where F is a *conjunction* of positive *atoms* and $x_1 \ldots x_n$ are the variables in F

$\mathcal{P}_1$

1. $\rightarrow loves(mary, john)$
2. $\rightarrow engaged(mary, john)$
3. $\forall x\ \forall y\ loves(x, y), engaged(x, y) \rightarrow marries(x, y)$

Query $\quad ?\ \exists z \exists w\ marries(z, w)$

Querying a predicate logic program

- Just like we did last week with propositional logic programs…
- Expand selected query atom (instead of literal) by choosing **rule** with **matching head**
- And then **replace** with **body** of rule query

$\mathcal{P}_1$

1. $\rightarrow \boldsymbol{loves(mary,john)}$
2. $\rightarrow \boldsymbol{engaged(mary,john)}$
3. $\forall x\ \forall y\ loves(x, y), engaged(x, y) \rightarrow \boldsymbol{marries(x, y)}$

Query $\quad ?\ \exists z \exists w\ marries(z, w)$

So let's try…

$\mathcal{P}_1$

1. $\rightarrow$ ***loves(mary,john)***
2. $\rightarrow$ ***engaged(mary,john)***
3. $\forall x\ \forall y\ loves(x, y), engaged(x, y) \rightarrow \boldsymbol{marries(x, y)}$

? $\exists z \exists w\ marries(z, w)$

- I have a matching head, but it does not look right…
- The head is *marries* ***(x, y)***
- But my query is *marries* ***(z, w)***
- If only I could substitute that ***x*** with a ***z*** and that ***y*** with ***w*** …

Substitution

- A substitution S is a finite set $\{ (x_1/t_1), \ldots, (x_n/t_n) \}$ where:
  - $x_1, \ldots, x_n$ are **distinct** variables (**only variables can be substituted**)
  - $t_1, \ldots, t_n$ are terms (**variables, constants** or functions applied to terms)

- **every instance** of variable x_i is **simultaneously** replaced by t_i

So let's try again…

1. $\rightarrow$ ***loves(mary,john)***
2. $\rightarrow$ ***engaged(mary,john)***
3. $\forall x\ \forall y$ *loves(x, y), engaged(x, y)* $\rightarrow$ ***marries(x, y)***

$\mathcal{P}_1$

? *marries(z, w)*

marries(x, y)
{ (x/z), (y/w) }

? *loves(z, w), engaged(z, w)*

loves(mary,john)
{ (z/mary), (w/john) }

? *engaged(mary, john)*

engaged(mary, john)

$\square$

Tutorials and Next Lecture

- **Large Group Tutorial:**
  - Question 1 is in slides; make sure you can do them yourself!
  - *Tutorial questions 2 and 3* not in slides
- **Small Group Tutorials:**
  - You can complete questions 3 and 4 now.

- **Next Lecture:**
  - Predicate definite clause programming, Part 2
    - Derivation trees recap
    - Control, negation, recursion