Chapter 08
Predicate Logic

Instructor

LE Thanh Sach, Ph.D.

Instructor's Information

LE Thanh Sach, Ph.D.

Office:

Department of Computer Science,

Faculty of Computer Science and Engineering,

HoChiMinh City University of Technology.

Office Address:

268 LyThuongKiet Str., Dist. 10, HoChiMinh City, Vietnam.

E-mail: LTSACH@cse.hcmut.edu.vn

E-home: http://cse.hcmut.edu.vn/~ltsach/

Tel: (+84) 83-864-7256 (Ext: 5839)

Acknowledgment

The slides in this PPT file are composed using the materials supplied by

- **Prof. Stuart Russell and Peter Norvig**: They are currently from University of California, Berkeley. They are also the author of the book "Artificial Intelligence: A Modern Approach", which is used as the textbook for the course

- **Prof. Tom Lenaerts**, from Université Libre de Bruxelles

Outline

- Why FOL?
- Syntax and semantics of FOL
- Using FOL
- Wumpus world in FOL
- Knowledge engineering in FOL

Why FOL?
Pros and cons of propositional logic

☺ Propositional logic is declarative

☺ Propositional logic allows partial/disjunctive/negated information

- (unlike most data structures and databases)

☺ Propositional logic is compositional:

- meaning of $B_{1,1} \wedge P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$

☺ Meaning in propositional logic is context-independent

- (unlike natural language, where meaning depends on context)

☹ Propositional logic has very limited expressive power

- (unlike natural language)
- E.g., cannot say "pits cause breezes in adjacent squares"
  - ✓ except by writing one sentence for each square

Why FOL?
First-order logic

- Whereas propositional logic assumes the world contains facts,
- first-order logic (like natural language) assumes the world contains
  - Objects: people, houses, numbers, colors, baseball games, wars, …
  - Relations: red, round, prime, brother of, bigger than, part of, comes between, …
  - Functions: father of, best friend, one more than, plus, …

Prepositional Logic: Examples

Sentence in natural language	Sentence in Prepositional Logic
Socrates is a man	SOCRATESMAN
Plato is a man	PLATOMAN
All men are mortal	MORTALMAN

Disadvantages:

1. **Socrates and Plato are human, but the representation is not similar**
2. **Can not state that each person is mortal**

→ Difficult for reasoning.

Predicate Logic: Examples

➔ Using predicate logic

Sentence in natural language	Sentence in Predicate Logic
Socrates is a man	man(socrates)
Plato is a man	man(plato)
All men are mortal	$\forall X$ (man(X) → mortal(X))

Predicate Logic: Examples

- Ability of reasoning

Sentence in natural language	Conclusion
Socrates is a man	Socrates is mortal
All men are mortal	

Sentence in Predicate Logic	Conclusion
man(socrates)	mortal(socrates)
$\forall X\ (man(X) \rightarrow mortal(X))$	

Predicate Logic: Examples

#	Representation	Rule
1.	man(socrates)	Axiom
2.	$\forall$X (man(X) $\rightarrow$ mortal(X))	Axiom
3	man(socrates)$\rightarrow$mortal(socrates)	2, X= “socrates”, UI
4.	mortal(socrates)	1,3, MP

Backward mapping (to natural language)

Socrates is mortal.

Syntax of FOL: Basic elements

- Constants:
  - KingJohn, 2, NUS,...
- Predicates :
  - Brother, >,...
- Functions:
  - Sqrt, LeftLegOf,...
- Variables:
  - x, y, a, b,...
- Connectives:
  - $\neg$, $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$
- Equality:
  - =
- Quantifiers:
  - Universal Quantifier: $\forall$
  - Existential Quantifier: $\exists$

Atomic sentences

Term : *function* (*term*$_1$,...,*term*$_n$)
or *constant* or *variable*

Atomic sentence : *predicate* (*term*$_1$,...,*term*$_n$)
or *term*$_1$ = *term*$_2$

- E.g.,
  - *Brother(KingJohn,RichardTheLionheart)*
  - *> (Length(LeftLegOf(Richard)), Length(LeftLegOf(KingJohn)))*

Complex sentences

❖ Complex sentences are made from atomic sentences using connectives, examples:

- $\neg S$
- $S_1 \wedge S_2$
- $S_1 \vee S_2$
- $S_1 \Rightarrow S_2$
- $S_1 \Leftrightarrow S_2$

E.g. *Sibling(KingJohn,Richard)* $\Rightarrow$ *Sibling(Richard,KingJohn)*

$>(1,2) \vee \leq (1,2)$

$>(1,2) \wedge \neg >(1,2)$

Truth in first-order logic

- Sentences are true with respect to a model and an interpretation
- Model contains objects (domain elements) and relations among them
- Interpretation specifies referents for

constant symbols	→	objects
predicate symbols	→	relations
function symbols	→	functional relations

- An atomic sentence $predicate(term_1,...,term_n)$ is true
  iff the objects referred to by $term_1,...,term_n$
  are in the relation referred to by *predicate*

Universal quantification

- $\forall$<*variables*> <*sentence*>

Everyone at NUS is smart:
$\forall x\ At(x,NUS) \Rightarrow Smart(x)$

- $\forall x\ P$ is true in a model *m* iff *P* is true with *x* being each possible object in the model

- Roughly speaking, equivalent to the conjunction of instantiations of *P*

$$
\begin{aligned}
& At(KingJohn,NUS) \Rightarrow Smart(KingJohn) \\
\wedge\ & At(Richard,NUS) \Rightarrow Smart(Richard) \\
\wedge\ & At(NUS,NUS) \Rightarrow Smart(NUS) \\
\wedge\ & \ldots
\end{aligned}
$$

A common mistake to avoid

- Typically, $\Rightarrow$ is the main connective with $\forall$
- **Common mistake**: using $\wedge$ as the main connective with $\forall$:

  $\forall x\ At(x,NUS) \wedge Smart(x)$

  means "Everyone is at NUS and everyone is smart"

Existential quantification

- ∃<*variables*> <*sentence*>

- Someone at NUS is smart:
- $\exists x$ At(x,NUS) ∧ Smart(x)$

- $\exists x\ P$ is true in a model *m* iff *P* is true with *x* being some possible object in the model

- Roughly speaking, equivalent to the disjunction of instantiations of *P*

  At(KingJohn,NUS) ∧ Smart(KingJohn)
  ∨ At(Richard,NUS) ∧ Smart(Richard)
  ∨ At(NUS,NUS) ∧ Smart(NUS)
  ∨ ...

Another common mistake to avoid

- Typically, $\wedge$ is the main connective with $\exists$
- **Common mistake**: using $\Rightarrow$ as the main connective with $\exists$:

$$\exists x \ \mathrm{At}(x,\mathrm{NUS}) \Rightarrow \mathrm{Smart}(x)$$

is true if there is anyone who is not at NUS!

Properties of quantifiers

- $\forall x\ \forall y$ is the same as $\forall y\ \forall x$
- $\exists x\ \exists y$ is the same as $\exists y\ \exists x$

- $\exists x\ \forall y$ is not the same as $\forall y\ \exists x$
- $\exists x\ \forall y$ Loves(x,y)
  - "There is a person who loves everyone in the world"
- $\forall y\ \exists x$ Loves(x,y)
  - "Everyone in the world is loved by at least one person"

- Quantifier duality: each can be expressed using the other
- $\forall x$ Likes(x,IceCream) $\qquad \neg\exists x\ \neg$Likes(x,IceCream)
- $\exists x$ Likes(x,Broccoli) $\qquad \neg\forall x\ \neg$Likes(x,Broccoli)

Equality

- $term_1 = term_2$ is true under a given interpretation if and only if $term_1$ and $term_2$ refer to the same object

- E.g., definition of *Sibling* in terms of *Parent*:

  $\forall x,y\ Sibling(x,y) \Leftrightarrow [\neg(x = y) \wedge \exists m,f\ \neg(m = f) \wedge Parent(m,x) \wedge Parent(f,x) \wedge Parent(m,y) \wedge Parent(f,y)]$

Using FOL

The kinship domain:

- Brothers are siblings

  $\forall x,y\ Brother(x,y) \Leftrightarrow Sibling(x,y)$

- One's mother is one's female parent

  $\forall m,c\ Mother(c) = m \Leftrightarrow (Female(m) \wedge Parent(m,c))$

- "Sibling" is symmetric

  $\forall x,y\ Sibling(x,y) \Leftrightarrow Sibling(y,x)$

Predicate Logic: Representation

- Representing the following sentences in Predicate Logic:

1. Marcus was a man.
2. Macus was a Pompeian.
3. All Pompians were Romans.
4. Caesar was a ruler.
5. All Romans were either loyal to Caesar or hated hime.
6. Everyone is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar.

Predicate Logic: Representation

1. Marcus was a man.
   man(Marcus)

2. Macus was a Pompeian.
   Pompeian(Marcus)

3. All Pompeians were Romans.
   $\forall X$: Pompeian(X) → Roman(X)

4. Caesar was a ruler.
   ruler(Caesar)

Predicate Logic: Representation

5. All Romans were either loyal to Caesar or hated hime.

"or" mean "inclusive or" - OR:

$\forall X$: Roman(X) → loyalto(X, Caesar) v hate(X, Caesar)

"or" mean "exclusive or" - XOR:

$\forall X$: Roman(X) → [(loyalto(X, Caesar) v hate(X, Caesar)) ^
¬(loyalto(X, Caesar) ^ hate(X, Caesar))]

or:

$\forall X$: Roman(X) → [(loyalto(X, Caesar) ^ ¬hate(X, Caesar)) v
(¬loyalto(X, Caesar) ^ hate(X, Caesar))]

Predicate Logic: Representation

6. Everyone is loyal to someone.

   $\forall X: \exists Y: loyalto(X,Y).$

7. People only try to assassinate rulers they are not loyal to.

   $\forall X: \forall Y: person(X) \wedge ruler(Y) \wedge tryassassinate(X,Y) \rightarrow \neg loyalto(X,Y)$

Predicate Logic: Representation

8. Marcus tried to assassinate Caesar.

   tryassassinate(Marcus, Caesar).

Predicate Logic: Representation

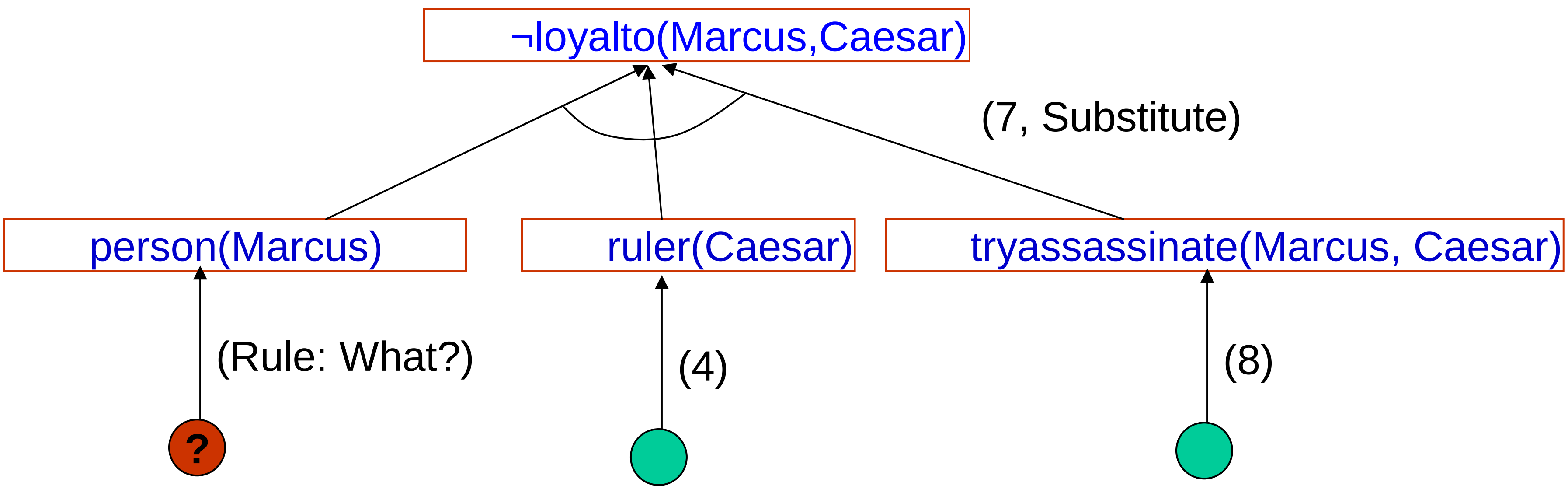
- Given the above sentences, can we make a conclusion as follows:

  "Marcus was not loyal to Caesar ?"

  or:

  $\neg loyalto(Marcus, Caesar)$

Predicate Logic: Proof

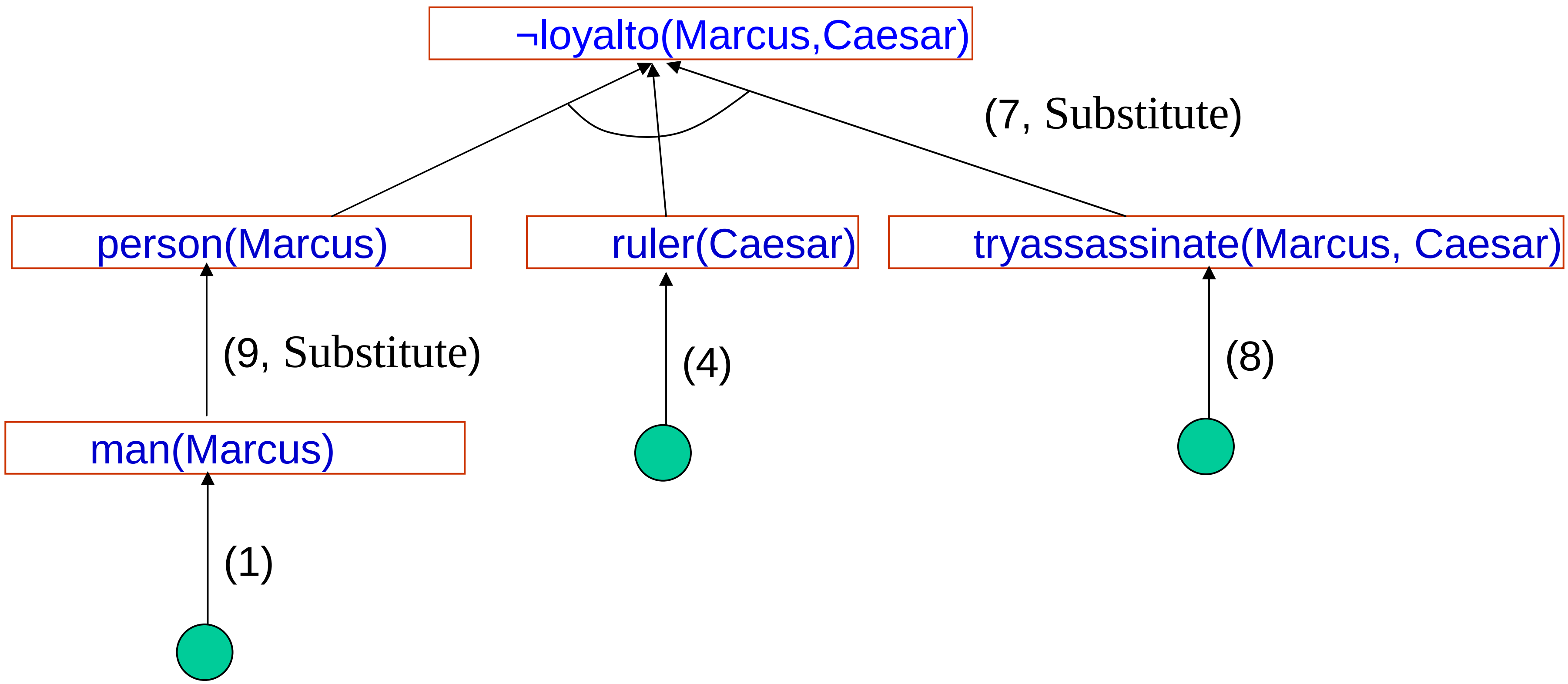


Add an implicit sentence:
9. "All men are people"

or:

$\forall$X: man(X) → X: man(X) → person(X)

Predicate Logic: Proof



Predicate Logic: Exception

- How to represent exceptions, example ?

  "Paulus was a Pompeian. Paulus was neither loyal nor hated Caesar"

Predicate Logic: Exception

- Exception representation:
  - Semantic Net: Simple!, by adding links (properties) to subclasses or instances.
  - Predicate Logic: Not so simple!, like

    Pompeian(Paulus) ^ ¬[loyalto(Paulus, Caesar) v hate(Pualus, Caesar)]

Predicate Logic: Exception

❖ Exception representation:

Not so simple Because there is conflict between:

$Pompeian(Paulus) \wedge \neg[loyalto(Paulus, Caesar) \vee hate(Pualus, Caesar)]$

and

$\forall X: Pompeian(X) \rightarrow loyalto(X, Caesar) \vee hate(X, Caesar)$

Predicate Logic: Exception

- Exception representation:
  - A Better way:

$\forall X$: Roman(X) ^ **¬eq(X, Paulus)** → loyalto(X, Caesar) v hate(X, Caesar

Exception

Predicate Logic:
Function + Computable Predicate

- In the case of describing relations between numbers, example:
  - ✓ $1 < 2$
  - ✓ $2 <3$
  - ✓ $7 > 3 +2$,
  - ✓ $3 > 1$
  - ✓ ….

$\Rightarrow$ Should not explicitly describe:

lt(1,2), lt(2,3), …

Predicate Logic: Function + Computable Predicate

- ⇒ Need functions and computable predicates.
- ⇒ For example,
  - Call a function to compute (3+2)
  - Pass the result to the predicate “gt”: gt(7, 3+2)
  - Get the final result (True)

Predicate Logic: Function + Computable Predicate, Example

- Facts:
  1. Marcus was a man.
     man(Marcus)
  2. Marcus was a Pompeian.
     Pompeian(Marcus)
  3. Marcus was born in 40 A.D
     born(Marcus,40)
  4. All men are mortal.
     $\forall X: man(X) \rightarrow mortal(X)$

Predicate Logic:
Function + Computable Predicate, Example

- Facts:

5. All Pompeian died when the vocano erupted in 79 AD.

   erupted(vocano, 79) ^ $\forall$X: [Pompeian(X) → died(X, 79)]

6. No mortal lives longer then 150 years.

   $\forall X: \forall T_1: \forall T_2$: mortal(X) ^ born(X, T_1) ^ gt($T_2 - T_1$, 150) → dead(X, T_2)

7. It is now 1991

   now = 1991

Predicate Logic:
Function + Computable Predicate, Example

- Question:

  Is Marcus alive?

  Or

  alive(Marcus, now)

  or

  ¬alive(Marcus, now)

Predicate Logic: Function + Computable Predicate, Example

❖ Add implicit knowledge:

✍ Relation between "dead" and "alive"

8. Alive means not dead.

$\forall X: \forall T: \quad [alive(X,T) \rightarrow \neg dead(X,T)] \wedge$
$\quad\quad [\neg dead(X,T) \rightarrow alive(X,T)]$

Predicate Logic:
Function + Computable Predicate, Example

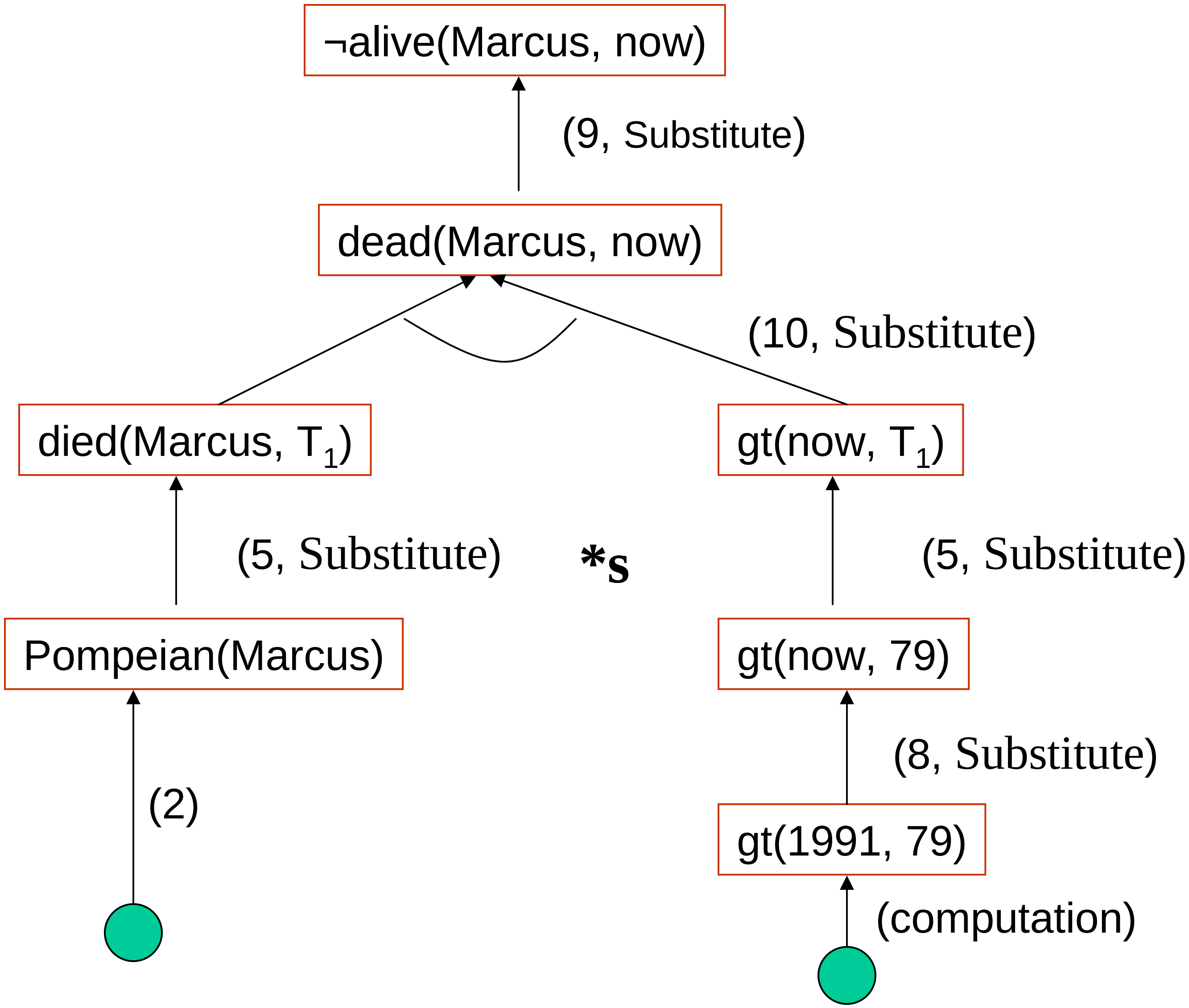
- Add implicit knowledge:
  - Time properties of an event, for example “dead”

9. Is someone dies, the he is dead at all later times

$\forall X: \forall T_1: \forall T_2: died(X,T_1) \wedge gt(T_2, T_1) \rightarrow dead(X, T_2)$

Predicate Logic: Function + Computable Predicate, Example

- Summary of facts:
  1. man(Marcus)
  2. Pompeian(Marcus)
  3. born(Marcus, 40)
  4. $\forall X$: man(X) → mortal(X)
  5. $\forall X$: Pompeian(X) → died(X, 79)
  6. erupted(volcano, 79)
  7. $\forall X: \forall T_1: \forall T_2$: mortal(X) ^ born(X, T_1) ^ gt($T_2 - T_1$, 150) → dead(X, T_2)
  8. now = 1991
  9. $\forall X: \forall T$: [alive(X,T) → ¬dead(X,T)] ^ [¬dead(X,T) → alive(X,T)]
  10. $\forall X: \forall T_1: \forall T_2$: died(X,$T_1$) ^ gt($T_2$, T_1) → dead(X, T_2)


¬alive(Marcus, now)
(9, Substitute)
dead(Marcus, now)
(10, Substitute)
died(Marcus, T1)
gt(now, T1)
(5, Substitute)
*s
(5, Substitute)
Pompeian(Marcus)
gt(now, 79)
(2)
(8, Substitute)
gt(1991, 79)
(computation)

Universal instantiation (UI)

- Every instantiation of a universally quantified sentence is entailed by it:

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

  for any variable v and ground term g

- E.g., $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ 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)

- For any sentence α, variable v, and constant symbol k that does not appear elsewhere in the knowledge base:

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

- E.g., $\exists x\ Crown(x) \wedge OnHead(x, John)$ yields:

$$Crown(C_1) \wedge OnHead(C_1, John)$$

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

Reduction to propositional inference

Suppose the KB contains just the following:

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

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

- The new KB is propositionalized: proposition symbols are

  King(John), Greedy(John), Evil(John), King(Richard), etc.

Reduction contd.

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

Resolution

- Convert to clause form, example:

  "All Romans who know Marcus either hate Caesar or think that anyone who hates anyone is crazy"

$$\forall X: \quad [roman(X) \wedge know(X, Marcus)] \rightarrow [hate(X, Ceasar) \vee (\forall Y: \exists Z: hate(Y,Z) \rightarrow thinkcrazy(X,Y))]$$

Resolution

1. Remove $\rightarrow$
   using the equivalence: $a \rightarrow b = \neg a \vee b$

Resolution

1. Remove $\rightarrow$
   using the equivalence: $a \rightarrow b = \neg a \vee b$

$\forall X$: [roman(X) ^ know(X, Marcus)] $\rightarrow$
[hate(X, Ceasar) v
($\forall Y$: $\exists Z$: hate(Y,Z) $\rightarrow$ thinkcrazy(X,Y))]

=

$\forall X$: ¬[roman(X) ^ know(X, Marcus)] v
[hate(X, Ceasar) v
($\forall Y$: ¬($\exists Z$: hate(Y,Z)) v thinkcrazy(X,Y))]

Resolution

2. Move ¬ next to terms:

   Using the following equivalence:

   a. Double negative:

   $\neg(\neg p) = p$

   b. DeMorgan:

   $\neg(a \vee b) = \neg a \wedge \neg b$

   $\neg(a \wedge b) = \neg a \vee \neg b$

   c. Equivalence of qualifiers :

   $\neg\forall X: P(X) = \exists X: \neg P(X)$

   $\neg\exists X: P(X) = \forall X: \neg P(X)$

Resolution

2. Move ¬ next to terms:

$\forall X$: ¬[roman(X) ^ know(X, Marcus)] v
[hate(X, Ceasar) v
($\forall Y$: ¬($\exists Z$: hate(Y,Z)) v thinkcrazy(X,Y))]

=

$\forall X$: [¬roman(X) v ¬know(X, Marcus)]v
[hate(X, Ceasar) v
($\forall Y$: $\forall Z$: ¬hate(Y,Z) v
thinkcrazy(X,Y))]

Resolution

3. Change name of variables as follows:

$$\forall X: P(X) \vee \forall X: Q(X)$$

$$=$$

$$\forall X: P(X) \vee \forall Y: Q(Y)$$

Resolution

4. Move qualifier to left, don't change the their order

$\forall$X: [¬roman(X) v ¬know(X, Marcus)]v
[hate(X, Ceasar) v
($\forall$Y: $\forall$Z: ¬hate(Y,Z) v
thinkcrazy(X,Y))]

=

$\forall$X:$\forall$Y:$\forall$Z: [¬roman(X) v ¬know(X, Marcus)] v
[hate(X, Ceasar) v
(¬hate(Y,Z) v thinkcrazy(X,Y))]

Resolution

5. Replace existence qualifier using Skolem function, as follows:

   Skolem Function:

   $\forall X: \forall Y: \exists Z : P(X,Y,Z)$

   $=$

   $\forall X: \forall Y: P(X,Y,f(X,Y))$

   - Variable of existence qualifier = function of all the preceding variables in the every qualifier

Resolution

6. Remove every qualifiers (the default qualifier is every)

$\forall X{:}\forall Y{:}\forall Z{:}$ [¬roman(X) v ¬know(X, Marcus)] v
[hate(X, Ceasar) v
(¬hate(Y,Z) v thinkcrazy(X,Y))]

=

[¬roman(X) v ¬know(X, Marcus)] v
[hate(X, Ceasar) v
(¬hate(Y,Z) v thinkcrazy(X,Y))]

Resolution

7. Convert to CNF.

   Using distribution rule of v and ^

   Examples:

   (a ^ b) v c = (a v c) ^ (b v c)

   (a ^ b) v (c ^ d) = (a v c) ^ (a v d) ^ (b v c) ^ (b v d)

Resolution

7. Convert to CNF.

   [¬roman(X) v ¬know(X, Marcus)] v

   [hate(X, Ceasar) v

   (¬hate(Y,Z) v thinkcrazy(X,Y))]

=

   ¬roman(X) v ¬know(X, Marcus) v

   hate(X, Ceasar) v ¬hate(Y,Z) v thinkcrazy(X,Y)

Resolution

8. Separate clauses.

- If having the clause form:

  (a v ¬b) ^ (¬a v c v d) ^ (a v ¬c v e)

  Then:

1. (a v ¬b)
2. (¬a v c v d)
3. (a v ¬c v e)

Resolution

9. Add every qualifier to clause:

That is:

$$(\forall X: P(X) \wedge Q(X))$$

$$=$$

$$\forall X: P(X) \wedge \forall X: Q(X)$$

Resolution

Exercises:

- ❖ Convert to clause form for sentences in previous example.
- ❖ Convert to clause form for:

1. $\forall X\ A(X) \vee \exists X\ B(X) \rightarrow \forall X C(X) \wedge \exists X\ D(X)$
2. $\forall X\ (p(X) \vee q(X)) \rightarrow \forall X\ p(X) \vee \forall X\ q(X)$
3. $\exists X\ p(X) \wedge \exists X\ q(X) \rightarrow \exists X (p(X) \wedge q(X))$
4. $\forall X\ \exists Y\ p(X,Y) \rightarrow \exists Y\ \forall X\ p(X,Y)$
5. $\forall X\ (p(X, f(X)) \rightarrow p(X,Y))$

Substitution and Unification

- X: Variable, t: term.
  - x/t: value of x will be t OR x is replaced by t.
  - A substitution = $\{x_1/t_1, x_2/t_2, .., x_n/t_n\}$
  - Empty substitution: ε
- E: Predicate expression
- θ: A substitution.

Substitution and Unification

- θ: A substitution.
  - $E\theta$: Apply θ to E.
  - Example:
    $E = p(X,Y,f(X))$,
    $\theta = \{X/a, y/f(b)\}$
    Then: $E\theta = p(a, f(b), f(a))$
- Union of θ and δ
  - E is a expression then:
    $E(\theta\delta) = (E\theta)\delta$

Substitution and Unification

- Union of θ and δ:
  - $\theta = \{x_1/t_1, x_2/t_2, .., x_n/t_n\}$
  - $\delta = \{y_1/s_1, y_2/s_2, .., y_n/s_n\}$

  - $\theta\delta$ is calculated as follows:

Substitution and Unification

- Union of θ and δ:
  - $\theta\delta$ is calculated as follows:

1. Apply δ to denominator of θ:
   $\{x_1/t_1\delta, x_2/t_2\delta, .., x_n/t_n\delta\}$
2. Remove all the form: x_i/x_i from 1.
3. Remove from δ the form: y_i/s_i
   if $y_i \in \{x_1, x_2, .., x_n\}$
4. $\theta\delta$ = union(line 2 and 3)

Substitution and Unification

- Union of θ and δ:
  - θ , δ : substitution.
  - ε : empty substitution.
  - E: Expression.

  - Properties:

1. $E(\theta\delta) = (E\theta)\delta$.
2. $E\varepsilon = E$
3. $\theta\varepsilon = \varepsilon\theta = \theta$

Substitution and Unification

- Unification:
  - If θ The most general unifier of S containing expression, then: Sθ is a set having single element.

Substitution and Unification

- mgu: most general unifier
  - S: set of expressions.
  - θ: mgu of S if:
    - ✓ Given any unifier δ of S then.
    - ✓ $\exists\ \alpha$ such that:

      $\delta = \theta\alpha$

Substitution and Unification

- Dis-set:
  - S: Set of expressions.
  - Find the common longest substring from the begining of each expression in S.
  - Dis-set = set of right next term in all the expressions.

Substitution and Unification

- Dis-set:
  - Example:

    S = {p(X, f(X), y), p(X,Y,Z), p(X, f(a), b)}

    → Common longest substring = "p(X,"

    → Dis-set = {f(X), Y, f(a)}

Substitution and Unification

- Unification algorithm:
  - Input: S = {atoms}
  - Output: mgu(S) or not unifiable(S).

1. Set K =0 and $\theta_0 = \varepsilon$. Goto 2
2. Compute $S\theta_k$.
   If it is a set having single element
   → stop: mgu = θ_k.
   Else D_k = dis-set($S\theta_k$). Goto 3.
3. If D_k contains variable V and a term t,
   and V is not in term t,
   Then compute $\theta_{k+1} = \theta_k \{V/t\}$, set K = K + 1. Goto 2.

Substitution and Unification

- Clause's properties:
  1. There is no common variable between two clauses.
  2. There exist one or many atoms: $L_1, L_2, .., L_k$ in a clause, one or many literals $\neg M_1, \neg M_2, .., \neg M_n$ in another clause so that there is a mgu for set $\{L_1, L_2, .., L_k, M_1, M_2, .., M_n\}$

Substitution and Unification

❖ Resolution rule:.

$$\frac{\begin{array}{c} L_1 \vee \ldots \vee L_k \vee C \\ \neg M_1 \vee \ldots \vee \neg M_n \vee D \end{array}}{\therefore (C\theta - N) \vee (D\theta - \neg N)}$$

$$\boxed{\begin{aligned} N &= L_i\theta \\ &= M_j\theta \end{aligned}}$$

Substitution and Unification

- Resolution rule:
  1. Verify the difference of variable name between two clauses.
  2. Find mgu, θ, for: $\{L_1, L_2, .., L_k , M_1, M_2, .., M_n \}$
  3. Apply θ into C và D.
  4. Compute: $N = L_1\theta$
  5. Remove N from $C\theta$
  6. Remove $\neg N$ from $D\theta$
  7. Compute union of 5 and 6, and result resovant.

Substitution and Unification

- Resolution rule:

  - Properties of resolvants:

    E, F are two clauses.

    G is their resolvant.

    {E,F} is unsatisfiable if and only if
    {E,F,G} is unsatisfiable.

Substitution and Unification

- Proof with resolution rule:
  - Set of expression F
  - Prove: P
  - Procedure:

1. Convert F into clauses.
2. Take ¬P, convert ¬P into clauses. Add to result of Step 1.
3. Apply resolution rule to produce empty set, i.e., find a contradiction.

Substitution and Unification

❖ Example:

STT	Clauses
1	man(Marcus)
2	Pompiean(Marcus)
3	¬Pompiean(X1) v Roman(X1)
4	Ruler(Caesar)
5	¬Roman(X2) v loyalto(X2, Caesar) v hate(X2, Caesar)
6	Loyalto(X3, f1(X3))
7	¬man(X4) v ¬ruler(Y1) v ¬tryassassinate(X4, Y1) v loyalto(X4, Y1)
8	Tryassassinate(Marcus, Caesar)

Prove: "Marcus hate Caesar." or hate(Marcus, Ceasar).

Proof

#	Clauses	Note
1	man(Marcus)	P
2	Pompiean(Marcus)	P
3	¬Pompiean(X1) v Roman(X1)	P
4	Ruler(Caesar)	P
5	¬Roman(X2) v loyalto(X2, Caesar) v hate(X2, Caesar)	P
6	Loyalto(X3, f1(X3))	P
7	¬man(X4) v ¬ruler(Y1) v ¬tryassassinate(X4, Y1) v ¬loyalto(X4, Y1)	P
8	Tryassassinate(Marcus, Caesar)	P
9	¬hate(Marcua, Ceasar).	P
10	¬Roman(Marcus) v loyalto(Marcus, Caesar)	5,9: X2= Marcus
11	¬Pompiean(Marcus) v loyalto(Marcus, Caesar)	3,10: X1 = Marcus

Proof

STT	Clauses	Ghi chú
12	loyalto(Marcus, Ceasar)	2,11:
13	¬man(Marcus) v ¬ruler(Ceasar) v ¬tryassassinate(Marcus, Ceasar)	12,7:X4=Marcus, y1=Ceasar
14	¬ruler(Ceasar) v ¬tryassassinate(Marcus, Ceasar)	13,1
15	¬tryassassinate(Marcus, Ceasar)	14,4
16	□	15,8

Homeworks

Prove: ¬alive(Marcus, now) from the previous example.

Exercises

- You know:
  - All people living in PMH are rich.
  - Rich people have much money
  - People who work only in government sector (or called government staff) do not have much money.
  - An is a government staff
- Prove:
  - An does not live in PMH.