Fundamentals of Artificial Intelligence
Chapter 09: **Inference in First-Order Logic**

Roberto Sebastiani

DISI, Università di Trento, Italy – roberto.sebastiani@unitn.it
http://disi.unitn.it/rseba/DIDATTICA/fai_2020/

Teaching assistant: Mauro Dragoni – dragoni@fbk.eu
http://www.maurodragoni.com/teaching/fai/

M.S. Course "Artificial Intelligence Systems", academic year 2020-2021

Last update: Tuesday 8th December, 2020, 13:08

Outline

1. Propositional vs. First-Order Inference
2. Unification and Lifting
3. Forward & Backward Chaining for Definite FOL KBs
4. Resolution for General FOL KBs

Outline

1. Propositional vs. First-Order Inference
2. Unification and Lifting
3. Forward & Backward Chaining for Definite FOL KBs
4. Resolution for General FOL KBs

A Brief History of Logical Reasoning

When	Who	What
322 B.C.	Aristotle	“Syllogisms” (inference rules), quantifiers
1867	Boole	PL
1879	Frege	FOL
1922	Wittgenstein	proof by truth tables
1930	Gödel	$\exists$ complete algorithm for FOL
1930	Herbrand	complete algorithm for FOL
1931	Gödel	$\neg\exists$ complete algorithm for arithmetic
1960	Davis/Putnam	“practical” algorithm for PL (DP/DPLL)
1965	Robinson	“practical” algorithm for FOL (resolution)

Term/Subformula Substitutions

Notation

- Substitution: "Subst($\{e_1/e_2\}, e$)" or "$e\{e_1/e_2\}$":
  the expression (term or formula) obtained by substituting every occurrence of e_1 with e_2 in e
  - e_1, e_2 either both terms (term substitution)
    or both subformulas (subformula substitution)
  - e is either a term or a formula (only term for term substitution)
- Examples:
  - (t. sub.): $(y + 1 = 1 + y)\{y/S(x)\} \Longrightarrow (S(x) + 1 = 1 + S(x))$
  - (s. sub.): $(Even(x) \vee Odd(x))\{Even(x)/Odd(S(x))\} \Longrightarrow$
    $((Odd(S(x)) \vee Odd(x))$
- Multiple substitution: $e\{e_1/e_2, e_3/e_4\} \stackrel{\text{def}}{=} (e\{e_1/e_1\})\{e_3/e_4\}$
  - ex: $(P(x, y) \rightarrow Q(x, y))\{x/1, y/2\} \Longrightarrow (P(1, 2) \rightarrow Q(1, 2))$
- If θ is a substitution list and e an expression (formula/term), then we denote the result of a substitution as $e\theta$
  - $e\emptyset = e$
  - $e(\theta_1\theta_2) = (e\theta_1)\theta_2$, denoted as $e\theta_1\theta_2$

Term/Subformula Substitutions

Notation

- Substitution: "Subst($\{e_1/e_2\}, e$)" or "$e\{e_1/e_2\}$":
  the expression (term or formula) obtained by substituting every occurrence of e_1 with e_2 in e
  - e_1, e_2 either both terms (term substitution)
    or both subformulas (subformula substitution)
  - e is either a term or a formula (only term for term substitution)
- Examples:
  - (t. sub.): $(y + 1 = 1 + y)\{y/S(x)\} \Longrightarrow (S(x) + 1 = 1 + S(x))$
  - (s. sub.): $(Even(x) \vee Odd(x))\{Even(x)/Odd(S(x))\} \Longrightarrow ((Odd(S(x)) \vee Odd(x))$
- Multiple substitution: $e\{e_1/e_2, e_3/e_4\} \stackrel{\text{def}}{=} (e\{e_1/e_1\})\{e_3/e_4\}$
  - ex: $(P(x, y) \rightarrow Q(x, y))\{x/1, y/2\} \Longrightarrow (P(1, 2) \rightarrow Q(1, 2))$
- If θ is a substitution list and e an expression (formula/term), then we denote the result of a substitution as $e\theta$
  - $e\emptyset = e$
  - $e(\theta_1\theta_2) = (e\theta_1)\theta_2$, denoted as $e\theta_1\theta_2$

Term/Subformula Substitutions

Notation

- Substitution: "$\text{Subst}(\{e_1/e_2\}, e)$" or "$e\{e_1/e_2\}$":
  the expression (term or formula) obtained by substituting every occurrence of e_1 with e_2 in e
  - e_1, e_2 either both terms (term substitution)
    or both subformulas (subformula substitution)
  - e is either a term or a formula (only term for term substitution)
- Examples:
  - (t. sub.): $(y + 1 = 1 + y)\{y/S(x)\} \Longrightarrow (S(x) + 1 = 1 + S(x))$
  - (s. sub.): $(Even(x) \vee Odd(x))\{Even(x)/Odd(S(x))\} \Longrightarrow$
    $((Odd(S(x)) \vee Odd(x))$
- Multiple substitution: $e\{e_1/e_2, e_3/e_4\} \stackrel{\text{def}}{=} (e\{e_1/e_1\})\{e_3/e_4\}$
  - ex: $(P(x, y) \rightarrow Q(x, y))\{x/1, y/2\} \Longrightarrow (P(1, 2) \rightarrow Q(1, 2))$
- If θ is a substitution list and e an expression (formula/term), then we denote the result of a substitution as $e\theta$
  - $e\emptyset = e$
  - $e(\theta_1\theta_2) = (e\theta_1)\theta_2$, denoted as $e\theta_1\theta_2$

Term/Subformula Substitutions

Notation

- Substitution: "Subst($\{e_1/e_2\}, e$)" or "$e\{e_1/e_2\}$":
  the expression (term or formula) obtained by substituting every occurrence of e_1 with e_2 in e
  - e_1, e_2 either both terms (term substitution) or both subformulas (subformula substitution)
  - e is either a term or a formula (only term for term substitution)
- Examples:
  - (t. sub.): $(y + 1 = 1 + y)\{y/S(x)\} \Longrightarrow (S(x) + 1 = 1 + S(x))$
  - (s. sub.): $(Even(x) \vee Odd(x))\{Even(x)/Odd(S(x))\} \Longrightarrow$ $((Odd(S(x)) \vee Odd(x))$
- Multiple substitution: $e\{e_1/e_2, e_3/e_4\} \stackrel{\text{def}}{=} (e\{e_1/e_1\})\{e_3/e_4\}$
  - ex: $(P(x, y) \rightarrow Q(x, y))\{x/1, y/2\} \Longrightarrow (P(1, 2) \rightarrow Q(1, 2))$
- If θ is a substitution list and e an expression (formula/term), then we denote the result of a substitution as $e\theta$
  - $e\emptyset = e$
  - $e(\theta_1\theta_2) = (e\theta_1)\theta_2$, denoted as $e\theta_1\theta_2$

Term/Subformula Substitutions

Notation

- Substitution: "Subst($\{e_1/e_2\}, e$)" or "$e\{e_1/e_2\}$":
  the expression (term or formula) obtained by substituting every occurrence of e_1 with e_2 in e
  - e_1, e_2 either both terms (term substitution)
    or both subformulas (subformula substitution)
  - e is either a term or a formula (only term for term substitution)
- Examples:
  - (t. sub.): $(y + 1 = 1 + y)\{y/S(x)\} \Longrightarrow (S(x) + 1 = 1 + S(x))$
  - (s. sub.): $(Even(x) \vee Odd(x))\{Even(x)/Odd(S(x))\} \Longrightarrow$
    $((Odd(S(x)) \vee Odd(x))$
- Multiple substitution: $e\{e_1/e_2, e_3/e_4\} \stackrel{\text{def}}{=} (e\{e_1/e_1\})\{e_3/e_4\}$
  - ex: $(P(x, y) \rightarrow Q(x, y))\{x/1, y/2\} \Longrightarrow (P(1, 2) \rightarrow Q(1, 2))$
- If θ is a substitution list and e an expression (formula/term), then we denote the result of a substitution as $e\theta$
  - $e\emptyset = e$
  - $e(\theta_1\theta_2) = (e\theta_1)\theta_2$, denoted as $e\theta_1\theta_2$

Substitution with equivalent formulas/terms

- Equal-term substitution rule:

$$\frac{KB \wedge (t_1 = t_2) \wedge \alpha}{KB \wedge (t_1 = t_2) \wedge \alpha \wedge \alpha\{t_1/t_2\}}$$

  - ex: $(S(x) = x + 1) \wedge (0 \neq S(x))$ thus $(S(x) = x + 1) \wedge (0 \neq S(x)) \wedge (0 \neq x + 1)$
  - preserves validity: $M(KB \wedge (t_1 = t_2) \wedge \alpha \wedge \alpha\{t_1/t_2\}) = M(KB \wedge (t_1 = t_2) \wedge \alpha)$
  - α can be safely dropped from the result

- Equivalent-subformula substitution rule:

$$\frac{KB \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha}{KB \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha \wedge \alpha\{\beta_1/\beta_2\}}$$

  - ex: $(Even(x) \leftrightarrow Odd(S(x))) \wedge (Even(x) \vee Odd(x)) \Longrightarrow (Even(x) \leftrightarrow Odd(S(x))) \wedge (Even(x) \vee Odd(x)) \wedge (Odd(S(x)) \vee Odd(x))$
  - preserves validity: $M(KB \wedge (\beta_1 = \beta_2) \wedge \alpha \wedge \alpha\{\beta_1/\beta_2\}) = M(KB \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha)$
  - α can be safely dropped from the result

Substitution with equivalent formulas/terms

- Equal-term substitution rule:

$$\frac{KB \wedge (t_1 = t_2) \wedge \alpha}{KB \wedge (t_1 = t_2) \wedge \alpha \wedge \alpha\{t_1/t_2\}}$$

  - ex: $(S(x) = x + 1) \wedge (0 \neq S(x))$ thus $(S(x) = x + 1) \wedge (0 \neq S(x)) \wedge (0 \neq x + 1)$
  - preserves validity: $M(KB \wedge (t_1 = t_2) \wedge \alpha \wedge \alpha\{t_1/t_2\}) = M(KB \wedge (t_1 = t_2) \wedge \alpha)$
  - α can be safely dropped from the result

- Equivalent-subformula substitution rule:

$$\frac{KB \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha}{KB \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha \wedge \alpha\{\beta_1/\beta_2\}}$$

  - ex: $(Even(x) \leftrightarrow Odd(S(x))) \wedge (Even(x) \vee Odd(x)) \Longrightarrow (Even(x) \leftrightarrow Odd(S(x))) \wedge (Even(x) \vee Odd(x)) \wedge (Odd(S(x)) \vee Odd(x))$
  - preserves validity: $M(KB \wedge (\beta_1 = \beta_2) \wedge \alpha \wedge \alpha\{\beta_1/\beta_2\}) = M(KB \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha)$
  - α can be safely dropped from the result

Substitution with equivalent formulas/terms

- Equal-term substitution rule:

$$\frac{KB \wedge (t_1 = t_2) \wedge \alpha}{KB \wedge (t_1 = t_2) \wedge \alpha \wedge \alpha\{t_1/t_2\}}$$

  - ex: $(S(x) = x + 1) \wedge (0 \neq S(x))$ thus $(S(x) = x + 1) \wedge (0 \neq S(x)) \wedge (0 \neq x + 1)$
  - preserves validity: $M(KB \wedge (t_1 = t_2) \wedge \alpha \wedge \alpha\{t_1/t_2\}) = M(KB \wedge (t_1 = t_2) \wedge \alpha)$
  - α can be safely dropped from the result

- Equivalent-subformula substitution rule:

$$\frac{KB \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha}{KB \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha \wedge \alpha\{\beta_1/\beta_2\}}$$

  - ex: $(Even(x) \leftrightarrow Odd(S(x))) \wedge (Even(x) \vee Odd(x)) \Longrightarrow$ $(Even(x) \leftrightarrow Odd(S(x))) \wedge (Even(x) \vee Odd(x)) \wedge (Odd(S(x)) \vee Odd(x))$
  - preserves validity: $M(KB \wedge (\beta_1 = \beta_2) \wedge \alpha \wedge \alpha\{\beta_1/\beta_2\}) = M(KB \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha)$
  - α can be safely dropped from the result

Substitution with equivalent formulas/terms

- Equal-term substitution rule:

$$\frac{KB \wedge (t_1 = t_2) \wedge \alpha}{KB \wedge (t_1 = t_2) \wedge \alpha \wedge \alpha\{t_1/t_2\}}$$

  - ex: $(S(x) = x + 1) \wedge (0 \neq S(x))$ thus $(S(x) = x + 1) \wedge (0 \neq S(x)) \wedge (0 \neq x + 1)$
  - preserves validity: $M(KB \wedge (t_1 = t_2) \wedge \alpha \wedge \alpha\{t_1/t_2\}) = M(KB \wedge (t_1 = t_2) \wedge \alpha)$
  - α can be safely dropped from the result

- Equivalent-subformula substitution rule:

$$\frac{KB \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha}{KB \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha \wedge \alpha\{\beta_1/\beta_2\}}$$

  - ex: $(Even(x) \leftrightarrow Odd(S(x))) \wedge (Even(x) \vee Odd(x)) \Longrightarrow$ $(Even(x) \leftrightarrow Odd(S(x))) \wedge (Even(x) \vee Odd(x)) \wedge (Odd(S(x)) \vee Odd(x))$
  - preserves validity: $M(KB \wedge (\beta_1 = \beta_2) \wedge \alpha \wedge \alpha\{\beta_1/\beta_2\}) = M(KB \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha)$
  - α can be safely dropped from the result

Universal Instantiation (UI)

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

$$\frac{KB \wedge \forall x.\alpha}{KB \wedge \forall x.\alpha \wedge \alpha\{x/t\}}$$

for every variable x and term t

- Ex: $\forall(x.(King(x) \wedge Greedy(x)) \rightarrow Evil(x))$
  - $(King(John) \wedge Greedy(John)) \rightarrow Evil(John)$
  - $(King(Richard) \wedge Greedy(Richard)) \rightarrow Evil(Richard)$
  - $(King(Father(John)) \wedge Greedy(Father(John))) \rightarrow Evil(Father(John))$
  - $(King(Father(Father(John))) \wedge Greedy(Father(Father(John)))) \rightarrow Evil(Father(Father(John)))$
  - ...
- Preserves validity: $M(KB \wedge \forall x.\alpha \wedge \alpha\{x/t\}) = M(KB \wedge \forall x.\alpha)$

Universal Instantiation (UI)

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

$$\frac{KB \wedge \forall x.\alpha}{KB \wedge \forall x.\alpha \wedge \alpha\{x/t\}}$$

  for every variable x and term t

- Ex: $\forall(x.(King(x) \wedge Greedy(x)) \rightarrow Evil(x))$
  - $(King(John) \wedge Greedy(John)) \rightarrow Evil(John)$
  - $(King(Richard) \wedge Greedy(Richard)) \rightarrow Evil(Richard)$
  - $(King(Father(John)) \wedge Greedy(Father(John))) \rightarrow Evil(Father(John))$
  - $(King(Father(Father(John))) \wedge Greedy(Father(Father(John)))) \rightarrow Evil(Father(Father(John)))$
  - ...
- Preserves validity:
  $M(KB \wedge \forall x.\alpha \wedge \alpha\{x/t\}) = M(KB \wedge \forall x.\alpha)$

Universal Instantiation (UI)

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

$$\frac{KB \wedge \forall x.\alpha}{KB \wedge \forall x.\alpha \wedge \alpha\{x/t\}}$$

  for every variable x and term t

- Ex: $\forall(x.(King(x) \wedge Greedy(x)) \rightarrow Evil(x))$
  - $(King(John) \wedge Greedy(John)) \rightarrow Evil(John)$
  - $(King(Richard) \wedge Greedy(Richard)) \rightarrow Evil(Richard)$
  - $(King(Father(John)) \wedge Greedy(Father(John))) \rightarrow Evil(Father(John))$
  - $(King(Father(Father(John))) \wedge Greedy(Father(Father(John)))) \rightarrow Evil(Father(Father(John)))$
  - ...
- Preserves validity: $M(KB \wedge \forall x.\alpha \wedge \alpha\{x/t\}) = M(KB \wedge \forall x.\alpha)$

Universal Instantiation (UI)

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

$$\frac{KB \wedge \forall x.\alpha}{KB \wedge \forall x.\alpha \wedge \alpha\{x/t\}}$$

  for every variable x and term t
- Ex: $\forall(x.(King(x) \wedge Greedy(x)) \rightarrow Evil(x))$
  - $(King(John) \wedge Greedy(John)) \rightarrow Evil(John)$
  - $(King(Richard) \wedge Greedy(Richard)) \rightarrow Evil(Richard)$
  - $(King(Father(John)) \wedge Greedy(Father(John))) \rightarrow Evil(Father(John))$
  - $(King(Father(Father(John))) \wedge Greedy(Father(Father(John)))) \rightarrow Evil(Father(Father(John)))$
  - ...
- Preserves validity: $M(KB \wedge \forall x.\alpha \wedge \alpha\{x/t\}) = M(KB \wedge \forall x.\alpha)$

Existential Instantiation (EI)

- An existentially quantified-sentence can be substituted by one of its instantation with a fresh constant:
$$\frac{KB \wedge \exists x.\alpha}{KB \wedge \alpha\{x/c\}}$$
for every variable x and for a "fresh" constant c, i.e. a constant which does not appear in $KB \wedge \exists x.\alpha$
- c is called Skolem constant, EI subcase of Skolemization
- Intuition: if there is an object satisfying some condition, then we give a (new) name to such object
- Ex: $\exists x.(Crown(x) \wedge OnHead(x, John))$
  - $(Crown(c) \wedge OnHead(c, John))$
  - given "There is a crown on John's head", I call "c" such crown
- Preserves satisfiability (aka preserves inferential equivalence)
$M(KB \wedge \alpha\{x/t\}) \neq \emptyset$ *iff* $M(KB \wedge \exists x.\alpha) \neq \emptyset$
(i.e.. $(KB \wedge \alpha\{x/t\}) \models \beta$ iff $(KB \wedge \exists x.\alpha) \models \beta$, for every β)
- Ex from math: $\exists x.(\frac{d(x^y)}{dy} = x^y)$, we call it "e" $\Longrightarrow$ $(\frac{d(e^y)}{dy} = e^y)$

Existential Instantiation (EI)

- An existentially quantified-sentence can be substituted by one of its instantiation with a fresh constant:

$$\frac{KB \wedge \exists x.\alpha}{KB \wedge \alpha\{x/c\}}$$

  for every variable x and for a "fresh" constant c, i.e. a constant which does not appear in $KB \wedge \exists x.\alpha$
- c is called Skolem constant, EI subcase of Skolemization
- Intuition: if there is an object satisfying some condition, then we give a (new) name to such object
- Ex: $\exists x.(Crown(x) \wedge OnHead(x, John))$
  - $(Crown(c) \wedge OnHead(c, John))$
  - given "There is a crown on John's head", I call "c" such crown
- Preserves satisfiability (aka preserves inferential equivalence)
  $M(KB \wedge \alpha\{x/t\}) \neq \emptyset$ *iff* $M(KB \wedge \exists x.\alpha) \neq \emptyset$
  (i.e.. $(KB \wedge \alpha\{x/t\}) \models \beta$ iff $(KB \wedge \exists x.\alpha) \models \beta$, for every β)
- Ex from math: $\exists x.(\frac{d(x^y)}{dy} = x^y)$, we call it "e" $\Longrightarrow (\frac{d(e^y)}{dy} = e^y)$

Existential Instantiation (EI)

- An existentially quantified-sentence can be substituted by one of its instantiation with a fresh constant:

$$\frac{KB \wedge \exists x.\alpha}{KB \wedge \alpha\{x/c\}}$$

  for every variable x and for a "fresh" constant c, i.e. a constant which does not appear in $KB \wedge \exists x.\alpha$
- c is called Skolem constant, EI subcase of Skolemization
- Intuition: if there is an object satisfying some condition, then we give a (new) name to such object
- Ex: $\exists x.(Crown(x) \wedge OnHead(x, John))$
  - $(Crown(c) \wedge OnHead(c, John))$
  - given "There is a crown on John's head", I call "c" such crown
- Preserves satisfiability (aka preserves inferential equivalence)
  $M(KB \wedge \alpha\{x/t\}) \neq \emptyset$ *iff* $M(KB \wedge \exists x.\alpha) \neq \emptyset$
  (i.e.. $(KB \wedge \alpha\{x/t\}) \models \beta$ iff $(KB \wedge \exists x.\alpha) \models \beta$, for every β)
- Ex from math: $\exists x.(\frac{d(x^y)}{dy} = x^y)$, we call it "e" $\Longrightarrow (\frac{d(e^y)}{dy} = e^y)$

Existential Instantiation (EI)

- An existentially quantified-sentence can be substituted by one of its instantiation with a fresh constant:

$$\frac{KB \land \exists x.\alpha}{KB \land \alpha\{x/c\}}$$

for every variable x and for a "fresh" constant c, i.e. a constant which does not appear in $KB \land \exists x.\alpha$

- c is called Skolem constant, EI subcase of Skolemization
- Intuition: if there is an object satisfying some condition, then we give a (new) name to such object
- Ex: $\exists x.(Crown(x) \land OnHead(x, John))$
  - $(Crown(c) \land OnHead(c, John))$
  - given "There is a crown on John's head", I call "c" such crown
- Preserves satisfiability (aka preserves inferential equivalence)
  $M(KB \land \alpha\{x/t\}) \neq \emptyset$ *iff* $M(KB \land \exists x.\alpha) \neq \emptyset$
  (i.e.. $(KB \land \alpha\{x/t\}) \models \beta$ iff $(KB \land \exists x.\alpha) \models \beta$, for every β)
- Ex from math: $\exists x.(\frac{d(x^y)}{dy} = x^y)$, we call it "e" $\Longrightarrow (\frac{d(e^y)}{dy} = e^y)$

Existential Instantiation (EI)

- An existentially quantified-sentence can be substituted by one of its instantiation with a fresh constant:
$$\frac{KB \wedge \exists x.\alpha}{KB \wedge \alpha\{x/c\}}$$
for every variable x and for a "fresh" constant c, i.e. a constant which does not appear in $KB \wedge \exists x.\alpha$
- c is called Skolem constant, EI subcase of Skolemization
- Intuition: if there is an object satisfying some condition, then we give a (new) name to such object
- Ex: $\exists x.(Crown(x) \wedge OnHead(x, John))$
  - $(Crown(c) \wedge OnHead(c, John))$
  - given "There is a crown on John's head", I call "c" such crown
- Preserves satisfiability (aka preserves inferential equivalence)
$M(KB \wedge \alpha\{x/t\}) \neq \emptyset$ *iff* $M(KB \wedge \exists x.\alpha) \neq \emptyset$
(i.e.. $(KB \wedge \alpha\{x/t\}) \models \beta$ iff $(KB \wedge \exists x.\alpha) \models \beta$, for every β)
- Ex from math: $\exists x.(\frac{d(x^y)}{dy} = x^y)$, we call it "e" $\Longrightarrow (\frac{d(e^y)}{dy} = e^y)$

Existential Instantiation (EI)

- An existentially quantified-sentence can be substituted by one of its instantiation with a fresh constant:

$$\frac{KB \wedge \exists x.\alpha}{KB \wedge \alpha\{x/c\}}$$

 for every variable x and for a "fresh" constant c, i.e. a constant which does not appear in $KB \wedge \exists x.\alpha$

- c is called Skolem constant, EI subcase of Skolemization
- Intuition: if there is an object satisfying some condition, then we give a (new) name to such object
- Ex: $\exists x.(Crown(x) \wedge OnHead(x, John))$
  - $(Crown(c) \wedge OnHead(c, John))$
  - given "There is a crown on John's head", I call "c" such crown
- Preserves satisfiability (aka preserves inferential equivalence)
  $M(KB \wedge \alpha\{x/t\}) \neq \emptyset \;\; iff \;\; M(KB \wedge \exists x.\alpha) \neq \emptyset$
  (i.e.. $(KB \wedge \alpha\{x/t\}) \models \beta$ iff $(KB \wedge \exists x.\alpha) \models \beta$, for every β)
- Ex from math: $\exists x.(\frac{d(x^y)}{dy} = x^y)$, we call it "e" $\Longrightarrow$ $(\frac{d(e^y)}{dy} = e^y)$

Remarks

- About Universal Instantiation:
  - UI can be applied several times to add new sentences;
  - the new KB is logically equivalent to the old KB
- About Existential Instantiation:
  - EI can be applied once to replace the existential sentence;
  - the new KB is not equivalent to the old,
  - but is (un)satisfiable iff the old KB is (un)satisfiable

  $\Longrightarrow$ the new KB can infer β iff the old KB can infer β

Before applying UI or EI, sentences must be rewritten s.t. negations must be pushed inside the quantifications:

- $\neg\forall x.\alpha \Longrightarrow \exists x.\neg\alpha$
- $\neg\exists x.\alpha \Longrightarrow \forall x.\neg\alpha$

Remarks

- About Universal Instantiation:
  - UI can be applied several times to add new sentences;
  - the new KB is logically equivalent to the old KB
- About Existential Instantiation:
  - EI can be applied once to replace the existential sentence;
  - the new KB is not equivalent to the old,
  - but is (un)satisfiable iff the old KB is (un)satisfiable

  $\Longrightarrow$ the new KB can infer β iff the old KB can infer β

Before applying UI or EI, sentences must be rewritten s.t. negations must be pushed inside the quantifications:

- $\neg\forall x.\alpha \Longrightarrow \exists x.\neg\alpha$
- $\neg\exists x.\alpha \Longrightarrow \forall x.\neg\alpha$

Remarks

- About Universal Instantiation:
  - UI can be applied several times to add new sentences;
  - the new KB is logically equivalent to the old KB
- About Existential Instantiation:
  - EI can be applied once to replace the existential sentence;
  - the new KB is not equivalent to the old,
  - but is (un)satisfiable iff the old KB is (un)satisfiable

  $\Longrightarrow$ the new KB can infer β iff the old KB can infer β

Before applying UI or EI, sentences must be rewritten s.t. negations must be pushed inside the quantifications:

- $\neg\forall x.\alpha \Longrightarrow \exists x.\neg\alpha$
- $\neg\exists x.\alpha \Longrightarrow \forall x.\neg\alpha$

Reduction to Propositional Inference

- Idea: Convert ($KB \land \neg\alpha$) to PL (aka propositionalization)
  $\Longrightarrow$ use a PL SAT solver to check PL (un)satisfiability
- Trick:
  - replace variables with ground terms, creating all possible instantiations of quantified sentences
  - convert atomic sentences into propositional symbols

    e.g. "King(John)" $\Longrightarrow$ "King_John",
    e.g. "Brother(John,Richard)" $\Longrightarrow$ "Brother_John-Richard",
- Theorem: (Herbrand, 1930)
  If a ground sentence α is entailed by an FOL KB,
  then it is entailed by a finite subset of the propositional KB

  $\Longrightarrow$ A ground sentence is entailed by the propositionalized KB if it is entailed by original KB
  $\Longrightarrow$ Every FOL KB can be propositionalized s.t. to preserve entailment
- The vice-versa does not hold
  $\Longrightarrow$ works if α is entailed, loops if α is not entailed

Reduction to Propositional Inference

- Idea: Convert $(KB \wedge \neg\alpha)$ to PL (aka propositionalization)
  $\Longrightarrow$ use a PL SAT solver to check PL (un)satisfiability
- Trick:
  - replace variables with ground terms, creating all possible instantiations of quantified sentences
  - convert atomic sentences into propositional symbols

    e.g. "King(John)" $\Longrightarrow$ "King_John",
    e.g. "Brother(John,Richard)" $\Longrightarrow$ "Brother_John-Richard",
- Theorem: (Herbrand, 1930)
  If a ground sentence α is entailed by an FOL KB,
  then it is entailed by a finite subset of the propositional KB

  $\Longrightarrow$ A ground sentence is entailed by the propositionalized KB if it is entailed by original KB

  $\Longrightarrow$ Every FOL KB can be propositionalized s.t. to preserve entailment
- The vice-versa does not hold
  $\Longrightarrow$ works if α is entailed, loops if α is not entailed

Reduction to Propositional Inference

- Idea: Convert ($KB \land \neg\alpha$) to PL (aka propositionalization)
  $\Longrightarrow$ use a PL SAT solver to check PL (un)satisfiability
- Trick:
  - replace variables with ground terms, creating all possible instantiations of quantified sentences
  - convert atomic sentences into propositional symbols

    e.g. "King(John)" $\Longrightarrow$ "King_John",
    e.g. "Brother(John,Richard)" $\Longrightarrow$ "Brother_John-Richard",
- Theorem: (Herbrand, 1930)
  If a ground sentence α is entailed by an FOL KB,
  then it is entailed by a finite subset of the propositional KB
  $\Longrightarrow$ A ground sentence is entailed by the propositionalized KB if it is entailed by original KB
  $\Longrightarrow$ Every FOL KB can be propositionalized s.t. to preserve entailment
- The vice-versa does not hold
  $\Longrightarrow$ works if α is entailed, loops if α is not entailed

Reduction to Propositional Inference

- Idea: Convert $(KB \wedge \neg\alpha)$ to PL (aka propositionalization)
  $\Longrightarrow$ use a PL SAT solver to check PL (un)satisfiability
- Trick:
  - replace variables with ground terms, creating all possible instantiations of quantified sentences
  - convert atomic sentences into propositional symbols

    e.g. "King(John)" $\Longrightarrow$ "King_John",
    e.g. "Brother(John,Richard)" $\Longrightarrow$ "Brother_John-Richard",
- Theorem: (Herbrand, 1930)
  If a ground sentence α is entailed by an FOL KB,
  then it is entailed by a finite subset of the propositional KB
  $\Longrightarrow$ A ground sentence is entailed by the propositionalized KB if it is entailed by original KB
  $\Longrightarrow$ Every FOL KB can be propositionalized s.t. to preserve entailment
- The vice-versa does not hold
  $\Longrightarrow$ works if α is entailed, loops if α is not entailed

Reduction to Propositional Inference: Example

- Suppose the KB contains only:

  $\forall x.((King(x) \land Greedy(x)) \rightarrow Evil(x))$
  $King(John)$
  $Greedy(John)$
  $Brother(Richard, John)$

- Instantiating the universal sentence in all possible ways:

  $(King(John) \land Greedy(John)) \rightarrow Evil(John)$
  $(King(Richard) \land Greedy(Richard)) \rightarrow Evil(Richard)$
  $King(John)$
  $Greedy(John)$
  $Brother(Richard, John)$

- The new KB is propositionalized:

  $(King\_John \land Greedy\_John) \rightarrow Evil\_John$
  $(King\_Richard \land Greedy\_Richard) \rightarrow Evil\_Richard$
  $King\_John$
  $Greedy\_John$
  $Brother\_Richard\text{-}John$

- *Evil_John* entailed by new KB ($Evil(John)$ entailed by old KB)

Reduction to Propositional Inference: Example

- Suppose the KB contains only:

  $\forall x.((King(x) \land Greedy(x)) \rightarrow Evil(x))$
  $King(John)$
  $Greedy(John)$
  $Brother(Richard, John)$

- Instantiating the universal sentence in all possible ways:

  $(King(John) \land Greedy(John)) \rightarrow Evil(John)$
  $(King(Richard) \land Greedy(Richard)) \rightarrow Evil(Richard)$
  $King(John)$
  $Greedy(John)$
  $Brother(Richard, John)$

- The new KB is propositionalized:

  $(King\_John \land Greedy\_John) \rightarrow Evil\_John$
  $(King\_Richard \land Greedy\_Richard) \rightarrow Evil\_Richard$
  $King\_John$
  $Greedy\_John$
  $Brother\_Richard\text{-}John$

- *Evil_John* entailed by new KB ($Evil(John)$ entailed by old KB)

Reduction to Propositional Inference: Example

- Suppose the KB contains only:

  $\forall x.((King(x) \land Greedy(x)) \rightarrow Evil(x))$
  $King(John)$
  $Greedy(John)$
  $Brother(Richard, John)$

- Instantiating the universal sentence in all possible ways:

  $(King(John) \land Greedy(John)) \rightarrow Evil(John)$
  $(King(Richard) \land Greedy(Richard)) \rightarrow Evil(Richard)$
  $King(John)$
  $Greedy(John)$
  $Brother(Richard, John)$

- The new KB is propositionalized:

  $(King\_John \land Greedy\_John) \rightarrow Evil\_John$
  $(King\_Richard \land Greedy\_Richard) \rightarrow Evil\_Richard$
  $King\_John$
  $Greedy\_John$
  $Brother\_Richard\text{-}John$

- $Evil\_John$ entailed by new KB ($Evil(John)$ entailed by old KB)

Reduction to Propositional Inference: Example

- Suppose the KB contains only:

  $\forall x.((King(x) \land Greedy(x)) \rightarrow Evil(x))$
  $King(John)$
  $Greedy(John)$
  $Brother(Richard, John)$

- Instantiating the universal sentence in all possible ways:

  $(King(John) \land Greedy(John)) \rightarrow Evil(John)$
  $(King(Richard) \land Greedy(Richard)) \rightarrow Evil(Richard)$
  $King(John)$
  $Greedy(John)$
  $Brother(Richard, John)$

- The new KB is propositionalized:

  (King_John ∧ Greedy_John) → Evil_John
  (King_Richard ∧ Greedy_Richard) → Evil_Richard
  King_John
  Greedy_John
  Brother_Richard-John

- *Evil_John* entailed by new KB (*Evil(John)* entailed by old KB)

Problems with Propositionalization

- Propositionalization generates lots of irrelevant sentences

  produces irrelevant atoms like Greedy(Richard)

  $\forall x.((King(x) \wedge Greedy(x)) \rightarrow Evil(x))$
  $King(John)$
  $\forall y.Greedy(y)$
  $Brother(Richard, John)$

$\Longrightarrow$ produces irrelevant atoms like Greedy(Richard)

- With p k-ary predicates and n constants, $p \cdot n^k$ instantiations
- What happens with function symbols?

Problems with Propositionalization

- Propositionalization generates lots of irrelevant sentences

  produces irrelevant atoms like Greedy(Richard)

  $\forall x.((King(x) \wedge Greedy(x)) \rightarrow Evil(x))$
  $King(John)$
  $\forall y.Greedy(y)$
  $Brother(Richard, John)$

$\Longrightarrow$ produces irrelevant atoms like Greedy(Richard)

- With p k-ary predicates and n constants, $p \cdot n^k$ instantiations
- What happens with function symbols?

Problems with Propositionalization

- Propositionalization generates lots of irrelevant sentences

  produces irrelevant atoms like Greedy(Richard)

  $\forall x.((King(x) \land Greedy(x)) \rightarrow Evil(x))$
  $King(John)$
  $\forall y.Greedy(y)$
  $Brother(Richard, John)$

$\Longrightarrow$ produces irrelevant atoms like Greedy(Richard)

- With p k-ary predicates and n constants, $p \cdot n^k$ instantiations
- What happens with function symbols?

Problems with Propositionalization

- Propositionalization generates lots of irrelevant sentences

  produces irrelevant atoms like Greedy(Richard)

  $\forall x.((King(x) \wedge Greedy(x)) \rightarrow Evil(x))$
  $King(John)$
  $\forall y.Greedy(y)$
  $Brother(Richard, John)$

$\Longrightarrow$ produces irrelevant atoms like Greedy(Richard)

- With p k-ary predicates and n constants, $p \cdot n^k$ instantiations
- What happens with function symbols?

Problems with Propositionalization [cont.]

- Problem: nested function applications
  - e.g. Father(John), Father(Father(John)), Father(Father(Father(John))), ...

  $\Longrightarrow$ infinite instantiations
- Actual Trick: for k = 0 to ∞, use terms of function nesting depth k
  - create propositionalized KB by instantiating depth-k terms
  - if $KB \models \alpha$, then will find a contradiction for some finite k
  - if $KB \not\models \alpha$, may find a loop forever
- Theorem: (Turing, 1936), (Church, 1936):
  Entailment in FOL is semidecidable

Problems with Propositionalization [cont.]

- Problem: nested function applications
  - e.g. Father(John), Father(Father(John)), Father(Father(Father(John))), ...

  $\Longrightarrow$ infinite instantiations
- Actual Trick: for k = 0 to ∞, use terms of function nesting depth k
  - create propositionalized KB by instantiating depth-k terms
  - if $KB \models \alpha$, then will find a contradiction for some finite k
  - if $KB \not\models \alpha$, may find a loop forever
- Theorem: (Turing, 1936), (Church, 1936):
  Entailment in FOL is semidecidable

Problems with Propositionalization [cont.]

- Problem: nested function applications
  - e.g. Father(John), Father(Father(John)), Father(Father(Father(John))), ...

  $\Longrightarrow$ infinite instantiations
- Actual Trick: for k = 0 to ∞, use terms of function nesting depth k
  - create propositionalized KB by instantiating depth-k terms
  - if $KB \models \alpha$, then will find a contradiction for some finite k
  - if $KB \not\models \alpha$, may find a loop forever
- Theorem: (Turing, 1936), (Church, 1936): Entailment in FOL is semidecidable

Problems with Propositionalization [cont.]

- Problem: nested function applications
  - e.g. Father(John), Father(Father(John)), Father(Father(Father(John))), ...

  $\Longrightarrow$ infinite instantiations
- Actual Trick: for k = 0 to ∞, use terms of function nesting depth k
  - create propositionalized KB by instantiating depth-k terms
  - if $KB \models \alpha$, then will find a contradiction for some finite k
  - if $KB \not\models \alpha$, may find a loop forever
- Theorem: (Turing, 1936), (Church, 1936):
  Entailment in FOL is semidecidable

Outline

1 Propositional vs. First-Order Inference

2 Unification and Lifting

3 Forward & Backward Chaining for Definite FOL KBs

4 Resolution for General FOL KBs

Generalized Modus Ponens (GMP)

- "Lifted inference": Combine PL inference with UI/EI
- Aristotle's "Modus Ponens" syllogism:
  "All men are mortal; Socrates is a man; thus Socrates is mortal."

$$\frac{Man(Socrates) \quad \forall x.(Man(x) \rightarrow Mortal(x))}{Mortal(Socrates)}$$

- Generalized Modus Ponens:
  if exists a substitution θ s.t., for all $i \in 1..k$, $\alpha'_i\theta = \alpha_i\theta$, then

$$\frac{\alpha'_1, \ \alpha'_2, \ ..., \ \alpha'_k, \ (\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_k) \rightarrow \beta}{\beta\theta}$$

  - all variables (implicitly) assumed as universally quantified
  - θ substitutes (universally quantified) variables with terms
- Ex: using $\theta \stackrel{\text{def}}{=} \{x/John, y/John\}$ we can infer Evil(John) from:
  $\forall x.((King(x) \wedge Greedy(x)) \rightarrow Evil(x)), King(John), \forall y.Greedy(y)$
- GMP used w. KB of definite clauses (exactly one positive literal)
  - Used in Prolog, Datalog, Production-rule systems,...

Generalized Modus Ponens (GMP)

- "Lifted inference": Combine PL inference with UI/EI
- Aristotle's "Modus Ponens" syllogism:
  "All men are mortal; Socrates is a man; thus Socrates is mortal."

$$\frac{Man(Socrates) \quad \forall x.(Man(x) \rightarrow Mortal(x))}{Mortal(Socrates)}$$

- Generalized Modus Ponens:
  if exists a substitution θ s.t., for all $i \in 1..k$, $\alpha_i'\theta = \alpha_i\theta$, then

$$\frac{\alpha_1',\ \alpha_2',\ ...,\ \alpha_k',\ (\alpha_1 \land \alpha_2 \land ... \land \alpha_k) \rightarrow \beta}{\beta\theta}$$

  - all variables (implicitly) assumed as universally quantified
  - θ substitutes (universally quantified) variables with terms
- Ex: using $\theta \stackrel{\text{def}}{=} \{x/John, y/John\}$ we can infer Evil(John) from:
  $\forall x.((King(x) \land Greedy(x)) \rightarrow Evil(x)), King(John), \forall y.Greedy(y)$
- GMP used w. KB of definite clauses (exactly one positive literal)
  - Used in Prolog, Datalog, Production-rule systems,...

Generalized Modus Ponens (GMP)

- "Lifted inference": Combine PL inference with UI/EI
- Aristotle's "Modus Ponens" syllogism:
  "All men are mortal; Socrates is a man; thus Socrates is mortal."

$$\frac{Man(Socrates) \quad \forall x.(Man(x) \rightarrow Mortal(x))}{Mortal(Socrates)}$$

- Generalized Modus Ponens:
  if exists a substitution θ s.t., for all $i \in 1..k$, $\alpha'_i\theta = \alpha_i\theta$, then

$$\frac{\alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k,\ (\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_k) \rightarrow \beta}{\beta\theta}$$

  - all variables (implicitly) assumed as universally quantified
  - θ substitutes (universally quantified) variables with terms
- Ex: using $\theta \stackrel{\text{def}}{=} \{x/John, y/John\}$ we can infer Evil(John) from:
  $\forall x.((King(x) \wedge Greedy(x)) \rightarrow Evil(x)), King(John), \forall y.Greedy(y)$
- GMP used w. KB of definite clauses (exactly one positive literal)
  - Used in Prolog, Datalog, Production-rule systems,...

Generalized Modus Ponens (GMP)

- "Lifted inference": Combine PL inference with UI/EI
- Aristotle's "Modus Ponens" syllogism:
  "All men are mortal; Socrates is a man; thus Socrates is mortal."

$$\frac{Man(Socrates) \quad \forall x.(Man(x) \rightarrow Mortal(x))}{Mortal(Socrates)}$$

- Generalized Modus Ponens:
  if exists a substitution θ s.t., for all $i \in 1..k$, $\alpha'_i\theta = \alpha_i\theta$, then

$$\frac{\alpha'_1,\ \alpha'_2,\ \ldots,\ \alpha'_k,\ (\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_k) \rightarrow \beta}{\beta\theta}$$

  - all variables (implicitly) assumed as universally quantified
  - θ substitutes (universally quantified) variables with terms
- Ex: using $\theta \stackrel{\text{def}}{=} \{x/John, y/John\}$ we can infer Evil(John) from:
  $\forall x.((King(x) \wedge Greedy(x)) \rightarrow Evil(x)), King(John), \forall y.Greedy(y)$
- GMP used w. KB of definite clauses (exactly one positive literal)
  - Used in Prolog, Datalog, Production-rule systems,...

Generalized Modus Ponens (GMP)

- "Lifted inference": Combine PL inference with UI/EI
- Aristotle's "Modus Ponens" syllogism:
  "All men are mortal; Socrates is a man; thus Socrates is mortal."

$$\frac{Man(Socrates) \quad \forall x.(Man(x) \rightarrow Mortal(x))}{Mortal(Socrates)}$$

- Generalized Modus Ponens:
  if exists a substitution θ s.t., for all $i \in 1..k$, $\alpha'_i\theta = \alpha_i\theta$, then

$$\frac{\alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k,\ (\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_k) \rightarrow \beta}{\beta\theta}$$

  - all variables (implicitly) assumed as universally quantified
  - θ substitutes (universally quantified) variables with terms
- Ex: using $\theta \stackrel{\text{def}}{=} \{x/John, y/John\}$ we can infer Evil(John) from:
  $\forall x.((King(x) \wedge Greedy(x)) \rightarrow Evil(x)), King(John), \forall y.Greedy(y)$
- GMP used w. KB of definite clauses (exactly one positive literal)
  - Used in Prolog, Datalog, Production-rule systems,...

Unification

- Unification: Given $\langle \alpha'_1, \alpha'_2, ..., \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i\theta = \alpha_i\theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1, \alpha'_2, ..., \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1, \alpha'_2, ..., \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i\theta = \alpha_i\theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1, \alpha'_2, ..., \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i\theta = \alpha_i\theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1, \ \alpha'_2, \ ..., \ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i\theta = \alpha_i\theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1, \ \alpha'_2, \ ..., \ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1, \alpha'_2, ..., \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i\theta = \alpha_i\theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1, \alpha'_2, ..., \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes

$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i\theta = \alpha_i\theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1, \ \alpha'_2, \ ..., \ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i\theta = \alpha_i\theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1, \ \alpha'_2, \ ..., \ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1, \ \alpha'_2, \ \dots, \ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, \dots, \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i \theta = \alpha_i \theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1, \ \alpha'_2, \ \dots, \ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, \dots, \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i\theta = \alpha_i\theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1, \ \alpha'_2, \ ..., \ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i\theta = \alpha_i\theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1, \ \alpha'_2, \ ..., \ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i\theta = \alpha_i\theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1, \alpha'_2, ..., \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i\theta = \alpha_i\theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1, \alpha'_2, ..., \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1, \alpha'_2, ..., \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i\theta = \alpha_i\theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1, \alpha'_2, ..., \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

- $\{\forall x.\alpha, \forall x.\beta\} \Longleftrightarrow \{\forall x_1.\alpha\{x/x_1\}, \forall x_2.\beta\{x/x_2\}\}$, s.t. x_1, x_2 new

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i\theta = \alpha_i\theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\quad \{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

- $\{\forall x.\alpha, \forall x.\beta\} \Longleftrightarrow \{\forall x_1.\alpha\{x/x_1\}, \forall x_2.\beta\{x/x_2\}\}$, s.t. x_1, x_2 new

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i\theta = \alpha_i\theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

- $\{\forall x.\alpha, \forall x.\beta\} \Longleftrightarrow \{\forall x_1.\alpha\{x/x_1\}, \forall x_2.\beta\{x/x_2\}\}$, s.t. x_1, x_2 new

Unification

- Unification: Given $\langle \alpha'_1, \alpha'_2, ..., \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a substitution θ s.t. θ s.t. $\alpha'_i\theta = \alpha_i\theta$, for all $i \in 1..k$
  - θ is called a unifier for $\langle \alpha'_1, \alpha'_2, ..., \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $Unify(Knows(John, x), Knows(John, Jane)) = \{x/Jane\}$
  $Unify(Knows(John, x), Knows(y, OJ)) = \{x/OJ, y/John\}$
  $Unify(Knows(John, x), Knows(y, Mother(y))) =$
  $\{y/John, x/Mother(John)\}$
  $Unify(Knows(John, x), Knows(x, OJ)) = FAIL : x/?$
- different (implicitly-universally-quantified) formulas should use different variables

$\Longrightarrow$ (standardizing apart): rename variables to avoid name clashes
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OBJ, x_2/John\}$

  - $\{\forall x.\alpha, \forall x.\beta\} \Longleftrightarrow \{\forall x_1.\alpha\{x/x_1\}, \forall x_2.\beta\{x/x_2\}\}$, s.t. x_1, x_2 new

Most-General Unifier (MGU)

- Unifiers are not unique
  - ex: $Unify(Knows(John, x), Knows(y, z))$ could return $\{y/John, x/z\}$ or $\{y/John, x/John, z/John\}$
- Given α, β, the unifier θ_1 is more general than the unifier θ_2 for α, β if exists θ_3 s.t. $\theta_2 = \theta_1\theta_3$
  - ex: $\{y/John, x/z\}$ more general than $\{y/John, x/John, z/John\}$: $\{y/John, x/John, z/John\} = \{y/John, x/z\}\{z/John\}$
- Theorem: If exists an unifier for α, β, then exists a most general unifier (MGU) θ for α, β
  - Ex: $\{y/John, x/z\}$ MGU for $Knows(John, x), Knows(y, z)$
  - Ex: an MGU is unique modulo variable renaming
- UNIFY() returns the MGU between two (lists of) formulas
  - efficiency optimizations based on predicate/term indexing techniques (see AIMA if interested)

Most-General Unifier (MGU)

- Unifiers are not unique
  - ex: $Unify(Knows(John, x), Knows(y, z))$ could return $\{y/John, x/z\}$ or $\{y/John, x/John, z/John\}$
- Given α, β, the unifier θ_1 is more general than the unifier θ_2 for α, β if exists θ_3 s.t. $\theta_2 = \theta_1\theta_3$
  - ex: $\{y/John, x/z\}$ more general than $\{y/John, x/John, z/John\}$: $\{y/John, x/John, z/John\} = \{y/John, x/z\}\{z/John\}$
- Theorem: If exists an unifier for α, β, then exists a most general unifier (MGU) θ for α, β
  - Ex: $\{y/John, x/z\}$ MGU for $Knows(John, x), Knows(y, z)$
  - Ex: an MGU is unique modulo variable renaming
- UNIFY() returns the MGU between two (lists of) formulas
  - efficiency optimizations based on predicate/term indexing techniques (see AIMA if interested)

Most-General Unifier (MGU)

- Unifiers are not unique
  - ex: $Unify(Knows(John, x), Knows(y, z))$ could return $\{y/John, x/z\}$ or $\{y/John, x/John, z/John\}$
- Given α, β, the unifier θ_1 is more general than the unifier θ_2 for α, β if exists θ_3 s.t. $\theta_2 = \theta_1\theta_3$
  - ex: $\{y/John, x/z\}$ more general than $\{y/John, x/John, z/John\}$: $\{y/John, x/John, z/John\} = \{y/John, x/z\}\{z/John\}$
- Theorem: If exists an unifier for α, β, then exists a most general unifier (MGU) θ for α, β
  - Ex: $\{y/John, x/z\}$ MGU for $Knows(John, x), Knows(y, z)$
  - Ex: an MGU is unique modulo variable renaming
- UNIFY() returns the MGU between two (lists of) formulas
  - efficiency optimizations based on predicate/term indexing techniques (see AIMA if interested)

Most-General Unifier (MGU)

- Unifiers are not unique
  - ex: $Unify(Knows(John, x), Knows(y, z))$ could return $\{y/John, x/z\}$ or $\{y/John, x/John, z/John\}$
- Given α, β, the unifier θ_1 is more general than the unifier θ_2 for α, β if exists θ_3 s.t. $\theta_2 = \theta_1\theta_3$
  - ex: $\{y/John, x/z\}$ more general than $\{y/John, x/John, z/John\}$: $\{y/John, x/John, z/John\} = \{y/John, x/z\}\{z/John\}$
- Theorem: If exists an unifier for α, β, then exists a most general unifier (MGU) θ for α, β
  - Ex: $\{y/John, x/z\}$ MGU for $Knows(John, x), Knows(y, z)$
  - Ex: an MGU is unique modulo variable renaming
- UNIFY() returns the MGU between two (lists of) formulas
  - efficiency optimizations based on predicate/term indexing techniques (see AIMA if interested)

Most-General Unifier (MGU)

- Unifiers are not unique
  - ex: $Unify(Knows(John, x), Knows(y, z))$ could return $\{y/John, x/z\}$ or $\{y/John, x/John, z/John\}$
- Given α, β, the unifier θ_1 is more general than the unifier θ_2 for α, β if exists θ_3 s.t. $\theta_2 = \theta_1\theta_3$
  - ex: $\{y/John, x/z\}$ more general than $\{y/John, x/John, z/John\}$: $\{y/John, x/John, z/John\} = \{y/John, x/z\}\{z/John\}$
- Theorem: If exists an unifier for α, β, then exists a most general unifier (MGU) θ for α, β
  - Ex: $\{y/John, x/z\}$ MGU for $Knows(John, x), Knows(y, z)$
  - Ex: an MGU is unique modulo variable renaming
- UNIFY() returns the MGU between two (lists of) formulas
  - efficiency optimizations based on predicate/term indexing techniques (see AIMA if interested)

The Procedure Unify

function UNIFY(x, y, θ) **returns** a substitution to make x and y identical
  inputs: x, a variable, constant, list, or compound expression
    y, a variable, constant, list, or compound expression
    θ, the substitution built up so far (optional, defaults to empty)

  if θ = failure **then return** failure
  else if $x = y$ **then return** θ
  else if VARIABLE?(x) **then return** UNIFY-VAR(x, y, θ)
  else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, θ)
  else if COMPOUND?(x) **and** COMPOUND?(y) **then**
    return UNIFY(x.ARGS, y.ARGS, UNIFY(x.OP, y.OP, θ))
  else if LIST?(x) **and** LIST?(y) **then**
    return UNIFY(x.REST, y.REST, UNIFY(x.FIRST, y.FIRST, θ))
  else return failure

---

function UNIFY-VAR(var, x, θ) **returns** a substitution

  if $\{var/val\} \in \theta$ **then return** UNIFY(val, x, θ)
  else if $\{x/val\} \in \theta$ **then return** UNIFY(var, val, θ)
  else if OCCUR-CHECK?(var, x) **then return** failure
  else return add $\{var/x\}$ to θ

Outline

First-Order Definite Clauses

- FOL Definite Clauses: clauses with exactly one positive literal
  - we omit universal quantifiers

  $\Longrightarrow$ variables are (implicitly) universally quantified
  - we remove existential quantifiers by EI

  $\Longrightarrow$ existentially-quantified variables are substituted by fresh constants (we assume no $\exists$ under the scope of $\forall$, see later for general case)
- Represent implications of atomic formulas
  - Ex: $\forall x.((King(x) \wedge Greedy(x)) \rightarrow Evil(x))$

  $\Longrightarrow$ $(\neg King(x) \vee \neg Greedy(x) \vee Evil(x))$
- Important subcase: Datalog KBs:
  sets of FOL definite clauses without function symbols
  - can represent statements typically made in relational databases
  - makes inference much easier

First-Order Definite Clauses

- FOL Definite Clauses: clauses with exactly one positive literal
  - we omit universal quantifiers

  $\Longrightarrow$ variables are (implicitly) universally quantified
  - we remove existential quantifiers by EI

  $\Longrightarrow$ existentially-quantified variables are substituted by fresh constants (we assume no $\exists$ under the scope of $\forall$, see later for general case)
- Represent implications of atomic formulas
  - Ex: $\forall x.((King(x) \wedge Greedy(x)) \rightarrow Evil(x))$

  $\Longrightarrow$ $(\neg King(x) \vee \neg Greedy(x) \vee Evil(x))$
- Important subcase: Datalog KBs: sets of FOL definite clauses without function symbols
  - can represent statements typically made in relational databases
  - makes inference much easier

First-Order Definite Clauses

- FOL Definite Clauses: clauses with exactly one positive literal
  - we omit universal quantifiers

  $\Longrightarrow$ variables are (implicitly) universally quantified
  - we remove existential quantifiers by EI

  $\Longrightarrow$ existentially-quantified variables are substituted by fresh constants (we assume no $\exists$ under the scope of $\forall$, see later for general case)
- Represent implications of atomic formulas
  - Ex: $\forall x.((King(x) \land Greedy(x)) \rightarrow Evil(x))$

  $\Longrightarrow$ $(\neg King(x) \lor \neg Greedy(x) \lor Evil(x))$
- Important subcase: Datalog KBs:
  sets of FOL definite clauses without function symbols
  - can represent statements typically made in relational databases
  - makes inference much easier

Example (Datalog)

KB: The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.
Goal: Prove that Col. West is a criminal.

Example (Datalog) [cont.]

- it is a crime for an American to sell weapons to hostile nations:
  $\forall x, y, z.((American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x, y, z)) \rightarrow Criminal(x))$

$\Longrightarrow$ $\neg American(x) \vee \neg Weapon(y) \vee \neg Hostile(z) \vee \neg Sells(x, y, z) \vee Criminal(x)$

- Nono ... has some missiles
  $\exists x.(Owns(Nono, x) \wedge Missile(x)) \Longrightarrow Owns(Nono, M_1) \wedge Missile(M_1)$
- All of its missiles were sold to it by Colonel West
  $\forall x.((Missile(x) \wedge Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$

$\Longrightarrow$ $\neg Missile(x) \vee \neg Owns(Nono, x) \vee Sells(West, x, Nono)$

- Missiles are weapons:
  $\forall x.(Missile(x) \rightarrow Weapon(x)) \Longrightarrow \neg Missile(x) \vee Weapon(x)$
- An enemy of America counts as "hostile":
  $\forall x.(Enemy(x, America) \rightarrow Hostile(x))$

$\Longrightarrow$ $\neg Enemy(x, America) \vee Hostile(x)$

- West, who is American ...: $American(West)$
- The country Nono, an enemy of America ...:
  $Enemy(Nono, America)$

Example (Datalog) [cont.]

- it is a crime for an American to sell weapons to hostile nations: $\forall x,y,z.((American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x,y,z)) \rightarrow Criminal(x))$

$\Longrightarrow$ $\neg American(x) \vee \neg Weapon(y) \vee \neg Hostile(z) \vee \neg Sells(x,y,z) \vee Criminal(x)$

- Nono ... has some missiles
  $\exists x.(Owns(Nono,x) \wedge Missile(x)) \Longrightarrow Owns(Nono,M_1) \wedge Missile(M_1)$
- All of its missiles were sold to it by Colonel West
  $\forall x.((Missile(x) \wedge Owns(Nono,x)) \rightarrow Sells(West,x,Nono))$

$\Longrightarrow$ $\neg Missile(x) \vee \neg Owns(Nono,x) \vee Sells(West,x,Nono)$

- Missiles are weapons:
  $\forall x.(Missile(x) \rightarrow Weapon(x)) \Longrightarrow \neg Missile(x) \vee Weapon(x)$
- An enemy of America counts as “hostile”:
  $\forall x.(Enemy(x,America) \rightarrow Hostile(x))$

$\Longrightarrow$ $\neg Enemy(x,America) \vee Hostile(x)$

- West, who is American ...: $American(West)$
- The country Nono, an enemy of America ...:
  $Enemy(Nono,America)$

Example (Datalog) [cont.]

- it is a crime for an American to sell weapons to hostile nations:
  $\forall x, y, z.((American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x, y, z)) \rightarrow Criminal(x))$

$\Longrightarrow$ $\neg American(x) \vee \neg Weapon(y) \vee \neg Hostile(z) \vee \neg Sells(x, y, z) \vee Criminal(x)$

- Nono ... has some missiles
  $\exists x.(Owns(Nono, x) \wedge Missile(x)) \Longrightarrow Owns(Nono, M_1) \wedge Missile(M_1)$
- All of its missiles were sold to it by Colonel West
  $\forall x.((Missile(x) \wedge Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$

$\Longrightarrow$ $\neg Missile(x) \vee \neg Owns(Nono, x) \vee Sells(West, x, Nono)$

- Missiles are weapons:
  $\forall x.(Missile(x) \rightarrow Weapon(x)) \Longrightarrow \neg Missile(x) \vee Weapon(x)$
- An enemy of America counts as “hostile”:
  $\forall x.(Enemy(x, America) \rightarrow Hostile(x))$

$\Longrightarrow$ $\neg Enemy(x, America) \vee Hostile(x)$

- West, who is American ...: $American(West)$
- The country Nono, an enemy of America ...:
  $Enemy(Nono, America)$

Example (Datalog) [cont.]

- it is a crime for an American to sell weapons to hostile nations:
  $\forall x, y, z.((American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x, y, z)) \rightarrow Criminal(x))$

$\Longrightarrow$ $\neg American(x) \vee \neg Weapon(y) \vee \neg Hostile(z) \vee \neg Sells(x, y, z) \vee Criminal(x)$

- Nono ... has some missiles
  $\exists x.(Owns(Nono, x) \wedge Missile(x)) \Longrightarrow Owns(Nono, M_1) \wedge Missile(M_1)$
- All of its missiles were sold to it by Colonel West
  $\forall x.((Missile(x) \wedge Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$

$\Longrightarrow$ $\neg Missile(x) \vee \neg Owns(Nono, x) \vee Sells(West, x, Nono)$

- Missiles are weapons:
  $\forall x.(Missile(x) \rightarrow Weapon(x)) \Longrightarrow \neg Missile(x) \vee Weapon(x)$
- An enemy of America counts as “hostile”:
  $\forall x.(Enemy(x, America) \rightarrow Hostile(x))$

$\Longrightarrow$ $\neg Enemy(x, America) \vee Hostile(x)$

- West, who is American ...: $American(West)$
- The country Nono, an enemy of America ...:
  $Enemy(Nono, America)$

Example (Datalog) [cont.]

- it is a crime for an American to sell weapons to hostile nations:
  $\forall x, y, z.((American(x) \land Weapon(y) \land Hostile(z) \land Sells(x, y, z)) \rightarrow Criminal(x))$

$\Longrightarrow \neg American(x) \lor \neg Weapon(y) \lor \neg Hostile(z) \lor \neg Sells(x, y, z) \lor Criminal(x)$

- Nono ... has some missiles
  $\exists x.(Owns(Nono, x) \land Missile(x)) \Longrightarrow Owns(Nono, M_1) \land Missile(M_1)$
- All of its missiles were sold to it by Colonel West
  $\forall x.((Missile(x) \land Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$

$\Longrightarrow \neg Missile(x) \lor \neg Owns(Nono, x) \lor Sells(West, x, Nono)$

- Missiles are weapons:
  $\forall x.(Missile(x) \rightarrow Weapon(x)) \Longrightarrow \neg Missile(x) \lor Weapon(x)$
- An enemy of America counts as “hostile”:
  $\forall x.(Enemy(x, America) \rightarrow Hostile(x))$

$\Longrightarrow \neg Enemy(x, America) \lor Hostile(x)$

- West, who is American ...: $American(West)$
- The country Nono, an enemy of America ...:
  $Enemy(Nono, America)$

Example (Datalog) [cont.]

- it is a crime for an American to sell weapons to hostile nations:
  $\forall x, y, z.((American(x) \land Weapon(y) \land Hostile(z) \land Sells(x, y, z)) \rightarrow Criminal(x))$

$\Longrightarrow$ $\neg American(x) \lor \neg Weapon(y) \lor \neg Hostile(z) \lor \neg Sells(x, y, z) \lor Criminal(x)$

- Nono ... has some missiles
  $\exists x.(Owns(Nono, x) \land Missile(x)) \Longrightarrow Owns(Nono, M_1) \land Missile(M_1)$
- All of its missiles were sold to it by Colonel West
  $\forall x.((Missile(x) \land Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$

$\Longrightarrow$ $\neg Missile(x) \lor \neg Owns(Nono, x) \lor Sells(West, x, Nono)$

- Missiles are weapons:
  $\forall x.(Missile(x) \rightarrow Weapon(x)) \Longrightarrow \neg Missile(x) \lor Weapon(x)$
- An enemy of America counts as “hostile”:
  $\forall x.(Enemy(x, America) \rightarrow Hostile(x))$

$\Longrightarrow$ $\neg Enemy(x, America) \lor Hostile(x)$

- West, who is American ...: $American(West)$
- The country Nono, an enemy of America ...:
  $Enemy(Nono, America)$

Example (Datalog) [cont.]

- it is a crime for an American to sell weapons to hostile nations:
  $\forall x, y, z.((American(x) \land Weapon(y) \land Hostile(z) \land Sells(x, y, z)) \rightarrow Criminal(x))$

$\Longrightarrow$ $\neg American(x) \lor \neg Weapon(y) \lor \neg Hostile(z) \lor \neg Sells(x, y, z) \lor Criminal(x)$

- Nono ... has some missiles
  $\exists x.(Owns(Nono, x) \land Missile(x)) \Longrightarrow Owns(Nono, M_1) \land Missile(M_1)$
- All of its missiles were sold to it by Colonel West
  $\forall x.((Missile(x) \land Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$

$\Longrightarrow$ $\neg Missile(x) \lor \neg Owns(Nono, x) \lor Sells(West, x, Nono)$

- Missiles are weapons:
  $\forall x.(Missile(x) \rightarrow Weapon(x)) \Longrightarrow \neg Missile(x) \lor Weapon(x)$
- An enemy of America counts as “hostile”:
  $\forall x.(Enemy(x, America) \rightarrow Hostile(x))$

$\Longrightarrow$ $\neg Enemy(x, America) \lor Hostile(x)$

- West, who is American ...: $American(West)$
- The country Nono, an enemy of America ...:
  $Enemy(Nono, America)$

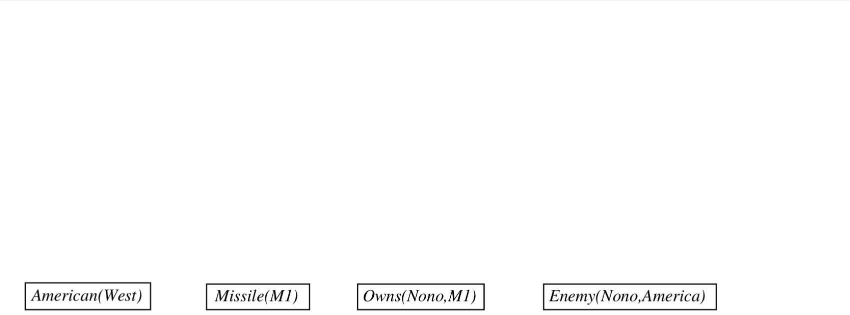
A (Very-Basic) Forward-Chaining Procerure

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**
      $(p_1 \land \ldots \land p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES($rule$)
      for each θ such that SUBST($\theta, p_1 \land \ldots \land p_n$) = SUBST($\theta, p'_1 \land \ldots \land p'_n$)
          for some $p'_1, \ldots, p'_n$ in KB
        $q' \leftarrow$ SUBST(θ, q)
        if q' does not unify with some sentence already in KB or new **then**
          add q' to new
          $\phi \leftarrow$ UNIFY(q', α)
          if ϕ is not *fail* **then return** ϕ
    add new to KB
  return *false*

Example of Forward Chaining

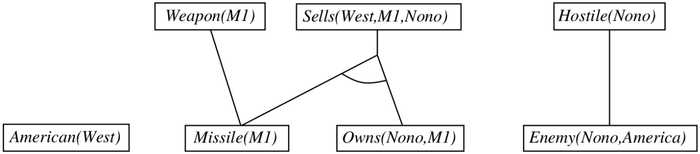
$American(West), Missile(M_1), Owns(Nono, M_1), Enemy(Nono, America)$
$\forall x.(Missile(x) \rightarrow Weapon(x))$
$\forall x.((Missile(x) \wedge Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$
$\forall x.(Enemy(x, America) \rightarrow Hostile(x))$
$\forall x, y, z.((American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x, y, z)) \rightarrow Criminal(x))$



(© S. Russell & P. Norwig, AIMA)

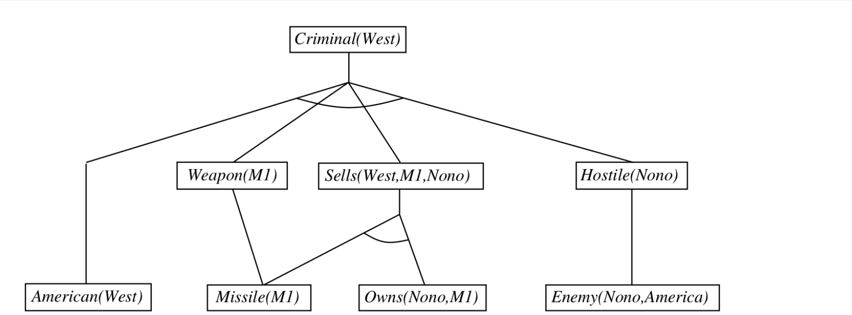
Example of Forward Chaining

$American(West), Missile(M_1), Owns(Nono, M_1), Enemy(Nono, America)$
$\forall x.(Missile(x) \rightarrow Weapon(x))$
$\forall x.((Missile(x) \wedge Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$
$\forall x.(Enemy(x, America) \rightarrow Hostile(x))$
$\forall x, y, z.((American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x, y, z)) \rightarrow Criminal(x))$



Example of Forward Chaining

$American(West), Missile(M_1), Owns(Nono, M_1), Enemy(Nono, America)$
$\forall x.(Missile(x) \rightarrow Weapon(x))$
$\forall x.((Missile(x) \wedge Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$
$\forall x.(Enemy(x, America) \rightarrow Hostile(x))$
$\forall x, y, z.((American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x, y, z)) \rightarrow Criminal(x))$



(© S. Russell & P. Norwig, AIMA)

Properties of Forward Chaining

- Sound: every inference is just an application of GMP
- Complete (for definite KBs): answers every query entailed by KB
- if $KB \models \alpha$, it always terminates
- if $KB \not\models \alpha$, may not terminate (Semi-decidable)
- Solves always Datalog queries in time: $O(p \cdot n^k)$, s.t. $p = \#predicates$, $n = \#number\ constants$, $k = manimum\ arity$
- Improvement: no need to match a rule on iteration k if a premise wasn't added on iteration k-1

  $\Longrightarrow$ match each rule whose premise contains a newly added literal
- Matching can be expensive
  - matching conjunctive premises against known facts is NP-hard
- Forward chaining is widely used in deductive databases and expert systems

Properties of Forward Chaining

- Sound: every inference is just an application of GMP
- Complete (for definite KBs): answers every query entailed by KB
- if $KB \models \alpha$, it always terminates
- if $KB \not\models \alpha$, may not terminate (Semi-decidable)
- Solves always Datalog queries in time: $O(p \cdot n^k)$, s.t. $p = \#predicates$, $n = \#number\ constants$, $k = manimum\ arity$
- Improvement: no need to match a rule on iteration k if a premise wasn't added on iteration k-1

  $\Longrightarrow$ match each rule whose premise contains a newly added literal
- Matching can be expensive
  - matching conjunctive premises against known facts is NP-hard
- Forward chaining is widely used in deductive databases and expert systems

Properties of Forward Chaining

- Sound: every inference is just an application of GMP
- Complete (for definite KBs): answers every query entailed by KB
- if $KB \models \alpha$, it always terminates
- if $KB \not\models \alpha$, may not terminate (Semi-decidable)
- Solves always Datalog queries in time: $O(p \cdot n^k)$, s.t. $p = \#predicates$, $n = \#number\ constants$, $k = manimum\ arity$
- Improvement: no need to match a rule on iteration k if a premise wasn't added on iteration k-1

  $\Longrightarrow$ match each rule whose premise contains a newly added literal
- Matching can be expensive
  - matching conjunctive premises against known facts is NP-hard
- Forward chaining is widely used in deductive databases and expert systems

Properties of Forward Chaining

- Sound: every inference is just an application of GMP
- Complete (for definite KBs): answers every query entailed by KB
- if $KB \models \alpha$, it always terminates
- if $KB \not\models \alpha$, may not terminate (Semi-decidable)
- Solves always Datalog queries in time: $O(p \cdot n^k)$, s.t. $p = \#predicates$, $n = \#number\ constants$, $k = manimum\ arity$
- Improvement: no need to match a rule on iteration k if a premise wasn't added on iteration k-1

  $\Longrightarrow$ match each rule whose premise contains a newly added literal
- Matching can be expensive
  - matching conjunctive premises against known facts is NP-hard
- Forward chaining is widely used in deductive databases and expert systems

Properties of Forward Chaining

- Sound: every inference is just an application of GMP
- Complete (for definite KBs): answers every query entailed by KB
- if $KB \models \alpha$, it always terminates
- if $KB \not\models \alpha$, may not terminate (Semi-decidable)
- Solves always Datalog queries in time: $O(p \cdot n^k)$, s.t.
  $p = \#predicates$, $n = \#number\ constants$, $k = manimum\ arity$
- Improvement: no need to match a rule on iteration k if a premise wasn't added on iteration k-1
  $\Longrightarrow$ match each rule whose premise contains a newly added literal
- Matching can be expensive
  - matching conjunctive premises against known facts is NP-hard
- Forward chaining is widely used in deductive databases and expert systems

Properties of Forward Chaining

- Sound: every inference is just an application of GMP
- Complete (for definite KBs): answers every query entailed by KB
- if $KB \models \alpha$, it always terminates
- if $KB \not\models \alpha$, may not terminate (Semi-decidable)
- Solves always Datalog queries in time: $O(p \cdot n^k)$, s.t.
  $p = \#predicates$, $n = \#number\ constants$, $k = manimum\ arity$
- Improvement: no need to match a rule on iteration k if a premise wasn't added on iteration k-1

  $\Longrightarrow$ match each rule whose premise contains a newly added literal
- Matching can be expensive
  - matching conjunctive premises against known facts is NP-hard
- Forward chaining is widely used in deductive databases and expert systems

Properties of Forward Chaining

- Sound: every inference is just an application of GMP
- Complete (for definite KBs): answers every query entailed by KB
- if $KB \models \alpha$, it always terminates
- if $KB \not\models \alpha$, may not terminate (Semi-decidable)
- Solves always Datalog queries in time: $O(p \cdot n^k)$, s.t.
  $p = \#predicates$, $n = \#number\ constants$, $k = manimum\ arity$
- Improvement: no need to match a rule on iteration k if a premise wasn't added on iteration k-1

  $\Longrightarrow$ match each rule whose premise contains a newly added literal
- Matching can be expensive
  - matching conjunctive premises against known facts is NP-hard
- Forward chaining is widely used in deductive databases and expert systems

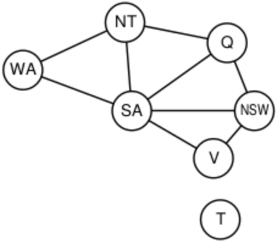
Properties of Forward Chaining

- Sound: every inference is just an application of GMP
- Complete (for definite KBs): answers every query entailed by KB
- if $KB \models \alpha$, it always terminates
- if $KB \not\models \alpha$, may not terminate (Semi-decidable)
- Solves always Datalog queries in time: $O(p \cdot n^k)$, s.t.
  $p = \#predicates$, $n = \#number\ constants$, $k = manimum\ arity$
- Improvement: no need to match a rule on iteration k if a premise wasn't added on iteration k-1

  $\Longrightarrow$ match each rule whose premise contains a newly added literal
- Matching can be expensive
  - matching conjunctive premises against known facts is NP-hard
- Forward chaining is widely used in deductive databases and expert systems

Hard Matching Example

- Colorable() is inferred iff the CSP has solution
  $\Longrightarrow$ NP-Hard



$$
\begin{aligned}
&\mathit{Diff}(wa, nt) \land \mathit{Diff}(wa, sa) \land \\
&\qquad \mathit{Diff}(nt, q)\, \mathit{Diff}(nt, sa) \land \\
&\qquad \mathit{Diff}(q, nsw) \land \mathit{Diff}(q, sa) \land \\
&\qquad \mathit{Diff}(nsw, v) \land \mathit{Diff}(nsw, sa) \land \\
&\qquad \mathit{Diff}(v, sa) \;\Rightarrow\; \mathit{Colorable}()
\end{aligned}
$$

$$
\begin{array}{ll}
\mathit{Diff}(Red, Blue) & \mathit{Diff}(Red, Green) \\
\mathit{Diff}(Green, Red) & \mathit{Diff}(Green, Blue) \\
\mathit{Diff}(Blue, Red) & \mathit{Diff}(Blue, Green)
\end{array}
$$

(© S. Russell & P. Norwig, AIMA)

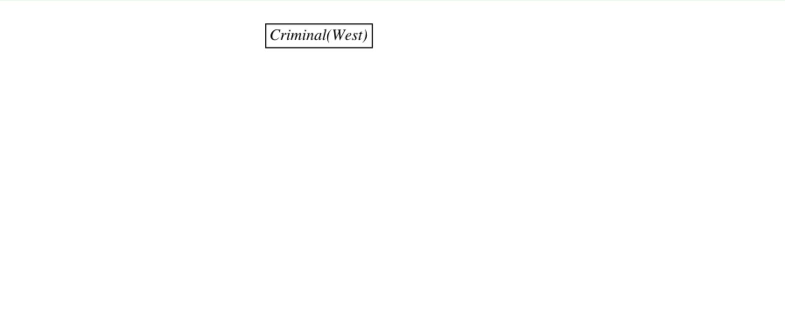
A (Very-Basic) Backward-Chaining Procerure

function FOL-BC-ASK($KB, goals, \theta$) **returns** a set of substitutions
  inputs: KB, a knowledge base
    $goals$, a list of conjuncts forming a query (θ already applied)
    θ, the current substitution, initially the empty substitution $\{ \}$
  local variables: $answers$, a set of substitutions, initially empty

  if $goals$ is empty **then return** $\{\theta\}$
  $q' \leftarrow \text{SUBST}(\theta, \text{FIRST}(goals))$
  for each sentence r **in** KB
    where STANDARDIZE-APART(r) $= (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
    and $\theta' \leftarrow \text{UNIFY}(q, q')$ succeeds
   $new\_goals \leftarrow [p_1, \ldots, p_n | \text{REST}(goals)]$
   $answers \leftarrow \text{FOL-BC-ASK}(KB, new\_goals, \text{COMPOSE}(\theta', \theta)) \cup answers$
  return $answers$

Backward Chaining: Example

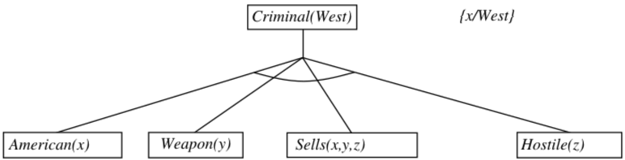
$American(West)$, $Missile(M_1)$, $Owns(Nono, M_1)$, $Enemy(Nono, America)$
$\forall x, y, z.((American(x) \land Weapon(y) \land Hostile(z) \land Sells(x, y, z)) \rightarrow Criminal(x))$
$\forall x.(Missile(x) \rightarrow Weapon(x))$
$\forall x.((Missile(x) \land Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$
$\forall x.(Enemy(x, America) \rightarrow Hostile(x))$



(© S. Russell & P. Norwig, AIMA)

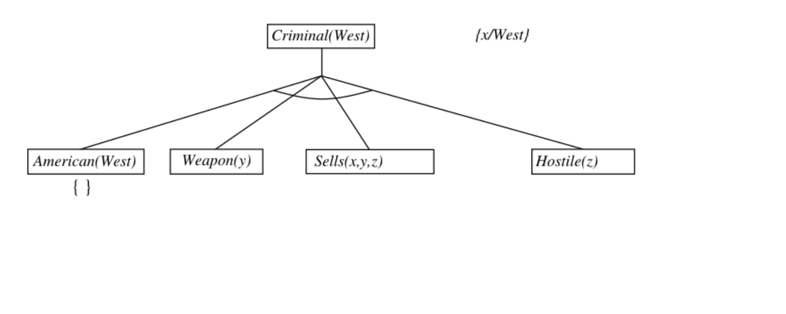
Backward Chaining: Example

$American(West)$, $Missile(M_1)$, $Owns(Nono, M_1)$, $Enemy(Nono, America)$
$\forall x, y, z.((American(x) \land Weapon(y) \land Hostile(z) \land Sells(x, y, z)) \rightarrow Criminal(x))$
$\forall x.(Missile(x) \rightarrow Weapon(x))$
$\forall x.((Missile(x) \land Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$
$\forall x.(Enemy(x, America) \rightarrow Hostile(x))$



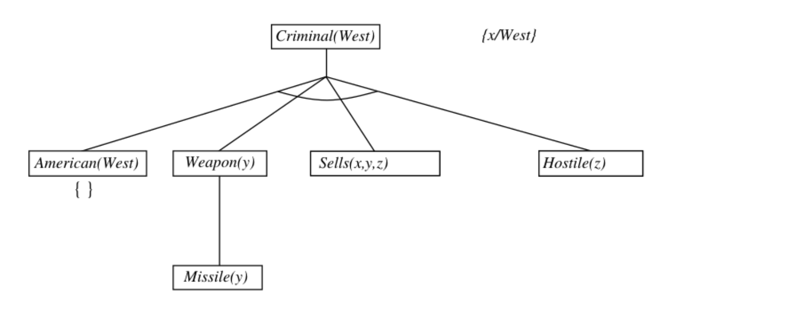
Backward Chaining: Example

$American(West)$, $Missile(M_1)$, $Owns(Nono, M_1)$, $Enemy(Nono, America)$
$\forall x, y, z.((American(x) \land Weapon(y) \land Hostile(z) \land Sells(x, y, z)) \rightarrow Criminal(x))$
$\forall x.(Missile(x) \rightarrow Weapon(x))$
$\forall x.((Missile(x) \land Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$
$\forall x.(Enemy(x, America) \rightarrow Hostile(x))$



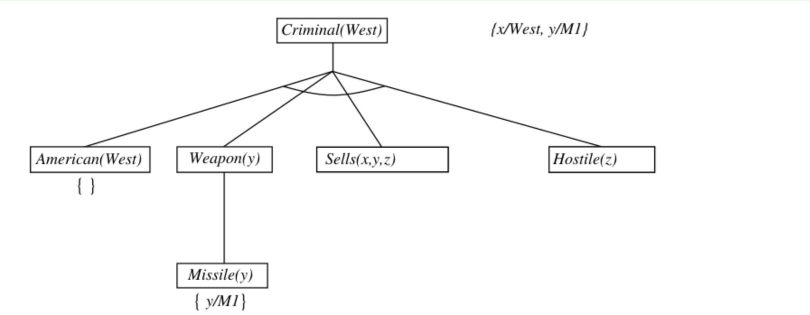
Backward Chaining: Example

$American(West)$, $Missile(M_1)$, $Owns(Nono, M_1)$, $Enemy(Nono, America)$
$\forall x, y, z.((American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x, y, z)) \rightarrow Criminal(x))$
$\forall x.(Missile(x) \rightarrow Weapon(x))$
$\forall x.((Missile(x) \wedge Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$
$\forall x.(Enemy(x, America) \rightarrow Hostile(x))$



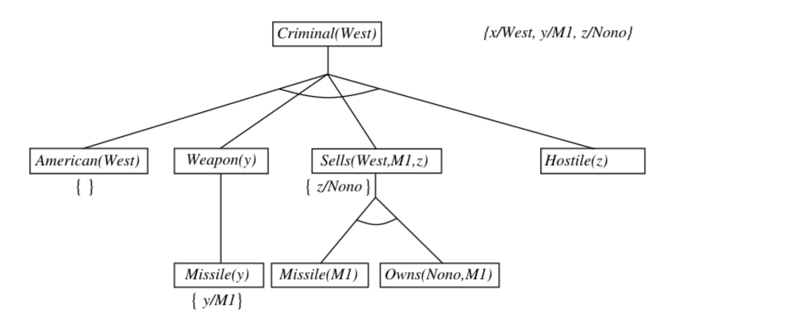
Backward Chaining: Example

American(*West*), *Missile*(M_1), *Owns*(*Nono*, M_1), *Enemy*(*Nono*, *America*)
$\forall x, y, z.((American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x, y, z)) \rightarrow Criminal(x))$
$\forall x.(Missile(x) \rightarrow Weapon(x))$
$\forall x.((Missile(x) \wedge Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$
$\forall x.(Enemy(x, America) \rightarrow Hostile(x))$



Backward Chaining: Example

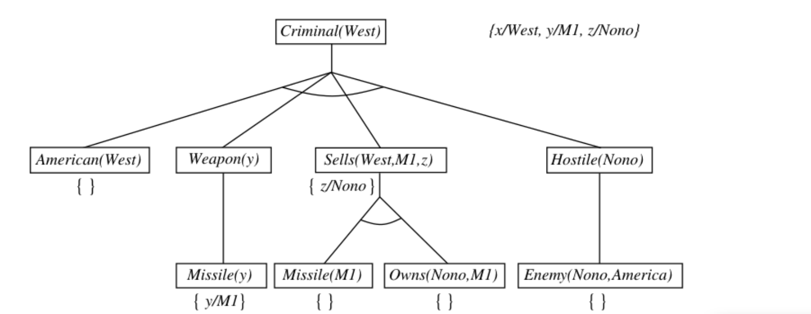
$American(West)$, $Missile(M_1)$, $Owns(Nono, M_1)$, $Enemy(Nono, America)$
$\forall x, y, z.((American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x, y, z)) \rightarrow Criminal(x))$
$\forall x.(Missile(x) \rightarrow Weapon(x))$
$\forall x.((Missile(x) \wedge Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$
$\forall x.(Enemy(x, America) \rightarrow Hostile(x))$



(© S. Russell & P. Norwig, AIMA)

Backward Chaining: Example

$American(West), Missile(M_1), Owns(Nono, M_1), Enemy(Nono, America)$
$\forall x, y, z.((American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x, y, z)) \rightarrow Criminal(x))$
$\forall x.(Missile(x) \rightarrow Weapon(x))$
$\forall x.((Missile(x) \wedge Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$
$\forall x.(Enemy(x, America) \rightarrow Hostile(x))$



(© S. Russell & P. Norwig, AIMA)

Properties of Backward Chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - e.g., $P(x) \to P(x) \implies P(c), P(c), P(c)...$ (easy to fix)
  - e.g., $Q(f(x)) \to Q(x) \implies Q(c), Q(f(c)), Q(f(f(c))), ...$
- Inefficient due to repeated subgoals
  - fix using caching of previous results $\implies$ need extra space!
- Widely used for logic programming (e.g. prolog)

Properties of Backward Chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - e.g., $P(x) \to P(x) \implies P(c), P(c), P(c)...$ (easy to fix)
  - e.g., $Q(f(x)) \to Q(x) \implies Q(c), Q(f(c)), Q(f(f(c))), \ldots$
- Inefficient due to repeated subgoals
  - fix using caching of previous results $\implies$ need extra space!
- Widely used for logic programming (e.g. prolog)

Properties of Backward Chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - e.g., $P(x) \to P(x) \implies P(c), P(c), P(c)...$ (easy to fix)
  - e.g., $Q(f(x)) \to Q(x) \implies Q(c), Q(f(c)), Q(f(f(c))), ...$
- Inefficient due to repeated subgoals
  - fix using caching of previous results $\implies$ need extra space!
- Widely used for logic programming (e.g. prolog)

Properties of Backward Chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - e.g., $P(x) \to P(x) \implies P(c), P(c), P(c)...$ (easy to fix)
  - e.g., $Q(f(x)) \to Q(x) \implies Q(c), Q(f(c)), Q(f(f(c))), ...$
- Inefficient due to repeated subgoals
  - fix using caching of previous results $\implies$ need extra space!
- Widely used for logic programming (e.g. prolog)

Properties of Backward Chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - e.g., $P(x) \rightarrow P(x) \implies P(c), P(c), P(c)...$ (easy to fix)
  - e.g., $Q(f(x)) \rightarrow Q(x) \implies Q(c), Q(f(c)), Q(f(f(c))), ...$
- Inefficient due to repeated subgoals
  - fix using caching of previous results $\implies$ need extra space!
- Widely used for logic programming (e.g. prolog)

Outline

1. Propositional vs. First-Order Inference
2. Unification and Lifting
3. Forward & Backward Chaining for Definite FOL KBs
4. Resolution for General FOL KBs

Conjunctive Normal Form (CNF)

- A FOL formula φ is in Conjunctive normal form iff it is a conjunction of disjunctions of quantifier-free literal:

$$\bigwedge_{i=1}^{L} \bigvee_{j_i=1}^{K_i} l_{j_i}$$

  - the disjunctions of literals $\bigvee_{j_i=1}^{K_i} l_{j_i}$ are called clauses
  - every literal a quantifier-free atom or its negation
  - free variables implicitly universally quantified
- Easier to handle: list of lists of literals.
  $\Longrightarrow$ no reasoning on the recursive structure of the formula
- Ex: $\neg Missile(x) \vee \neg Owns(Nono, x) \vee Sells(West, x, Nono)$

FOL CNF Conversion $CNF(\varphi)$

Every FOL formula φ can be reduced into CNF:

1. Eliminate implications and biconditionals:
   $$\alpha \rightarrow \beta \quad\Longrightarrow\quad \neg\alpha \vee \beta$$
   $$\alpha \leftrightarrow \beta \quad\Longrightarrow\quad (\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$$
2. Push inwards negations recursively:
   $$\neg(\alpha \wedge \beta) \quad\Longrightarrow\quad \neg\alpha \vee \neg\beta$$
   $$\neg(\alpha \vee \beta) \quad\Longrightarrow\quad \neg\alpha \wedge \neg\beta$$
   $$\neg\neg\alpha \quad\Longrightarrow\quad \alpha$$
   $$\neg\forall x.\alpha \quad\Longrightarrow\quad \exists x.\neg\alpha$$
   $$\neg\exists x.\alpha \quad\Longrightarrow\quad \forall x.\neg\alpha$$

$\Longrightarrow$ Negation normal form: negations only in front of atomic formulae

$\Longrightarrow$ quantifiers occur under the scope of no negation symbol

3. Standardize variables: each quantifier should use a different var
   $$(\forall x.\exists y.\alpha) \wedge \exists y.\beta \wedge \forall x.\gamma \quad\Longrightarrow\quad (\forall x.\exists y.\alpha) \wedge \exists y_1.\beta\{y/y_1\} \wedge \forall x_1.\gamma\{x/x_1\}$$

FOL CNF Conversion $CNF(\varphi)$

Every FOL formula φ can be reduced into CNF:

1. Eliminate implications and biconditionals:
$$\alpha \rightarrow \beta \quad \Longrightarrow \quad \neg\alpha \vee \beta$$
$$\alpha \leftrightarrow \beta \quad \Longrightarrow \quad (\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$$
2. Push inwards negations recursively:
$$\neg(\alpha \wedge \beta) \quad \Longrightarrow \quad \neg\alpha \vee \neg\beta$$
$$\neg(\alpha \vee \beta) \quad \Longrightarrow \quad \neg\alpha \wedge \neg\beta$$
$$\neg\neg\alpha \quad \Longrightarrow \quad \alpha$$
$$\neg\forall x.\alpha \quad \Longrightarrow \quad \exists x.\neg\alpha$$
$$\neg\exists x.\alpha \quad \Longrightarrow \quad \forall x.\neg\alpha$$

$\Longrightarrow$ Negation normal form: negations only in front of atomic formulae

$\Longrightarrow$ quantifiers occur under the scope of no negation symbol

3. Standardize variables: each quantifier should use a different var
$$(\forall x.\exists y.\alpha) \wedge \exists y.\beta \wedge \forall x.\gamma \quad \Longrightarrow \quad (\forall x.\exists y.\alpha) \wedge \exists y_1.\beta\{y/y_1\} \wedge \forall x_1.\gamma\{x/x_1\}$$

FOL CNF Conversion $CNF(\varphi)$

Every FOL formula φ can be reduced into CNF:

1. Eliminate implications and biconditionals:
$$\alpha \rightarrow \beta \quad \Longrightarrow \quad \neg\alpha \vee \beta$$
$$\alpha \leftrightarrow \beta \quad \Longrightarrow \quad (\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$$

2. Push inwards negations recursively:
$$\neg(\alpha \wedge \beta) \quad \Longrightarrow \quad \neg\alpha \vee \neg\beta$$
$$\neg(\alpha \vee \beta) \quad \Longrightarrow \quad \neg\alpha \wedge \neg\beta$$
$$\neg\neg\alpha \quad \Longrightarrow \quad \alpha$$
$$\neg\forall x.\alpha \quad \Longrightarrow \quad \exists x.\neg\alpha$$
$$\neg\exists x.\alpha \quad \Longrightarrow \quad \forall x.\neg\alpha$$

$\Longrightarrow$ Negation normal form: negations only in front of atomic formulae

$\Longrightarrow$ quantifiers occur under the scope of no negation symbol

3. Standardize variables: each quantifier should use a different var
$$(\forall x.\exists y.\alpha) \wedge \exists y.\beta \wedge \forall x.\gamma \quad \Longrightarrow \quad (\forall x.\exists y.\alpha) \wedge \exists y_1.\beta\{y/y_1\} \wedge \forall x_1.\gamma\{x/x_1\}$$

FOL CNF Conversion $CNF(\varphi)$ [cont.]

4. Skolemize (a generalization of EI):
   Each existential variable is replaced by a fresh Skolem function applied to the enclosing universally-quantified variables

$$\begin{array}{lll} \exists y.\alpha & \Longrightarrow & \alpha\{y/c\} \\ \forall x.(...\exists y.\alpha...) & \Longrightarrow & \forall x.(...\alpha\{y/F_1(x)\}...) \\ \forall x_1 x_2.(...\exists y.\alpha...) & \Longrightarrow & \forall x_1 x_2.(...\alpha\{y/F_1(x_1,x_2)...)\} \\ \exists y_1 \forall x_1 x_2 \exists y_2 \forall x_3 \exists y_3.\alpha & \Longrightarrow & \forall x_1 x_2 x_3. \\ & & \quad \alpha\{y_1/c, y_2/F_1(x_1,x_2), y_3/F_2(x_1,x_2,x_3)\} \end{array}$$

   Ex: $\forall x \exists y.Father(x,y) \Longrightarrow \forall x.Father(x,s(x))$
   ($s(x)$ implictly means "son of x" although s() is a fresh function)

FOL CNF Conversion $CNF(\varphi)$ [cont.]

4. Skolemize (a generalization of EI):
   Each existential variable is replaced by a fresh Skolem function applied to the enclosing universally-quantified variables

$$
\begin{array}{lcl}
\exists y.\alpha & \Longrightarrow & \alpha\{y/c\} \\
\forall x.(...\exists y.\alpha...) & \Longrightarrow & \forall x.(...\alpha\{y/F_1(x)\}...) \\
\forall x_1 x_2.(...\exists y.\alpha...) & \Longrightarrow & \forall x_1 x_2.(...\alpha\{y/F_1(x_1, x_2)...)\} \\
\exists y_1 \forall x_1 x_2 \exists y_2 \forall x_3 \exists y_3.\alpha & \Longrightarrow & \forall x_1 x_2 x_3. \\
 & & \quad \alpha\{y_1/c, y_2/F_1(x_1, x_2), y_3/F_2(x_1, x_2, x_3)\}
\end{array}
$$

   Ex: $\forall x \exists y.Father(x, y) \Longrightarrow \forall x.Father(x, s(x))$
   ($s(x)$ implictly means "son of x" although s() is a fresh function)

FOL CNF Conversion $CNF(\varphi)$ [cont.]

5. Drop universal quantifiers:
   $\forall x_1 \ldots x_k . \alpha \quad \Longrightarrow \quad \alpha$
   $\Longrightarrow$ free variables implicitly universally quantified
6. CNF-ize propositionally (see Ch.08):
   either apply recursively the DeMorgan's Rule:
   $(\alpha \wedge \beta) \vee \gamma \quad \Longrightarrow \quad (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$
   or rename subformulas and add definitions:
   $(\alpha \wedge \beta) \vee \gamma \quad \Longrightarrow \quad (B \vee \gamma) \wedge CNF(B \leftrightarrow (\alpha \wedge \beta))$

Preserves satisfiability: $M(\varphi) \neq \emptyset$ iff $M(CNF(\varphi)) \neq \emptyset$

FOL CNF Conversion $CNF(\varphi)$ [cont.]

5. Drop universal quantifiers:
   $\forall x_1 \ldots x_k . \alpha \quad \Longrightarrow \quad \alpha$
   $\Longrightarrow$ free variables implicitly universally quantified
6. CNF-ize propositionally (see Ch.08):
   either apply recursively the DeMorgan's Rule:
   $(\alpha \wedge \beta) \vee \gamma \quad \Longrightarrow \quad (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$
   or rename subformulas and add definitions:
   $(\alpha \wedge \beta) \vee \gamma \quad \Longrightarrow \quad (B \vee \gamma) \wedge CNF(B \leftrightarrow (\alpha \wedge \beta))$

Preserves satisfiability: $M(\varphi) \neq \emptyset$ iff $M(CNF(\varphi)) \neq \emptyset$

FOL CNF Conversion $CNF(\varphi)$ [cont.]

5. Drop universal quantifiers:
   $\forall x_1 \ldots x_k . \alpha \quad \Longrightarrow \quad \alpha$
   $\Longrightarrow$ free variables implicitly universally quantified
6. CNF-ize propositionally (see Ch.08):
   either apply recursively the DeMorgan's Rule:
   $(\alpha \wedge \beta) \vee \gamma \quad \Longrightarrow \quad (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$
   or rename subformulas and add definitions:
   $(\alpha \wedge \beta) \vee \gamma \quad \Longrightarrow \quad (B \vee \gamma) \wedge CNF(B \leftrightarrow (\alpha \wedge \beta))$

Preserves satisfiability: $M(\varphi) \neq \emptyset$ iff $M(CNF(\varphi)) \neq \emptyset$

FOL CNF Conversion $CNF(\varphi)$ [cont.]

5. Drop universal quantifiers:
   $\forall x_1 ... x_k . \alpha \quad \Longrightarrow \quad \alpha$
   $\Longrightarrow$ free variables implicitly universally quantified
6. CNF-ize propositionally (see Ch.08):
   either apply recursively the DeMorgan's Rule:
   $(\alpha \wedge \beta) \vee \gamma \quad \Longrightarrow \quad (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$
   or rename subformulas and add definitions:
   $(\alpha \wedge \beta) \vee \gamma \quad \Longrightarrow \quad (B \vee \gamma) \wedge CNF(B \leftrightarrow (\alpha \wedge \beta))$

Preserves satisfiability: $M(\varphi) \neq \emptyset$ iff $M(CNF(\varphi)) \neq \emptyset$

Conversion to CNF: Example

Consider: "Everyone who loves all animals is loved by someone"

$\forall x.([\forall y.(Animal(y) \rightarrow Loves(x, y))] \rightarrow [\exists y.Loves(y, x)])$

1. Eliminate implications and biconditionals:

   $\forall x.(\neg[\forall y.(\neg Animal(y) \vee Loves(x, y))] \vee [\exists y.Loves(y, x)])$

2. Push inwards negations recursively:

   $\forall x.([\exists y.\neg(\neg Animal(y) \vee Loves(x, y))] \vee [\exists y.Loves(y, x)])$
   $\forall x.([\exists y.(\neg\neg Animal(y) \wedge \neg Loves(x, y))] \vee [\exists y.Loves(y, x)])$
   $\forall x.([\exists y.(Animal(y) \wedge \neg Loves(x, y))] \vee [\exists y.Loves(y, x)])$

3. Standardize variables:

   $\forall x.([\exists y.(Animal(y) \wedge \neg Loves(x, y))] \vee [\exists z.Loves(z, x)])$

4. Skolemize:

   $\forall x.([Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee [Loves(G(x), x)])$
   (F(x): "an animal unloved by x"; G(x): "someone who loves x")

5. Drop universal quantifiers::

   $[Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee [Loves(G(x), x)]$

6. CNF-ize propositionally:

$(Animal(F(x)) \vee Loves(G(x), x)) \wedge (\neg Loves(x, F(x)) \vee Loves(G(x), x))$

Conversion to CNF: Example

Consider: "Everyone who loves all animals is loved by someone"

$\forall x.([\forall y.(Animal(y) \rightarrow Loves(x,y))] \rightarrow [\exists y.Loves(y,x)])$

1. Eliminate implications and biconditionals:

   $\forall x.(\neg[\forall y.(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y.Loves(y,x)])$

2. Push inwards negations recursively:

   $\forall x.([\exists y.\neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y.Loves(y,x)])$
   $\forall x.([\exists y.(\neg\neg Animal(y) \wedge \neg Loves(x,y))] \vee [\exists y.Loves(y,x)])$
   $\forall x.([\exists y.(Animal(y) \wedge \neg Loves(x,y))] \vee [\exists y.Loves(y,x)])$

3. Standardize variables:

   $\forall x.([\exists y.(Animal(y) \wedge \neg Loves(x,y))] \vee [\exists z.Loves(z,x)])$

4. Skolemize:

   $\forall x.([Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee [Loves(G(x),x)])$
   (F(x): "an animal unloved by x"; G(x): "someone who loves x")

5. Drop universal quantifiers::

   $[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee [Loves(G(x),x)]$

6. CNF-ize propositionally:

$(Animal(F(x)) \vee Loves(G(x),x)) \wedge (\neg Loves(x,F(x)) \vee Loves(G(x),x))$

Conversion to CNF: Example

Consider: "Everyone who loves all animals is loved by someone"

$\forall x.([\forall y.(Animal(y) \rightarrow Loves(x, y))] \rightarrow [\exists y.Loves(y, x)])$

1. Eliminate implications and biconditionals:

   $\forall x.(\neg[\forall y.(\neg Animal(y) \vee Loves(x, y))] \vee [\exists y.Loves(y, x)])$

2. Push inwards negations recursively:

   $\forall x.([\exists y.\neg(\neg Animal(y) \vee Loves(x, y))] \vee [\exists y.Loves(y, x)])$
   $\forall x.([\exists y.(\neg\neg Animal(y) \wedge \neg Loves(x, y))] \vee [\exists y.Loves(y, x)])$
   $\forall x.([\exists y.(Animal(y) \wedge \neg Loves(x, y))] \vee [\exists y.Loves(y, x)])$

3. Standardize variables:

   $\forall x.([\exists y.(Animal(y) \wedge \neg Loves(x, y))] \vee [\exists z.Loves(z, x)])$

4. Skolemize:

   $\forall x.([Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee [Loves(G(x), x)])$
   (F(x): "an animal unloved by x"; G(x): "someone who loves x")

5. Drop universal quantifiers::

   $[Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee [Loves(G(x), x)]$

6. CNF-ize propositionally:

$(Animal(F(x)) \vee Loves(G(x), x)) \wedge (\neg Loves(x, F(x)) \vee Loves(G(x), x))$

Conversion to CNF: Example

Consider: "Everyone who loves all animals is loved by someone"

$\forall x.([\forall y.(Animal(y) \rightarrow Loves(x, y))] \rightarrow [\exists y.Loves(y, x)])$

1. Eliminate implications and biconditionals:

   $\forall x.(\neg[\forall y.(\neg Animal(y) \vee Loves(x, y))] \vee [\exists y.Loves(y, x)])$

2. Push inwards negations recursively:

   $\forall x.([\exists y.\neg(\neg Animal(y) \vee Loves(x, y))] \vee [\exists y.Loves(y, x)])$
   $\forall x.([\exists y.(\neg\neg Animal(y) \wedge \neg Loves(x, y))] \vee [\exists y.Loves(y, x)])$
   $\forall x.([\exists y.(Animal(y) \wedge \neg Loves(x, y))] \vee [\exists y.Loves(y, x)])$

3. Standardize variables:

   $\forall x.([\exists y.(Animal(y) \wedge \neg Loves(x, y))] \vee [\exists z.Loves(z, x)])$

4. Skolemize:

   $\forall x.([Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee [Loves(G(x), x)])$
   (F(x): "an animal unloved by x"; G(x): "someone who loves x")

5. Drop universal quantifiers::

   $[Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee [Loves(G(x), x)]$

6. CNF-ize propositionally:

$(Animal(F(x)) \vee Loves(G(x), x)) \wedge (\neg Loves(x, F(x)) \vee Loves(G(x), x))$

Conversion to CNF: Example

Consider: "Everyone who loves all animals is loved by someone"

$\forall x.([\forall y.(Animal(y) \rightarrow Loves(x,y))] \rightarrow [\exists y.Loves(y,x)])$

1. Eliminate implications and biconditionals:

   $\forall x.(\neg[\forall y.(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y.Loves(y,x)])$

2. Push inwards negations recursively:

   $\forall x.([\exists y.\neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y.Loves(y,x)])$
   $\forall x.([\exists y.(\neg\neg Animal(y) \wedge \neg Loves(x,y))] \vee [\exists y.Loves(y,x)])$
   $\forall x.([\exists y.(Animal(y) \wedge \neg Loves(x,y))] \vee [\exists y.Loves(y,x)])$

3. Standardize variables:

   $\forall x.([\exists y.(Animal(y) \wedge \neg Loves(x,y))] \vee [\exists z.Loves(z,x)])$

4. Skolemize:

   $\forall x.([Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee [Loves(G(x),x)])$
   (F(x): "an animal unloved by x"; G(x): "someone who loves x")

5. Drop universal quantifiers::

   $[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee [Loves(G(x),x)]$

6. CNF-ize propositionally:

   $(Animal(F(x)) \vee Loves(G(x),x)) \wedge (\neg Loves(x,F(x)) \vee Loves(G(x),x))$

Conversion to CNF: Example

Consider: "Everyone who loves all animals is loved by someone"

$\forall x.([\forall y.(Animal(y) \to Loves(x,y))] \to [\exists y.Loves(y,x)])$

1. Eliminate implications and biconditionals:

   $\forall x.(\neg[\forall y.(\neg Animal(y) \lor Loves(x,y))] \lor [\exists y.Loves(y,x)])$

2. Push inwards negations recursively:

   $\forall x.([\exists y.\neg(\neg Animal(y) \lor Loves(x,y))] \lor [\exists y.Loves(y,x)])$

   $\forall x.([\exists y.(\neg\neg Animal(y) \land \neg Loves(x,y))] \lor [\exists y.Loves(y,x)])$

   $\forall x.([\exists y.(Animal(y) \land \neg Loves(x,y))] \lor [\exists y.Loves(y,x)])$

3. Standardize variables:

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

4. Skolemize:

   $\forall x.([Animal(F(x)) \land \neg Loves(x,F(x))] \lor [Loves(G(x),x)])$

   (F(x): "an animal unloved by x"; G(x): "someone who loves x")

5. Drop universal quantifiers::

   $[Animal(F(x)) \land \neg Loves(x,F(x))] \lor [Loves(G(x),x)]$

6. CNF-ize propositionally:

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

Conversion to CNF: Example

Consider: "Everyone who loves all animals is loved by someone"

$\forall x.([\forall y.(Animal(y) \rightarrow Loves(x,y))] \rightarrow [\exists y.Loves(y,x)])$

1. Eliminate implications and biconditionals:

   $$\forall x.(\neg[\forall y.(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y.Loves(y,x)])$$

2. Push inwards negations recursively:

   $$\forall x.([\exists y.\neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y.Loves(y,x)])$$
   $$\forall x.([\exists y.(\neg\neg Animal(y) \wedge \neg Loves(x,y))] \vee [\exists y.Loves(y,x)])$$
   $$\forall x.([\exists y.(Animal(y) \wedge \neg Loves(x,y))] \vee [\exists y.Loves(y,x)])$$

3. Standardize variables:

   $$\forall x.([\exists y.(Animal(y) \wedge \neg Loves(x,y))] \vee [\exists z.Loves(z,x)])$$

4. Skolemize:

   $$\forall x.([Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee [Loves(G(x),x)])$$
   (F(x): "an animal unloved by x"; G(x): "someone who loves x")

5. Drop universal quantifiers::

   $$[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee [Loves(G(x),x)]$$

6. CNF-ize propositionally:

$$(Animal(F(x)) \vee Loves(G(x),x)) \wedge (\neg Loves(x,F(x)) \vee Loves(G(x),x))$$

Resolution

- FOL resolution rule, let $\theta \stackrel{\text{def}}{=} Unify(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

$$\frac{(l_1 \vee ... \vee l_k) \qquad (m_1 \vee ... \vee m_n)}{(l_1 \vee ... \vee l_{i-1} \vee l_{i+1} \vee ... \vee l_k \vee m_1 \vee ... \vee m_{j-1} \vee m_{j+1} \vee ... \vee m_n)\theta}$$

- Ex: $\frac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $KB \models \alpha$ in FOL:
  - convert $KB \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(KB \wedge \neg\alpha)$ until
    - the empty clause is generate $\Longrightarrow KB \models \alpha$
    - no more resolution step is applicable $\Longrightarrow KB \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
    - unit resolution alone complete for definite clauses
- Complete:
  - If there is a substitution θ such that $KB \models \theta\alpha$, then it will return θ
  - If there is no such θ, then the procedure may not terminate
- Many strategies and tools available

Resolution

- FOL resolution rule, let $\theta \stackrel{\text{def}}{=} Unify(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

$$\frac{(l_1 \vee \ldots \vee l_k) \qquad (m_1 \vee \ldots \vee m_n)}{(l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)\theta}$$

- Ex: $\frac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $KB \models \alpha$ in FOL:
  - convert $KB \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(KB \wedge \neg\alpha)$ until
    - the empty clause is generate $\Longrightarrow KB \models \alpha$
    - no more resolution step is applicable $\Longrightarrow KB \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
    - unit resolution alone complete for definite clauses
- Complete:
  - If there is a substitution θ such that $KB \models \theta\alpha$, then it will return θ
  - If there is no such θ, then the procedure may not terminate
- Many strategies and tools available

Resolution

- FOL resolution rule, let $\theta \stackrel{\text{def}}{=} Unify(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

$$\frac{(l_1 \vee ... \vee l_k) \qquad (m_1 \vee ... \vee m_n)}{(l_1 \vee ... \vee l_{i-1} \vee l_{i+1} \vee ... \vee l_k \vee m_1 \vee ... \vee m_{j-1} \vee m_{j+1} \vee ... \vee m_n)\theta}$$

- Ex: $\frac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $KB \models \alpha$ in FOL:
  - convert $KB \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(KB \wedge \neg\alpha)$ until
    - the empty clause is generate $\Longrightarrow KB \models \alpha$
    - no more resolution step is applicable $\Longrightarrow KB \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
    - unit resolution alone complete for definite clauses
- Complete:
  - If there is a substitution θ such that $KB \models \theta\alpha$, then it will return θ
  - If there is no such θ, then the procedure may not terminate
- Many strategies and tools available

Resolution

- FOL resolution rule, let $\theta \stackrel{\text{def}}{=} Unify(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

$$\frac{(l_1 \vee ... \vee l_k) \qquad (m_1 \vee ... \vee m_n)}{(l_1 \vee ... \vee l_{i-1} \vee l_{i+1} \vee ... \vee l_k \vee m_1 \vee ... \vee m_{j-1} \vee m_{j+1} \vee ... \vee m_n)\theta}$$

- Ex: $\frac{Man(Socrates) \qquad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $KB \models \alpha$ in FOL:
  - convert $KB \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(KB \wedge \neg\alpha)$ until
    - the empty clause is generate $\Longrightarrow KB \models \alpha$
    - no more resolution step is applicable $\Longrightarrow KB \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
    - unit resolution alone complete for definite clauses
- Complete:
  - If there is a substitution θ such that $KB \models \theta\alpha$, then it will return θ
  - If there is no such θ, then the procedure may not terminate
- Many strategies and tools available

Resolution

- **FOL resolution rule**, let $\theta \stackrel{\text{def}}{=} Unify(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

$$\frac{(l_1 \vee ... \vee l_k) \qquad (m_1 \vee ... \vee m_n)}{(l_1 \vee ... \vee l_{i-1} \vee l_{i+1} \vee ... \vee l_k \vee m_1 \vee ... \vee m_{j-1} \vee m_{j+1} \vee ... \vee m_n)\theta}$$

- Ex: $\frac{Man(Socrates) \qquad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $KB \models \alpha$ in FOL:
  - convert $KB \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(KB \wedge \neg\alpha)$ until
    - the empty clause is generate $\Longrightarrow KB \models \alpha$
    - no more resolution step is applicable $\Longrightarrow KB \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
    - unit resolution alone complete for definite clauses
- Complete:
  - If there is a substitution θ such that $KB \models \theta\alpha$, then it will return θ
  - If there is no such θ, then the procedure may not terminate
- Many strategies and tools available

Resolution

- FOL resolution rule, let $\theta \stackrel{\text{def}}{=} Unify(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

$$\frac{(l_1 \vee ... \vee l_k) \qquad (m_1 \vee ... \vee m_n)}{(l_1 \vee ... \vee l_{i-1} \vee l_{i+1} \vee ... \vee l_k \vee m_1 \vee ... \vee m_{j-1} \vee m_{j+1} \vee ... \vee m_n)\theta}$$

- Ex: $\frac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $KB \models \alpha$ in FOL:
  - convert $KB \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(KB \wedge \neg\alpha)$ until
    - the empty clause is generate $\Longrightarrow KB \models \alpha$
    - no more resolution step is applicable $\Longrightarrow KB \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
    - unit resolution alone complete for definite clauses
- Complete:
  - If there is a substitution θ such that $KB \models \theta\alpha$, then it will return θ
  - If there is no such θ, then the procedure may not terminate
- Many strategies and tools available

Resolution

- FOL resolution rule, let $\theta \stackrel{\text{def}}{=} Unify(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

$$\frac{(l_1 \vee ... \vee l_k) \qquad (m_1 \vee ... \vee m_n)}{(l_1 \vee ... \vee l_{i-1} \vee l_{i+1} \vee ... \vee l_k \vee m_1 \vee ... \vee m_{j-1} \vee m_{j+1} \vee ... \vee m_n)\theta}$$

- Ex: $\frac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $KB \models \alpha$ in FOL:
  - convert $KB \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(KB \wedge \neg\alpha)$ until
    - the empty clause is generate $\Longrightarrow KB \models \alpha$
    - no more resolution step is applicable $\Longrightarrow KB \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
    - unit resolution alone complete for definite clauses
- Complete:
  - If there is a substitution θ such that $KB \models \theta\alpha$, then it will return θ
  - If there is no such θ, then the procedure may not terminate
- Many strategies and tools available

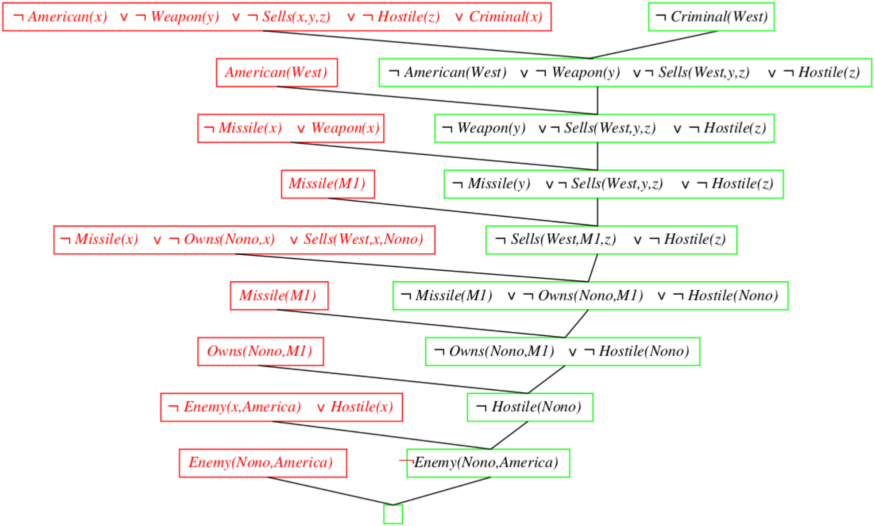
Resolution

- FOL resolution rule, let $\theta \stackrel{\text{def}}{=} Unify(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

$$\frac{(l_1 \vee ... \vee l_k) \qquad (m_1 \vee ... \vee m_n)}{(l_1 \vee ... \vee l_{i-1} \vee l_{i+1} \vee ... \vee l_k \vee m_1 \vee ... \vee m_{j-1} \vee m_{j+1} \vee ... \vee m_n)\theta}$$

- Ex: $\frac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $KB \models \alpha$ in FOL:
  - convert $KB \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(KB \wedge \neg\alpha)$ until
    - the empty clause is generate $\Longrightarrow KB \models \alpha$
    - no more resolution step is applicable $\Longrightarrow KB \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
    - unit resolution alone complete for definite clauses
- Complete:
  - If there is a substitution θ such that $KB \models \theta\alpha$, then it will return θ
  - If there is no such θ, then the procedure may not terminate
- Many strategies and tools available

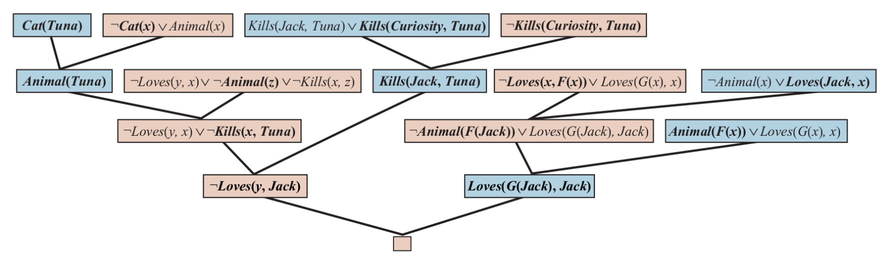
Example: Resolution with Definite Clauses



(© S. Russell & P. Norwig, AIMA)

Example: Resolution with General Clauses

Everyone who loves all animals is loved by someone.
Anyone who kills an animal is loved by no one.
Jack loves all animals.
Either Jack or Curiosity killed the cat, who is named Tuna.
Did Curiosity kill the cat?
(See AIMA book for FOL formalization and CNF-ization, or do it by exercise)



Dealing with Term Equalities [hints.]

To deal with equality formulas ($t_1 = t_2$)

- Combine resolution with Equal-term substitution rule
- Ex:

$$\frac{(4 \geq 3) \quad \dfrac{(S(x) = x+1) \quad (\neg(y \geq z) \vee (S(y) \geq S(z)))}{(\neg(y \geq z) \vee (y+1 \geq z+1))}}{4+1 \geq 3+1}$$

- Ad-hoc rules rule for equality: Demodulation and Paramodulation (not covered here)

Dealing with Term Equalities [hints.]

To deal with equality formulas $(t_1 = t_2)$

- Combine resolution with Equal-term substitution rule
- Ex:

$$\frac{(4 \geq 3) \quad \dfrac{(S(x)=x+1) \quad (\neg(y \geq z) \vee (S(y) \geq S(z)))}{(\neg(y \geq z) \vee (y+1 \geq z+1))}}{4+1 \geq 3+1}$$

- Ad-hoc rules rule for equality: Demodulation and Paramodulation (not covered here)

Dealing with Term Equalities [hints.]

To deal with equality formulas $(t_1 = t_2)$

- Combine resolution with Equal-term substitution rule
- Ex:

$$\frac{(4 \geq 3) \quad \dfrac{(S(x)=x+1) \quad (\neg(y \geq z) \vee (S(y) \geq S(z)))}{(\neg(y \geq z) \vee (y+1 \geq z+1))}}{4+1 \geq 3+1}$$

- Ad-hoc rules rule for equality: Demodulation and Paramodulation (not covered here)

Dealing with Term Equalities [hints.]

To deal with equality formulas ($t_1 = t_2$)

- Combine resolution with Equal-term substitution rule
- Ex:

$$
(4 \geq 3) \frac{\dfrac{(S(x) = x+1) \quad (\neg(y \geq z) \vee (S(y) \geq S(z)))}{(\neg(y \geq z) \vee (y + 1 \geq z + 1))}}{4 + 1 \geq 3 + 1}
$$

- Ad-hoc rules rule for equality: Demodulation and Paramodulation (not covered here)

Exercise

Consider the FOL formalization of the Wumpus World in Ch. 08

1. CNF-ize the KB
2. Perform the inference steps via resolution & equal-term substitution