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

Roberto Sebastiani

DISI, Università di Trento, Italy – roberto.sebastiani@unitn.it

https://disi.unitn.it/rseba/DIDATTICA/fai_2023/

Teaching assistants:

Mauro Dragoni, dragoni@fbk.eu, https://www.maurodragoni.com/teaching/fai/

Paolo Morettin, paolo.morettin@unitn.it, https://paolomorettin.github.io/

M.S. Course "Artificial Intelligence Systems", academic year 2023-2024

Last update: Monday 13th November, 2023, 09:03

Outline

1. Basic First-Order Reasoning
   - Substitutions & Instantiations
   - From Propositional to First-Order Reasoning
   - Unification and Lifting
2. Handling Definite FOL KBs & Datalog
   - Forward Chaining (hints)
   - Backward Chaining (hints)
3. Resolution for General FOL KBs
   - CNF-Ization
   - Resolution
   - A Complete Example

Outline

Outline

Term/Subformula Substitutions

Notation

- Substitution: "Subst($\{e_1/e_2\}, e$)" or "$e\{e_1/e_2\}$":
  the expression obtained by simultaneously 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.f. sub.): $(Even(x) \vee Odd(x))\{Even(x)/Odd(S(x))\} \Longrightarrow ((Odd(S(x)) \vee Odd(x))$
- Multiple substitution: apply simulteneously all substitutions in a list: $e\{e_1/e_2, e_3/e_4\}$
  - ex: $(P(x, y) \rightarrow Q(x, y))\{x/1, y/2\} \Longrightarrow (P(1, 2) \rightarrow Q(1, 2))$
  - multiple substitutions are simultaneous:
    ex: $P(x) \vee Q(y)\{x/y, y/f(b)\} = P(y) \vee Q(f(b)$ (not $P(f(b)) \vee Q(f(b))$)
- If θ is a substitution list and e an expression (formula/term),
  then we denote the result of a substitution as $e\theta$

Term/Subformula Substitutions

Notation

- Substitution: "Subst($\{e_1/e_2\}, e$)" or "$e\{e_1/e_2\}$":
  the expression obtained by simultaneously 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.f. sub.): $(Even(x) \vee Odd(x))\{Even(x)/Odd(S(x))\} \Longrightarrow ((Odd(S(x)) \vee Odd(x))$
- Multiple substitution: apply simulteneously all substitutions in a list: $e\{e_1/e_2, e_3/e_4\}$
  - ex: $(P(x, y) \rightarrow Q(x, y))\{x/1, y/2\} \Longrightarrow (P(1, 2) \rightarrow Q(1, 2))$
  - multiple substitutions are simultaneous:
    ex: $P(x) \vee Q(y)\{x/y, y/f(b)\} = P(y) \vee Q(f(b)$ (not $P(f(b)) \vee Q(f(b))$)
- If θ is a substitution list and e an expression (formula/term),
  then we denote the result of a substitution as $e\theta$

Term/Subformula Substitutions

Notation

- Substitution: "Subst($\{e_1/e_2\}, e$)" or "$e\{e_1/e_2\}$":
  the expression obtained by simultaneously 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.f. sub.): $(Even(x) \vee Odd(x))\{Even(x)/Odd(S(x))\} \Longrightarrow ((Odd(S(x)) \vee Odd(x))$
- Multiple substitution: apply simulteneously all substitutions in a list: $e\{e_1/e_2, e_3/e_4\}$
  - ex: $(P(x, y) \rightarrow Q(x, y))\{x/1, y/2\} \Longrightarrow (P(1, 2) \rightarrow Q(1, 2))$
  - multiple substitutions are simultaneous:
    ex: $P(x) \vee Q(y)\{x/y, y/f(b)\} = P(y) \vee Q(f(b)$ (not $P(f(b)) \vee Q(f(b))$)
- If θ is a substitution list and e an expression (formula/term),
  then we denote the result of a substitution as $e\theta$

Term/Subformula Substitutions

Notation

- Substitution: "Subst($\{e_1/e_2\}, e$)" or "$e\{e_1/e_2\}$":
  the expression obtained by simultaneously 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.f. sub.): $(Even(x) \vee Odd(x))\{Even(x)/Odd(S(x))\} \Longrightarrow ((Odd(S(x)) \vee Odd(x))$
- Multiple substitution: apply simulteneously all substitutions in a list: $e\{e_1/e_2, e_3/e_4\}$
  - ex: $(P(x, y) \rightarrow Q(x, y))\{x/1, y/2\} \Longrightarrow (P(1, 2) \rightarrow Q(1, 2))$
  - multiple substitutions are simultaneous:
    ex: $P(x) \vee Q(y)\{x/y, y/f(b)\} = P(y) \vee Q(f(b)$ (not $P(f(b)) \vee Q(f(b))$)
- If θ is a substitution list and e an expression (formula/term),
  then we denote the result of a substitution as $e\theta$

Substitution with equivalent terms

Equal-term substitution rule

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

- Ex: $(S(x) = x + 1) \wedge (0 \neq S(x)) \Longrightarrow (S(x) = x + 1) \wedge (0 \neq S(x)) \wedge (0 \neq x + 1)$
- Preserves validity: $M(\Gamma \wedge (t_1 = t_2) \wedge \alpha \wedge \alpha\{t_1/t_2\}) = M(\Gamma \wedge (t_1 = t_2) \wedge \alpha)$
- α can be safely dropped from the result

Substitution with equivalent terms

Equal-term substitution rule

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

- Ex: $(S(x) = x + 1) \wedge (0 \neq S(x)) \Longrightarrow (S(x) = x + 1) \wedge (0 \neq S(x)) \wedge (0 \neq x + 1)$
- Preserves validity: $M(\Gamma \wedge (t_1 = t_2) \wedge \alpha \wedge \alpha\{t_1/t_2\}) = M(\Gamma \wedge (t_1 = t_2) \wedge \alpha)$
- α can be safely dropped from the result

Substitution with equivalent terms

Equal-term substitution rule

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

- Ex: $(S(x) = x + 1) \wedge (0 \neq S(x)) \Longrightarrow (S(x) = x + 1) \wedge (0 \neq S(x)) \wedge (0 \neq x + 1)$
- Preserves validity: $M(\Gamma \wedge (t_1 = t_2) \wedge \alpha \wedge \alpha\{t_1/t_2\}) = M(\Gamma \wedge (t_1 = t_2) \wedge \alpha)$
- α can be safely dropped from the result

Substitution with equivalent terms

Equal-term substitution rule

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

- Ex: $(S(x) = x + 1) \wedge (0 \neq S(x)) \Longrightarrow (S(x) = x + 1) \wedge (0 \neq S(x)) \wedge (0 \neq x + 1)$
- Preserves validity: $M(\Gamma \wedge (t_1 = t_2) \wedge \alpha \wedge \alpha\{t_1/t_2\}) = M(\Gamma \wedge (t_1 = t_2) \wedge \alpha)$
- α can be safely dropped from the result

Substitution with equivalent formulas

Equivalent-subformula substitution rule

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

- 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: $M(\Gamma \wedge (\beta_1 = \beta_2) \wedge \alpha \wedge \alpha\{\beta_1/\beta_2\}) = M(\Gamma \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha)$
- α can be safely dropped from the result

Substitution with equivalent formulas

Equivalent-subformula substitution rule

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

- 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: $M(\Gamma \wedge (\beta_1 = \beta_2) \wedge \alpha \wedge \alpha\{\beta_1/\beta_2\}) = M(\Gamma \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha)$
- α can be safely dropped from the result

Substitution with equivalent formulas

Equivalent-subformula substitution rule

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

- 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: $M(\Gamma \wedge (\beta_1 = \beta_2) \wedge \alpha \wedge \alpha\{\beta_1/\beta_2\}) = M(\Gamma \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha)$
- α can be safely dropped from the result

Substitution with equivalent formulas

Equivalent-subformula substitution rule

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

- 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: $M(\Gamma \wedge (\beta_1 = \beta_2) \wedge \alpha \wedge \alpha\{\beta_1/\beta_2\}) = M(\Gamma \wedge (\beta_1 \leftrightarrow \beta_2) \wedge \alpha)$
- α can be safely dropped from the result

Universal Instantiation (UI)

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

$$\frac{\Gamma \land \forall x.\alpha}{\Gamma \land \forall x.\alpha \land \alpha\{x/t\}}$$

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

Universal Instantiation (UI)

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

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

  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:
  $M(\Gamma \wedge \forall x.\alpha \wedge \alpha\{x/t\}) = M(\Gamma \wedge \forall x.\alpha)$

Universal Instantiation (UI)

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

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

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

Existential Instantiation (EI)

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

$$\frac{\Gamma \wedge \exists x.\alpha}{\Gamma \wedge \alpha\{x/C\}}$$

  for every variable x and for a "fresh" constant C, i.e. a constant which does not appear in $\Gamma \wedge \exists x.\alpha$
- C is a Skolem constant, EI subcase of Skolemization (see later)
- Intuition: if there is an object satisfying some condition, then we give a (new) name to it
- 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)
  $M(\Gamma \wedge \alpha\{x/C\}) \neq \emptyset$ *iff* $M(\Gamma \wedge \exists x.\alpha) \neq \emptyset$
  (i.e.. $(\Gamma \wedge \alpha\{x/C\}) \models \beta$ iff $(\Gamma \wedge \exists x.\alpha) \models \beta$, for every β)
- Example 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:

$$\frac{\Gamma \wedge \exists x.\alpha}{\Gamma \wedge \alpha\{x/C\}}$$

  for every variable x and for a "fresh" constant C, i.e. a constant which does not appear in $\Gamma \wedge \exists x.\alpha$
- C is a Skolem constant, EI subcase of Skolemization (see later)
- Intuition: if there is an object satisfying some condition, then we give a (new) name to it
- 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)
  $M(\Gamma \wedge \alpha\{x/C\}) \neq \emptyset$ *iff* $M(\Gamma \wedge \exists x.\alpha) \neq \emptyset$
  (i.e.. $(\Gamma \wedge \alpha\{x/C\}) \models \beta$ iff $(\Gamma \wedge \exists x.\alpha) \models \beta$, for every β)
- Example 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:

$$\frac{\Gamma \wedge \exists x.\alpha}{\Gamma \wedge \alpha\{x/C\}}$$

  for every variable x and for a "fresh" constant C, i.e. a constant which does not appear in $\Gamma \wedge \exists x.\alpha$
- C is a Skolem constant, EI subcase of Skolemization (see later)
- Intuition: if there is an object satisfying some condition, then we give a (new) name to it
- 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)
  $M(\Gamma \wedge \alpha\{x/C\}) \neq \emptyset$ *iff* $M(\Gamma \wedge \exists x.\alpha) \neq \emptyset$
  (i.e.. $(\Gamma \wedge \alpha\{x/C\}) \models \beta$ iff $(\Gamma \wedge \exists x.\alpha) \models \beta$, for every β)
- Example 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:

$$\frac{\Gamma \wedge \exists x.\alpha}{\Gamma \wedge \alpha\{x/C\}}$$

  for every variable x and for a "fresh" constant C, i.e. a constant which does not appear in $\Gamma \wedge \exists x.\alpha$
- C is a Skolem constant, EI subcase of Skolemization (see later)
- Intuition: if there is an object satisfying some condition, then we give a (new) name to it
- 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)
  $M(\Gamma \wedge \alpha\{x/C\}) \neq \emptyset$ *iff* $M(\Gamma \wedge \exists x.\alpha) \neq \emptyset$
  (i.e.. $(\Gamma \wedge \alpha\{x/C\}) \models \beta$ iff $(\Gamma \wedge \exists x.\alpha) \models \beta$, for every β)
- Example 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:

$$\frac{\Gamma \wedge \exists x.\alpha}{\Gamma \wedge \alpha\{x/C\}}$$

  for every variable x and for a "fresh" constant C, i.e. a constant which does not appear in $\Gamma \wedge \exists x.\alpha$
- C is a Skolem constant, EI subcase of Skolemization (see later)
- Intuition: if there is an object satisfying some condition, then we give a (new) name to it
- 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)
  $M(\Gamma \wedge \alpha\{x/C\}) \neq \emptyset$ *iff* $M(\Gamma \wedge \exists x.\alpha) \neq \emptyset$
  (i.e.. $(\Gamma \wedge \alpha\{x/C\}) \models \beta$ iff $(\Gamma \wedge \exists x.\alpha) \models \beta$, for every β)
- Example 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 Γ is logically equivalent to the old Γ
- About Existential Instantiation:
  - EI can be applied once to replace the existential sentence;
  - the new Γ is not equivalent to the old,
  - but is (un)satisfiable iff the old Γ is (un)satisfiable

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

Before applying UI or EI, sentences must be rewritten s.t. negations (even when implicit) must be pushed inside the quantifications:

- $\neg \forall x.\alpha \Longrightarrow \exists x.\neg \alpha$
- $\neg \exists x.\alpha \Longrightarrow \forall x.\neg \alpha$
- ex: $\forall x.P(x) \rightarrow \neg \exists y.Q(y)$

  $\Longrightarrow \neg \forall x.P(x) \vee \neg \exists y.Q(y)$

  $\Longrightarrow \exists x.\neg P(x) \vee \forall y.\neg Q(y)$

Remarks

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

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

Before applying UI or EI, sentences must be rewritten s.t. negations (even when implicit) must be pushed inside the quantifications:

- $\neg\forall x.\alpha \Longrightarrow \exists x.\neg\alpha$
- $\neg\exists x.\alpha \Longrightarrow \forall x.\neg\alpha$
- ex: $\forall x.P(x) \rightarrow \neg\exists y.Q(y)$

  $\Longrightarrow \neg\forall x.P(x) \vee \neg\exists y.Q(y)$

  $\Longrightarrow \exists x.\neg P(x) \vee \forall y.\neg Q(y)$

Remarks

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

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

Before applying UI or EI, sentences must be rewritten s.t. negations (even when implicit) must be pushed inside the quantifications:

- $\neg\forall x.\alpha \Longrightarrow \exists x.\neg\alpha$
- $\neg\exists x.\alpha \Longrightarrow \forall x.\neg\alpha$
- ex: $\forall x.P(x) \rightarrow \neg\exists y.Q(y)$
  $\Longrightarrow \neg\forall x.P(x) \vee \neg\exists y.Q(y)$
  $\Longrightarrow \exists x.\neg P(x) \vee \forall y.\neg Q(y)$

Remarks

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

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

Before applying UI or EI, sentences must be rewritten s.t. negations (even when implicit) must be pushed inside the quantifications:

- $\neg\forall x.\alpha \Longrightarrow \exists x.\neg\alpha$
- $\neg\exists x.\alpha \Longrightarrow \forall x.\neg\alpha$
- ex: $\forall x.P(x) \rightarrow \neg\exists y.Q(y)$

  $\Longrightarrow \neg\forall x.P(x) \vee \neg\exists y.Q(y)$

  $\Longrightarrow \exists x.\neg P(x) \vee \forall y.\neg Q(y)$

Outline

Reduction to Propositional Inference (aka propositionalization))

- Idea: Given a FOL closed KB Γ and query α, Convert $(\Gamma \land \neg\alpha)$ to PL
  $\Longrightarrow$ 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 Γ,
  then it is entailed by a finite subset of the propositionalized KB Γ

  $\Longrightarrow$ Every FOL KB Γ 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 (aka propositionalization))

- Idea: Given a FOL closed KB Γ and query α, Convert $(\Gamma \wedge \neg\alpha)$ to PL
  $\Longrightarrow$ 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 Γ,
  then it is entailed by a finite subset of the propositionalized KB Γ

  $\Longrightarrow$ Every FOL KB Γ 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 (aka propositionalization))

- Idea: Given a FOL closed KB Γ and query α, Convert $(\Gamma \wedge \neg\alpha)$ to PL
  $\Longrightarrow$ 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 Γ,
  then it is entailed by a finite subset of the propositionalized KB Γ
  $\Longrightarrow$ Every FOL KB Γ 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 (aka propositionalization))

- Idea: Given a FOL closed KB Γ and query α, Convert $(\Gamma \wedge \neg\alpha)$ to PL
  $\Longrightarrow$ 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 Γ,
  then it is entailed by a finite subset of the propositionalized KB Γ

  $\Longrightarrow$ Every FOL KB Γ 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 Γ 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 Γ 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 Γ ($Evil(John)$ entailed by old Γ)

Reduction to Propositional Inference: Example

- Suppose Γ contains only:

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

- Instantiating the universal sentence in all possible ways:

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

- The new Γ is propositionalized:

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

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

Reduction to Propositional Inference: Example

- Suppose Γ contains only:

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

- Instantiating the universal sentence in all possible ways:

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

- The new Γ is propositionalized:

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

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

Reduction to Propositional Inference: Example

- Suppose Γ contains only:

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

- Instantiating the universal sentence in all possible ways:

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

- The new Γ is propositionalized:

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

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

Problems with Propositionalization

- Propositionalization generates lots of irrelevant sentences

  Ex:

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

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

- With p k-ary predicates and n constants, $p \cdot n^k$ instantiations

Problems with Propositionalization

- Propositionalization generates lots of irrelevant sentences

  Ex:

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

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

- With p k-ary predicates and n constants, $p \cdot n^k$ instantiations

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 Γ by instantiating depth-k terms
  - if $\Gamma \models \alpha$, then will find a contradiction for some finite k
  - if $\Gamma \not\models \alpha$, may find a loop forever
- Theorem: (Turing, 1936), (Church, 1936):
  Entailment in FOL is semidecidable
- Propositionalization not very efficient in general, and used only in very particular cases

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 Γ by instantiating depth-k terms
  - if $\Gamma \models \alpha$, then will find a contradiction for some finite k
  - if $\Gamma \not\models \alpha$, may find a loop forever
- Theorem: (Turing, 1936), (Church, 1936):
  Entailment in FOL is semidecidable
- Propositionalization not very efficient in general, and used only in very particular cases

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 Γ by instantiating depth-k terms
  - if $\Gamma \models \alpha$, then will find a contradiction for some finite k
  - if $\Gamma \not\models \alpha$, may find a loop forever
- Theorem: (Turing, 1936), (Church, 1936):
  Entailment in FOL is semidecidable
- Propositionalization not very efficient in general, and used only in very particular cases

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 Γ by instantiating depth-k terms
  - if $\Gamma \models \alpha$, then will find a contradiction for some finite k
  - if $\Gamma \not\models \alpha$, may find a loop forever
- Theorem: (Turing, 1936), (Church, 1936):
  Entailment in FOL is semidecidable
- Propositionalization not very efficient in general, and used only in very particular cases

Outline

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 variable-to-term 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 (free) 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 variable-to-term 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 (free) 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 variable-to-term 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 (free) 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 variable-to-term 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 (free) 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 variable-to-term 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 (free) 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 variable-to-term 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 (free) 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 variable 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 $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$
  - $\mathit{Unify}(\alpha, \beta) = \theta$ iff $\alpha\theta = \beta\theta$
- Ex:
  $\mathit{Unify}(\mathit{Knows}(\mathit{John}, x), \mathit{Knows}(\mathit{John}, \mathit{Jane})) = \{x/\mathit{Jane}\}$
  $\mathit{Unify}(\mathit{Knows}(\mathit{John}, x), \mathit{Knows}(y, \mathit{OJ})) = \{x/\mathit{OJ}, y/\mathit{John}\}$
  $\mathit{Unify}(\mathit{Knows}(\mathit{John}, x), \mathit{Knows}(y, \mathit{Mother}(y))) = \{y/\mathit{John}, x/\mathit{Mother}(\mathit{John})\}$
  $\mathit{Unify}(\mathit{Knows}(\mathit{John}, x), \mathit{Knows}(x, \mathit{OJ})) = \mathit{FAIL} \ : \ x/?$
- Different (implicitly-universally-quantified) formulas should use different variables!

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

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1, \ \alpha'_2, \ ..., \ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1, \alpha'_2, ..., \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ \dots,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, \dots, \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1, \alpha'_2, ..., \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

- $\{\forall x.\alpha, \forall x.\beta\} \Longleftrightarrow \{\forall x_1.\alpha\{x/x_1\}, \forall x_2.\beta\{x/x_2\}\}$, 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 variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

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

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ \ldots,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, \ldots, \alpha_k \rangle$, find a variable 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 $\langle \alpha'_1,\ \alpha'_2,\ \ldots,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, \ldots, \alpha_k \rangle$
  - $Unify(\alpha, \beta) = \theta$ 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

- $\{\forall x.\alpha, \forall x.\beta\} \Longleftrightarrow \{\forall x_1.\alpha\{x/x_1\}, \forall x_2.\beta\{x/x_2\}\}$, 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 variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

- $\{\forall x.\alpha, \forall x.\beta\} \Longleftrightarrow \{\forall x_1.\alpha\{x/x_1\}, \forall x_2.\beta\{x/x_2\}\}$, 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 variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1, \ \alpha'_2, \ ..., \ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ \dots,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, \dots, \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle\alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k\rangle$ and $\langle\alpha_1, \alpha_2, ..., \alpha_k\rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1, \ \alpha'_2, \ ..., \ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1, \alpha'_2, ..., \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$,
  find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle \alpha'_1, \ \alpha'_2, \ ..., \ \alpha'_k \rangle$ and $\langle \alpha_1, \alpha_2, ..., \alpha_k \rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

Unification

- Unification: Given $\langle\alpha'_1,\ \alpha'_2,\ ...,\ \alpha'_k\rangle$ and $\langle\alpha_1, \alpha_2, ..., \alpha_k\rangle$, find a variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$
- $\{\forall x.\alpha, \forall x.\beta\} \Longleftrightarrow \{\forall x_1.\alpha\{x/x_1\}, \forall x_2.\beta\{x/x_2\}\}$, 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 variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

- $\{\forall x.\alpha, \forall x.\beta\} \Longleftrightarrow \{\forall x_1.\alpha\{x/x_1\}, \forall x_2.\beta\{x/x_2\}\}$, 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 variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

  - $\{\forall x.\alpha, \forall x.\beta\} \Longleftrightarrow \{\forall x_1.\alpha\{x/x_1\}, \forall x_2.\beta\{x/x_2\}\}$, 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 variable 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
$Unify(Knows(John, x_1), Knows(x_2, OJ)) = \{x_1/OJ, x_2/John\}$

  - $\{\forall x.\alpha, \forall x.\beta\} \Longleftrightarrow \{\forall x_1.\alpha\{x/x_1\}, \forall x_2.\beta\{x/x_2\}\}$, 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)

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

Exercises

- Find the MGU of the following formulas by the Unify() procedure, or say there is none. (If needed, standardize apart them beforehand.)
  - $Knows(John, x),\ Knows(y, Mother(y))$
  - $Knows(John, x),\ Knows(x, OJ)$
  - $R(f(x), z),\ R(f(g(B)), y)$
  - $P(f(x)),\ P(g(f(y)))$
  - $P(h(x), B),\ P(A, y)$
- Invent arbitrary pairs of (lists of) atomic FOL formulas and apply Unify() to them

Exercises

- Find the MGU of the following formulas by the Unify() procedure, or say there is none. (If needed, standardize apart them beforehand.)
  - $Knows(John, x),\ Knows(y, Mother(y))$
  - $Knows(John, x),\ Knows(x, OJ)$
  - $R(f(x), z),\ R(f(g(B)), y)$
  - $P(f(x)),\ P(g(f(y)))$
  - $P(h(x), B),\ P(A, y)$
- Invent arbitrary pairs of (lists of) atomic FOL formulas and apply Unify() to them

Outline

Outline

First-Order Definite Clauses & Datalog

- We assume no function symbol and no $\exists$ under the scope of $\forall$ (see later for general case)
- 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
- 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 application: 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 & Datalog

- We assume no function symbol and no $\exists$ under the scope of $\forall$ (see later for general case)
- 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
- 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 application: 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 & Datalog

- We assume no function symbol and no $\exists$ under the scope of $\forall$ (see later for general case)
- 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
- 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 application: 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 & Datalog

- We assume no function symbol and no $\exists$ under the scope of $\forall$ (see later for general case)
- 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
- 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 application: 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 Colonel 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:
  $\forall x,y,z.((American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x,y,z)) \rightarrow Criminal(x))$

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

- 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": $\forall x.(Enemy(x, America) \rightarrow Hostile(x))$

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

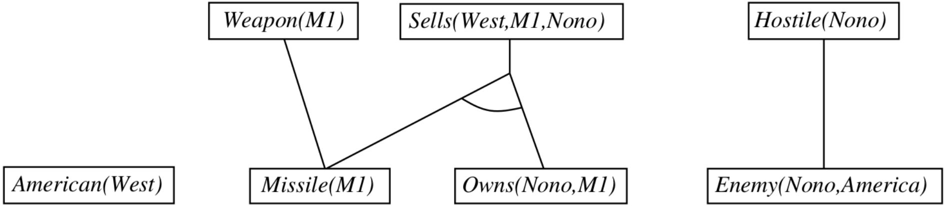
- West, who is American ...: $American(West)$
- The country Nono, an enemy of America ...: $Enemy(Nono, America)$

Example of Forward Chaining

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

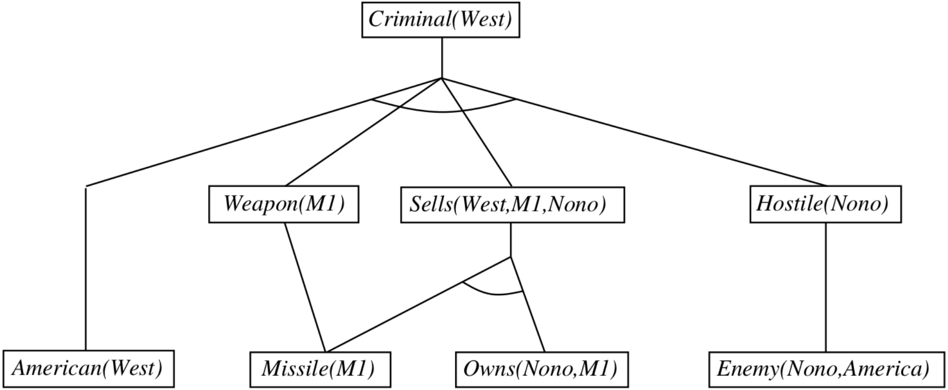
Example of Forward Chaining

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



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

Intuition: at every loop, add all new atomic sentences you can infer by GMP, checking them against the goal

- 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. $p = \#predicates$, $n = \#number\ constants$, $k = maximum\ arity$
- Improvement: match a rule on iteration k only if a premise was 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
    (see AIMA book for reduction of colorability to matching)
- Forward chaining is used in deductive databases and expert systems

Properties of Forward Chaining

Intuition: at every loop, add all new atomic sentences you can infer by GMP, checking them against the goal

- 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. $p = \#predicates$, $n = \#number\ constants$, $k = maximum\ arity$
- Improvement: match a rule on iteration k only if a premise was 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
    (see AIMA book for reduction of colorability to matching)
- Forward chaining is used in deductive databases and expert systems

Properties of Forward Chaining

Intuition: at every loop, add all new atomic sentences you can infer by GMP, checking them against the goal

- 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. $p = \#predicates$, $n = \#number\ constants$, $k = maximum\ arity$
- Improvement: match a rule on iteration k only if a premise was 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
    (see AIMA book for reduction of colorability to matching)
- Forward chaining is used in deductive databases and expert systems

Properties of Forward Chaining

Intuition: at every loop, add all new atomic sentences you can infer by GMP, checking them against the goal

- 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. $p = \#predicates$, $n = \#number\ constants$, $k = maximum\ arity$
- Improvement: match a rule on iteration k only if a premise was 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
    (see AIMA book for reduction of colorability to matching)
- Forward chaining is used in deductive databases and expert systems

Properties of Forward Chaining

Intuition: at every loop, add all new atomic sentences you can infer by GMP, checking them against the goal

- 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. $p = \#predicates$, $n = \#number\ constants$, $k = maximum\ arity$
- Improvement: match a rule on iteration k only if a premise was 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
    (see AIMA book for reduction of colorability to matching)
- Forward chaining is used in deductive databases and expert systems

Properties of Forward Chaining

Intuition: at every loop, add all new atomic sentences you can infer by GMP, checking them against the goal

- 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. $p = \#predicates$, $n = \#number\ constants$, $k = maximum\ arity$
- Improvement: match a rule on iteration k only if a premise was 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
    (see AIMA book for reduction of colorability to matching)
- Forward chaining is used in deductive databases and expert systems

Properties of Forward Chaining

Intuition: at every loop, add all new atomic sentences you can infer by GMP, checking them against the goal

- 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. $p = \#predicates$, $n = \#number\ constants$, $k = maximum\ arity$
- Improvement: match a rule on iteration k only if a premise was 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
    (see AIMA book for reduction of colorability to matching)
- Forward chaining is used in deductive databases and expert systems

Properties of Forward Chaining

Intuition: at every loop, add all new atomic sentences you can infer by GMP, checking them against the goal

- 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. $p = \#predicates$, $n = \#number\ constants$, $k = maximum\ arity$
- Improvement: match a rule on iteration k only if a premise was 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 (see AIMA book for reduction of colorability to matching)
- Forward chaining is used in deductive databases and expert systems

Properties of Forward Chaining

Intuition: at every loop, add all new atomic sentences you can infer by GMP, checking them against the goal

- 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. $p = \#predicates$, $n = \#number\ constants$, $k = maximum\ arity$
- Improvement: match a rule on iteration k only if a premise was 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
    (see AIMA book for reduction of colorability to matching)
- Forward chaining is used in deductive databases and expert systems

Outline

1. Basic First-Order Reasoning
   - Substitutions & Instantiations
   - From Propositional to First-Order Reasoning
   - Unification and Lifting
2. Handling Definite FOL KBs & Datalog
   - Forward Chaining (hints)
   - Backward Chaining (hints)
3. Resolution for General FOL KBs
   - CNF-Ization
   - Resolution
   - A Complete Example

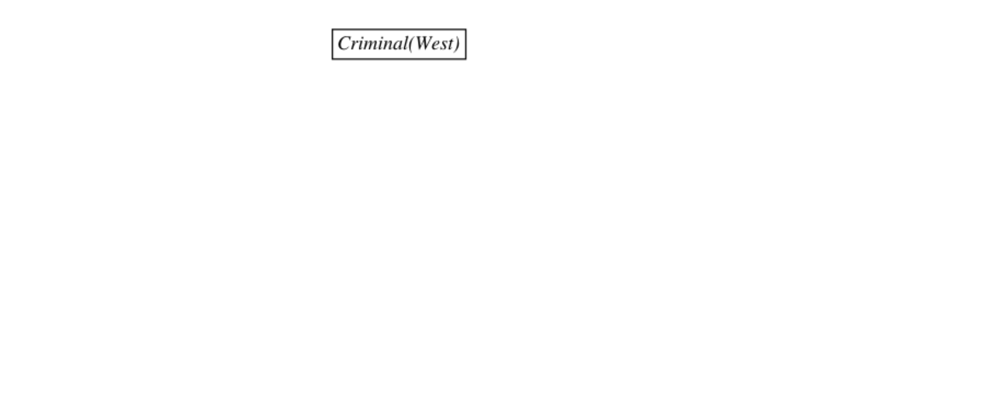
Backward Chaining: Example

$American(West)$, $Missile(M_1)$, $Owns(Nono, M_1)$, $Enemy(Nono, America)$

$\forall x, y, z.((American(x) \land Weapon(y) \land Hostile(z) \land Sells(x, y, z)) \rightarrow Criminal(x))$

$\forall x.(Missile(x) \rightarrow Weapon(x))$ $\forall x.((Missile(x) \land Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$

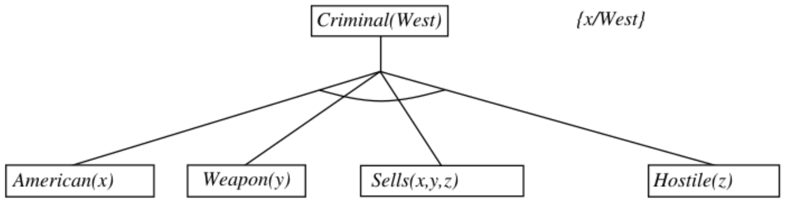
$\forall x.(Enemy(x, America) \rightarrow Hostile(x))$



(© S. Russell & P. Norwig, AIMA)

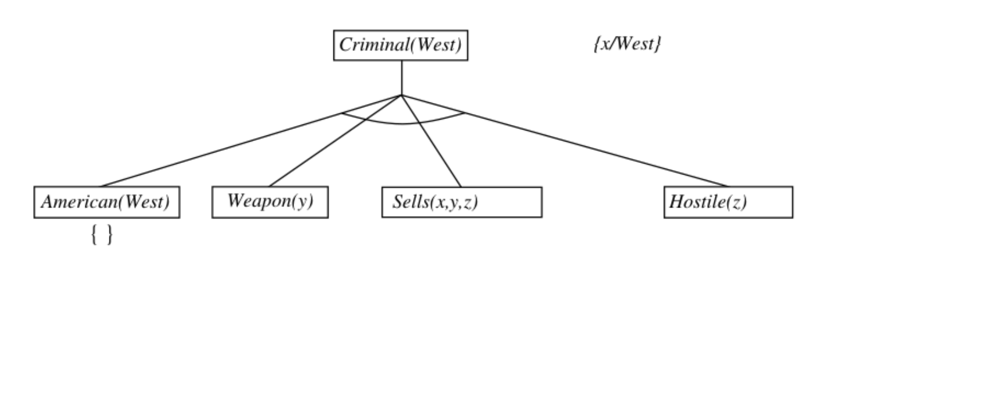
Backward Chaining: Example

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



Backward Chaining: Example

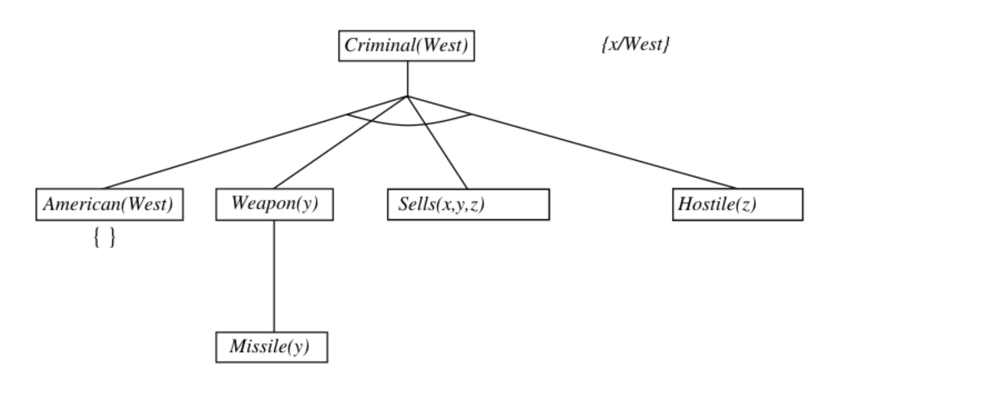
$American(West)$, $Missile(M_1)$, $Owns(Nono, M_1)$, $Enemy(Nono, America)$
$\forall x, y, z.((American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x, y, z)) \rightarrow Criminal(x))$
$\forall x.(Missile(x) \rightarrow Weapon(x))$ $\forall x.((Missile(x) \wedge Owns(Nono, x)) \rightarrow Sells(West,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



(© S. Russell & P. Norwig, AIMA)

Backward Chaining: Example

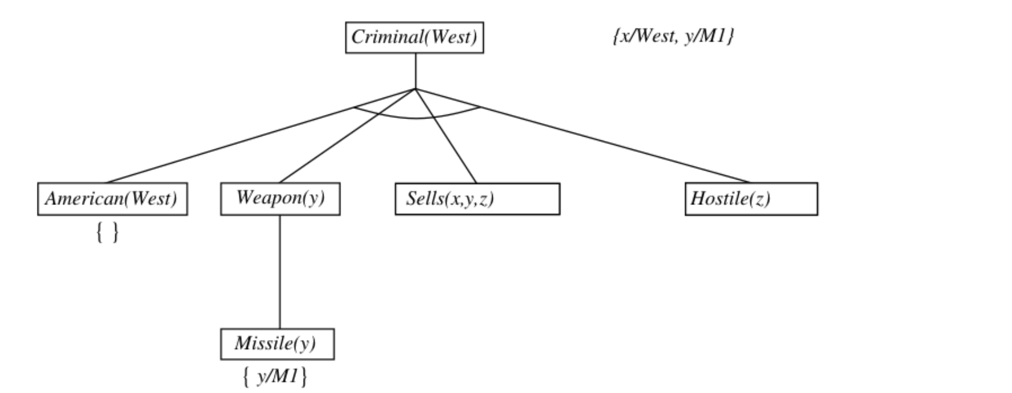
$American(West)$, $Missile(M_1)$, $Owns(Nono, M_1)$, $Enemy(Nono, America)$
$\forall x, y, z.((American(x) \land Weapon(y) \land Hostile(z) \land Sells(x, y, z)) \rightarrow Criminal(x))$
$\forall x.(Missile(x) \rightarrow Weapon(x))$ $\forall x.((Missile(x) \land Owns(Nono, x)) \rightarrow Sells(West,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



(© S. Russell & P. Norwig, AIMA)

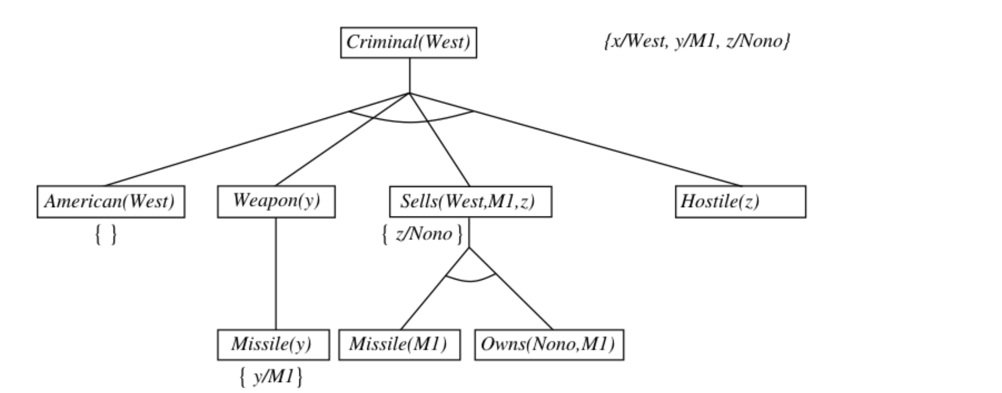
Backward Chaining: Example

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



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)

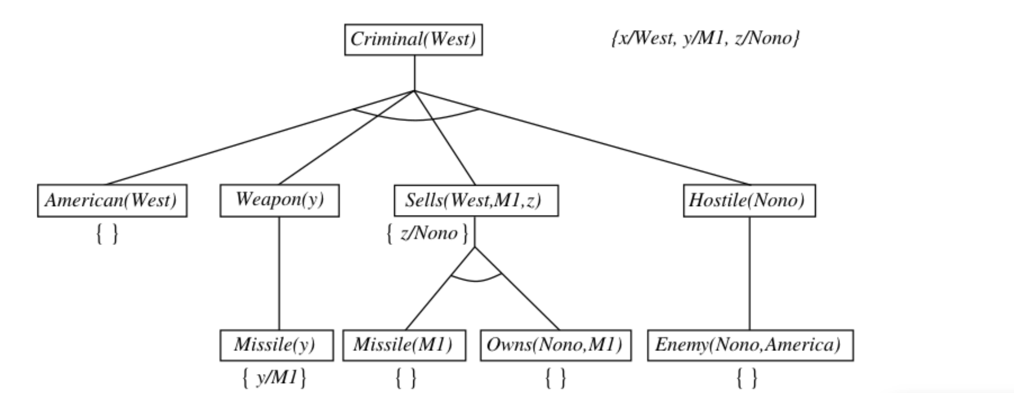
Backward Chaining: Example

$American(West)$, $Missile(M_1)$, $Owns(Nono, M_1)$, $Enemy(Nono, America)$

$\forall x, y, z.((American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x, y, z)) \rightarrow Criminal(x))$

$\forall x.(Missile(x) \rightarrow Weapon(x))$ $\forall x.((Missile(x) \wedge Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$

$\forall x.(Enemy(x, America) \rightarrow Hostile(x))$



(© S. Russell & P. Norwig, AIMA)

Properties of Backward Chaining

Intuition: at every loop, pick a goal and one implication and apply GMP backwards, inferring the list of (unified) premises as sub-goals

- 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)

Properties of Backward Chaining

Intuition: at every loop, pick a goal and one implication and apply GMP backwards, inferring the list of (unified) premises as sub-goals

- 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)

Properties of Backward Chaining

Intuition: at every loop, pick a goal and one implication and apply GMP backwards, inferring the list of (unified) premises as sub-goals

- 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

Intuition: at every loop, pick a goal and one implication and apply GMP backwards, inferring the list of (unified) premises as sub-goals

- 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)

Properties of Backward Chaining

Intuition: at every loop, pick a goal and one implication and apply GMP backwards, inferring the list of (unified) premises as sub-goals

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

Outline

Outline

Conjunctive Normal Form (CNF)

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

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

  - the disjunctions of literals $\bigvee_{j_i=1}^{K_i} l_{j_i}$ are called clauses
  - every literal is 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 $CNF(\varphi)$

Convert into NNF

Every FOL formula φ can be reduced into CNF:

1. Eliminate implications and biconditionals:

$$\alpha \rightarrow \beta \implies \neg\alpha \vee \beta$$
$$\alpha \leftrightarrow \beta \implies (\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$$

2. Push inwards negations recursively:

$$\neg(\alpha \wedge \beta) \implies \neg\alpha \vee \neg\beta$$
$$\neg(\alpha \vee \beta) \implies \neg\alpha \wedge \neg\beta$$
$$\neg\neg\alpha \implies \alpha$$
$$\neg\forall x.\alpha \implies \exists x.\neg\alpha$$
$$\neg\exists x.\alpha \implies \forall x.\neg\alpha$$

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

$\implies$ quantified subformulas occur only with positive polarity

FOL CNF Conversion $CNF(\varphi)$

Convert into NNF

Every FOL formula φ can be reduced into CNF:

1. Eliminate implications and biconditionals:

$$\alpha \to \beta \implies \neg\alpha \vee \beta$$
$$\alpha \leftrightarrow \beta \implies (\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$$

2. Push inwards negations recursively:

$$\neg(\alpha \wedge \beta) \implies \neg\alpha \vee \neg\beta$$
$$\neg(\alpha \vee \beta) \implies \neg\alpha \wedge \neg\beta$$
$$\neg\neg\alpha \implies \alpha$$
$$\neg\forall x.\alpha \implies \exists x.\neg\alpha$$
$$\neg\exists x.\alpha \implies \forall x.\neg\alpha$$

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

$\implies$ quantified subformulas occur only with positive polarity

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

Remove quantifiers

3. **Standardize variables**: each quantifier should use a different var
   $(\forall x.\exists y.\alpha) \wedge \exists y.\beta \wedge \forall x.\gamma \implies (\forall x.\exists y.\alpha) \wedge \exists y_1.\beta\{y/y_1\} \wedge \forall x_1.\gamma\{x/x_1\}$
4. **Skolemize** (a generalization of EI):
   Each existential variable is replaced by a fresh Skolem function applied to the enclosing universally-quantified variables
   $$\begin{array}{lcl} \exists y.\alpha & \implies & \alpha\{y/c\} \\ \forall x.(...\exists y.\alpha...) & \implies & \forall x.(...\alpha\{y/F_1(x)\}...) \\ \forall x_1 x_2.(...\exists y.\alpha...) & \implies & \forall x_1 x_2.(...\alpha\{y/F_1(x_1,x_2)...)\} \\ \exists y_1 \forall x_1 x_2 \exists y_2 \forall x_3 \exists y_3.\alpha & \implies & \forall x_1 x_2 x_3.\alpha\{y_1/c, y_2/F_1(x_1,x_2), y_3/F_2(x_1,x_2,x_3)\} \end{array}$$
   Ex: $\forall x \exists y.Father(y,x) \implies \forall x.Father(s(x),x)$
   ($s(x)$ implictly means "father of x" although s() is a fresh function)
5. **Drop universal quantifiers:** $\forall x_1...x_k.\alpha \implies \alpha$
   $\implies$ free variables implicitly universally quantified

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

Remove quantifiers

3. Standardize variables: each quantifier should use a different var
   $(\forall x.\exists y.\alpha) \land \exists y.\beta \land \forall x.\gamma \implies (\forall x.\exists y.\alpha) \land \exists y_1.\beta\{y/y_1\} \land \forall x_1.\gamma\{x/x_1\}$
4. Skolemize (a generalization of EI):
   Each existential variable is replaced by a fresh Skolem function applied to the enclosing universally-quantified variables
   $\exists y.\alpha \implies \alpha\{y/c\}$
   $\forall x.(...\exists y.\alpha...) \implies \forall x.(...\alpha\{y/F_1(x)\}...)$
   $\forall x_1 x_2.(...\exists y.\alpha...) \implies \forall x_1 x_2.(...\alpha\{y/F_1(x_1,x_2)...)\}$
   $\exists y_1 \forall x_1 x_2 \exists y_2 \forall x_3 \exists y_3.\alpha \implies \forall x_1 x_2 x_3.\alpha\{y_1/c, y_2/F_1(x_1,x_2), y_3/F_2(x_1,x_2,x_3)\}$
   Ex: $\forall x \exists y.Father(y,x) \implies \forall x.Father(s(x),x)$
   ($s(x)$ implictly means "father of x" although s() is a fresh function)
5. Drop universal quantifiers: $\forall x_1 ... x_k.\alpha \implies \alpha$
   $\implies$ free variables implicitly universally quantified

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

Remove quantifiers

3. Standardize variables: each quantifier should use a different var
   $(\forall x.\exists y.\alpha) \land \exists y.\beta \land \forall x.\gamma \implies (\forall x.\exists y.\alpha) \land \exists y_1.\beta\{y/y_1\} \land \forall x_1.\gamma\{x/x_1\}$
4. Skolemize (a generalization of EI):
   Each existential variable is replaced by a fresh Skolem function applied to the enclosing universally-quantified variables
   $$\exists y.\alpha \implies \alpha\{y/c\}$$
   $$\forall x.(...\exists y.\alpha...) \implies \forall x.(...\alpha\{y/F_1(x)\}...)$$
   $$\forall x_1 x_2.(...\exists y.\alpha...) \implies \forall x_1 x_2.(...\alpha\{y/F_1(x_1,x_2)...)\}$$
   $$\exists y_1 \forall x_1 x_2 \exists y_2 \forall x_3 \exists y_3.\alpha \implies \forall x_1 x_2 x_3.\alpha\{y_1/c, y_2/F_1(x_1,x_2), y_3/F_2(x_1,x_2,x_3)\}$$
   Ex: $\forall x \exists y.Father(y,x) \implies \forall x.Father(s(x),x)$
   ($s(x)$ implictly means "father of x" although s() is a fresh function)
5. Drop universal quantifiers: $\forall x_1 ... x_k.\alpha \implies \alpha$
   $\implies$ free variables implicitly universally quantified

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

Remove quantifiers

3. Standardize variables: each quantifier should use a different var
   $(\forall x.\exists y.\alpha) \land \exists y.\beta \land \forall x.\gamma \implies (\forall x.\exists y.\alpha) \land \exists y_1.\beta\{y/y_1\} \land \forall x_1.\gamma\{x/x_1\}$
4. Skolemize (a generalization of EI):
   Each existential variable is replaced by a fresh Skolem function applied to the enclosing universally-quantified variables
   $\exists y.\alpha \implies \alpha\{y/c\}$
   $\forall x.(...\exists y.\alpha...) \implies \forall x.(...\alpha\{y/F_1(x)\}...)$
   $\forall x_1 x_2.(...\exists y.\alpha...) \implies \forall x_1 x_2.(...\alpha\{y/F_1(x_1, x_2)...)\}$
   $\exists y_1 \forall x_1 x_2 \exists y_2 \forall x_3 \exists y_3.\alpha \implies \forall x_1 x_2 x_3.\alpha\{y_1/c, y_2/F_1(x_1, x_2), y_3/F_2(x_1, x_2, x_3)\}$
   Ex: $\forall x \exists y.Father(y, x) \implies \forall x.Father(s(x), x)$
   ($s(x)$ implictly means "father of x" although s() is a fresh function)
5. Drop universal quantifiers: $\forall x_1 ... x_k.\alpha \implies \alpha$
   $\implies$ free variables implicitly universally quantified

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

CNF-ize propositionally

6. CNF-ize propositionally (see previous chapters):
   either apply recursively the DeMorgan's Rule: $(\alpha \wedge \beta) \vee \gamma \implies (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$
   or rename subformulas and add definitions: $(\alpha \wedge \beta) \vee \gamma \implies (B \vee \gamma) \wedge CNF(B \leftrightarrow (\alpha \wedge \beta))$
7. Standardize Apart (again) (Personal suggestion, not in AIMA book):
   prevent the same (implicitly universally-quantified) variable to occur in distinct clauses
   (correct because $\forall x.(\alpha \wedge \beta)$ equivalent to $\forall x.\alpha \wedge \forall y.\beta$)

Properties of FOL CNF-ization

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

$\Longrightarrow$ Preserves entailment: $\varphi \models \alpha$ iff $CNF(\varphi) \models \alpha$ (in fact, $\varphi \wedge \neg\alpha$ unsat iff $\varphi \wedge \neg CNF(\alpha)$ unsat)

- Does not preserve validity (but we do not need it)

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

CNF-ize propositionally

6. CNF-ize propositionally (see previous chapters):
   either apply recursively the DeMorgan's Rule: $(\alpha \wedge \beta) \vee \gamma \implies (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$
   or rename subformulas and add definitions: $(\alpha \wedge \beta) \vee \gamma \implies (B \vee \gamma) \wedge CNF(B \leftrightarrow (\alpha \wedge \beta))$
7. Standardize Apart (again) (Personal suggestion, not in AIMA book):
   prevent the same (implicitly universally-quantified) variable to occur in distinct clauses
   (correct because $\forall x.(\alpha \wedge \beta)$ equivalent to $\forall x.\alpha \wedge \forall y.\beta$)

Properties of FOL CNF-ization

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

$\Longrightarrow$ Preserves entailment: $\varphi \models \alpha$ iff $CNF(\varphi) \models \alpha$ (in fact, $\varphi \wedge \neg\alpha$ unsat iff $\varphi \wedge \neg CNF(\alpha)$ unsat)

- Does not preserve validity (but we do not need it)

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

CNF-ize propositionally

6. CNF-ize propositionally (see previous chapters):
   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: $(\alpha \wedge \beta) \vee \gamma \quad \Longrightarrow \quad (B \vee \gamma) \wedge CNF(B \leftrightarrow (\alpha \wedge \beta))$
7. Standardize Apart (again) (Personal suggestion, not in AIMA book):
   prevent the same (implicitly universally-quantified) variable to occur in distinct clauses
   (correct because $\forall x.(\alpha \wedge \beta)$ equivalent to $\forall x.\alpha \wedge \forall y.\beta$)

Properties of FOL CNF-ization

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

$\Longrightarrow$ Preserves entailment: $\varphi \models \alpha$ iff $CNF(\varphi) \models \alpha$ (in fact, $\varphi \wedge \neg\alpha$ unsat iff $\varphi \wedge \neg CNF(\alpha)$ unsat)

- Does not preserve validity (but we do not need it)

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

CNF-ize propositionally

6. CNF-ize propositionally (see previous chapters):
   either apply recursively the DeMorgan's Rule: $(\alpha \wedge \beta) \vee \gamma \implies (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$
   or rename subformulas and add definitions: $(\alpha \wedge \beta) \vee \gamma \implies (B \vee \gamma) \wedge CNF(B \leftrightarrow (\alpha \wedge \beta))$
7. Standardize Apart (again) (Personal suggestion, not in AIMA book):
   prevent the same (implicitly universally-quantified) variable to occur in distinct clauses
   (correct because $\forall x.(\alpha \wedge \beta)$ equivalent to $\forall x.\alpha \wedge \forall y.\beta$)

Properties of FOL CNF-ization

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

$\implies$ Preserves entailment: $\varphi \models \alpha$ iff $CNF(\varphi) \models \alpha$ (in fact, $\varphi \wedge \neg\alpha$ unsat iff $\varphi \wedge \neg CNF(\alpha)$ unsat)

- Does not preserve validity (but we do not need it)

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 (NNF)

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

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 (and standardize apart the result):

$(Animal(F(x)) \vee Loves(G(x),x)) \wedge (\neg Loves(x_1,F(x_1)) \vee Loves(G(x_1),x_1))$

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 (NNF)

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

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 (and standardize apart the result):

$(Animal(F(x)) \vee Loves(G(x),x)) \wedge (\neg Loves(x_1,F(x_1)) \vee Loves(G(x_1),x_1))$

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 (NNF)

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

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 (and standardize apart the result):

$(Animal(F(x)) \vee Loves(G(x),x)) \wedge (\neg Loves(x_1,F(x_1)) \vee Loves(G(x_1),x_1))$

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 (NNF)

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

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 (and standardize apart the result):

$$(Animal(F(x)) \vee Loves(G(x),x)) \wedge (\neg Loves(x_1,F(x_1)) \vee Loves(G(x_1),x_1))$$

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 (NNF)

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

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 (and standardize apart the result):

$(Animal(F(x)) \vee Loves(G(x),x)) \wedge (\neg Loves(x_1,F(x_1)) \vee Loves(G(x_1),x_1))$

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 (NNF)
   $$\forall x.([\exists y.\neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y.Loves(y,x)])$$
   $$\forall x.([\exists y.(\neg\neg Animal(y) \wedge \neg Loves(x,y))] \vee [\exists y.Loves(y,x)])$$
   $$\forall x.([\exists y.(Animal(y) \wedge \neg Loves(x,y))] \vee [\exists y.Loves(y,x)])$$
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 (and standardize apart the result):

$$(Animal(F(x)) \vee Loves(G(x),x)) \wedge (\neg Loves(x_1,F(x_1)) \vee Loves(G(x_1),x_1))$$

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 (NNF)

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

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 (and standardize apart the result):

$(Animal(F(x)) \lor Loves(G(x),x)) \land (\neg Loves(x_1,F(x_1)) \lor Loves(G(x_1),x_1))$

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 (NNF)

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

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 (and standardize apart the result):

   $(Animal(F(x)) \vee Loves(G(x),x)) \wedge (\neg Loves(x_1,F(x_1)) \vee Loves(G(x_1),x_1))$

Remark about Skolemization

Common mistake to avoid

- Do not
  - apply Skolemization or
  - drop universal quantifiers

  before converting into NNF & standardize apart variables!
- Polarity of quantified subformulas affects Skolemization!

$\Longrightarrow$ NNF-ization may convert $\exists$'s into $\forall$'s, and vice versa

- Same-name quantified variable may cause errors

$\Longrightarrow$ standardize variable may rename variables
(which, e.g., could be wrongly Skolemized into the same function)

Remark about Skolemization

Common mistake to avoid

- Do <u>not</u>
  - apply Skolemization or
  - drop universal quantifiers

  before converting into NNF & standardize apart variables!
- Polarity of quantified subformulas affects Skolemization!

$\Longrightarrow$ NNF-ization may convert $\exists$'s into $\forall$'s, and vice versa

- Same-name quantified variable may cause errors

$\Longrightarrow$ standardize variable may rename variables
(which, e.g., could be wrongly Skolemized into the same function)

Remark about Skolemization

Common mistake to avoid

- Do not
  - apply Skolemization or
  - drop universal quantifiers

  before converting into NNF & standardize apart variables!
- Polarity of quantified subformulas affects Skolemization!

$\Longrightarrow$ NNF-ization may convert $\exists$'s into $\forall$'s, and vice versa

- Same-name quantified variable may cause errors

$\Longrightarrow$ standardize variable may rename variables
(which, e.g., could be wrongly Skolemized into the same function)

Remark about Skolemization: Example

Wrong CNF-ization

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

1. Too-early Skolemization & universal-quantifier dropping:
   $\forall x.([\forall y.(Animal(y) \rightarrow Loves(x,y))] \rightarrow [Loves(G(x),x)])$
   $([(Animal(y) \rightarrow Loves(x,y))] \rightarrow [Loves(G(x),x)])$
2. NNF-ization and CNF-ization $([(Animal(y) \land \neg Loves(x,y))] \lor [Loves(G(x),x)])$
   $((Animal(y) \lor Loves(G(x),x)) \land ((\neg Loves(x,y)) \lor Loves(G(x),x)))$

"y" should be a Skolem function F(x) instead
because "$\forall y.(...)$" occurred negatively
$\Longrightarrow$ should become "$\exists y.\neg(...)$", and hence y Skolemized into $F(x)$
(compare with previous slide)

Remark about Skolemization: Example

Wrong CNF-ization

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

1. Too-early Skolemization & universal-quantifier dropping:
   $\forall x.([\forall y.(Animal(y) \rightarrow Loves(x,y))] \rightarrow [Loves(G(x),x)])$
   $([(Animal(y) \rightarrow Loves(x,y))] \rightarrow [Loves(G(x),x)])$
2. NNF-ization and CNF-ization $([(Animal(y) \wedge \neg Loves(x,y))] \vee [Loves(G(x),x)])$
   $((Animal(y) \vee Loves(G(x),x)) \wedge ((\neg Loves(x,y)) \vee Loves(G(x),x)))$

"y" should be a Skolem function F(x) instead
because "$\forall y.(...)$" occurred negatively
$\Longrightarrow$ should become "$\exists y.\neg(...)$", and hence y Skolemized into $F(x)$
(compare with previous slide)

Remark about Skolemization: Example

Wrong CNF-ization

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

1. Too-early Skolemization & universal-quantifier dropping:
   $\forall x.([\forall y.(Animal(y) \rightarrow Loves(x,y))] \rightarrow [Loves(G(x),x)])$
   $([(Animal(y) \rightarrow Loves(x,y))] \rightarrow [Loves(G(x),x)])$
2. NNF-ization and CNF-ization $([(Animal(y) \wedge \neg Loves(x,y))] \vee [Loves(G(x),x)])$
   $((Animal(y) \vee Loves(G(x),x)) \wedge ((\neg Loves(x,y)) \vee Loves(G(x),x)))$

"y" should be a Skolem function F(x) instead
because "$\forall y.(...)$" occurred negatively
$\Longrightarrow$ should become "$\exists y.\neg(...)$", and hence y Skolemized into $F(x)$
(compare with previous slide)

Remark about Skolemization: Example

Wrong CNF-ization

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

1. Too-early Skolemization & universal-quantifier dropping:
   $\forall x.([\forall y.(Animal(y) \rightarrow Loves(x, y))] \rightarrow [Loves(G(x), x)])$
   $([(Animal(y) \rightarrow Loves(x, y))] \rightarrow [Loves(G(x), x)])$
2. NNF-ization and CNF-ization $([(Animal(y) \wedge \neg Loves(x, y))] \vee [Loves(G(x), x)])$
   $((Animal(y) \vee Loves(G(x), x)) \wedge ((\neg Loves(x, y)) \vee Loves(G(x), x)))$

"y" should be a Skolem function F(x) instead
because "$\forall y.(...)$" occurred negatively
$\Longrightarrow$ should become "$\exists y.\neg(...)$", and hence y Skolemized into $F(x)$
(compare with previous slide)

Exercise

Did Curiosity kill the cat?

Formalize and CNF-ize the following:

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 also AIMA book for FOL formalization and CNF-ization)

Outline

Resolution

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

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

- Ex: $\dfrac{Man(Socrates) \qquad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $\Gamma \models \alpha$ in FOL:
  - convert $\Gamma \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(\Gamma \wedge \neg\alpha)$ until either
    - the empty clause is generated $\Longrightarrow \Gamma \models \alpha$
    - no more resolution step is applicable $\Longrightarrow \Gamma \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
  - Unit resolution alone complete for definite clauses
    - choose positive unit-clauses first (DFS) $\Longrightarrow$ Forward chaining
    - choose negative clauses first (DFS) $\Longrightarrow$ Backward chaining
- Refutation-Complete:
  - If there is a substitution θ such that $\Gamma \models \theta\alpha$, 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 $\theta \stackrel{\text{def}}{=} mgu(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

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

- Ex: $\frac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $\Gamma \models \alpha$ in FOL:
  - convert $\Gamma \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(\Gamma \wedge \neg\alpha)$ until either
    - the empty clause is generated $\Longrightarrow \Gamma \models \alpha$
    - no more resolution step is applicable $\Longrightarrow \Gamma \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
  - Unit resolution alone complete for definite clauses
    - choose positive unit-clauses first (DFS) $\Longrightarrow$ Forward chaining
    - choose negative clauses first (DFS) $\Longrightarrow$ Backward chaining
- Refutation-Complete:
  - If there is a substitution θ such that $\Gamma \models \theta\alpha$, 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 $\theta \stackrel{\text{def}}{=} mgu(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

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

- Ex: $\frac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $\Gamma \models \alpha$ in FOL:
  - convert $\Gamma \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(\Gamma \wedge \neg\alpha)$ until either
    - the empty clause is generated $\Longrightarrow \Gamma \models \alpha$
    - no more resolution step is applicable $\Longrightarrow \Gamma \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
  - Unit resolution alone complete for definite clauses
    - choose positive unit-clauses first (DFS) $\Longrightarrow$ Forward chaining
    - choose negative clauses first (DFS) $\Longrightarrow$ Backward chaining
- Refutation-Complete:
  - If there is a substitution θ such that $\Gamma \models \theta\alpha$, 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 $\theta \stackrel{\text{def}}{=} mgu(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

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

- Ex: $\frac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $\Gamma \models \alpha$ in FOL:
  - convert $\Gamma \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(\Gamma \wedge \neg\alpha)$ until either
    - the empty clause is generated $\Longrightarrow \Gamma \models \alpha$
    - no more resolution step is applicable $\Longrightarrow \Gamma \not\models \alpha$
    - resources (time, memory) exhausted $\Longrightarrow$???
  - Hint: apply resolution first to unit clauses (unit resolution)
  - Unit resolution alone complete for definite clauses
    - choose positive unit-clauses first (DFS) $\Longrightarrow$ Forward chaining
    - choose negative clauses first (DFS) $\Longrightarrow$ Backward chaining
- Refutation-Complete:
  - If there is a substitution θ such that $\Gamma \models \theta\alpha$, 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 $\theta \stackrel{\text{def}}{=} mgu(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

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

- Ex: $\frac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $\Gamma \models \alpha$ in FOL:
  - convert $\Gamma \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(\Gamma \wedge \neg\alpha)$ until either
    - the empty clause is generated $\Longrightarrow \Gamma \models \alpha$
    - no more resolution step is applicable $\Longrightarrow \Gamma \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
  - Unit resolution alone complete for definite clauses
    - choose positive unit-clauses first (DFS) $\Longrightarrow$ Forward chaining
    - choose negative clauses first (DFS) $\Longrightarrow$ Backward chaining
- Refutation-Complete:
  - If there is a substitution θ such that $\Gamma \models \theta\alpha$, 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 $\theta \stackrel{\text{def}}{=} mgu(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

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

- Ex: $\frac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $\Gamma \models \alpha$ in FOL:
  - convert $\Gamma \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(\Gamma \wedge \neg\alpha)$ until either
    - the empty clause is generated $\Longrightarrow \Gamma \models \alpha$
    - no more resolution step is applicable $\Longrightarrow \Gamma \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
  - Unit resolution alone complete for definite clauses
    - choose positive unit-clauses first (DFS) $\Longrightarrow$ Forward chaining
    - choose negative clauses first (DFS) $\Longrightarrow$ Backward chaining
- Refutation-Complete:
  - If there is a substitution θ such that $\Gamma \models \theta\alpha$, 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 $\theta \stackrel{\text{def}}{=} mgu(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

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

- Ex: $\dfrac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $\Gamma \models \alpha$ in FOL:
  - convert $\Gamma \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(\Gamma \wedge \neg\alpha)$ until either
    - the empty clause is generated $\Longrightarrow \Gamma \models \alpha$
    - no more resolution step is applicable $\Longrightarrow \Gamma \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
  - Unit resolution alone complete for definite clauses
    - choose positive unit-clauses first (DFS) $\Longrightarrow$ Forward chaining
    - choose negative clauses first (DFS) $\Longrightarrow$ Backward chaining
- Refutation-Complete:
  - If there is a substitution θ such that $\Gamma \models \theta\alpha$, 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 $\theta \stackrel{\text{def}}{=} mgu(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

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

- Ex: $\dfrac{Man(Socrates) \qquad (\neg Man(x) \lor Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $\Gamma \models \alpha$ in FOL:
  - convert $\Gamma \land \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(\Gamma \land \neg\alpha)$ until either
    - the empty clause is generated $\Longrightarrow \Gamma \models \alpha$
    - no more resolution step is applicable $\Longrightarrow \Gamma \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
  - Unit resolution alone complete for definite clauses
    - choose positive unit-clauses first (DFS) $\Longrightarrow$ Forward chaining
    - choose negative clauses first (DFS) $\Longrightarrow$ Backward chaining
- Refutation-Complete:
  - If there is a substitution θ such that $\Gamma \models \theta\alpha$, 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 $\theta \stackrel{\text{def}}{=} mgu(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

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

- Ex: $\frac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $\Gamma \models \alpha$ in FOL:
  - convert $\Gamma \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(\Gamma \wedge \neg\alpha)$ until either
    - the empty clause is generated $\Longrightarrow \Gamma \models \alpha$
    - no more resolution step is applicable $\Longrightarrow \Gamma \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
  - Unit resolution alone complete for definite clauses
    - choose positive unit-clauses first (DFS) $\Longrightarrow$ Forward chaining
    - choose negative clauses first (DFS) $\Longrightarrow$ Backward chaining
- Refutation-Complete:
  - If there is a substitution θ such that $\Gamma \models \theta\alpha$, 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 $\theta \stackrel{\text{def}}{=} mgu(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

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

- Ex: $\dfrac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $\Gamma \models \alpha$ in FOL:
  - convert $\Gamma \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(\Gamma \wedge \neg\alpha)$ until either
    - the empty clause is generated $\Longrightarrow \Gamma \models \alpha$
    - no more resolution step is applicable $\Longrightarrow \Gamma \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
  - Unit resolution alone complete for definite clauses
    - choose positive unit-clauses first (DFS) $\Longrightarrow$ Forward chaining
    - choose negative clauses first (DFS) $\Longrightarrow$ Backward chaining
- Refutation-Complete:
  - If there is a substitution θ such that $\Gamma \models \theta\alpha$, 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 $\theta \stackrel{\text{def}}{=} mgu(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

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

- Ex: $\dfrac{Man(Socrates) \qquad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $\Gamma \models \alpha$ in FOL:
  - convert $\Gamma \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(\Gamma \wedge \neg\alpha)$ until either
    - the empty clause is generated $\Longrightarrow \Gamma \models \alpha$
    - no more resolution step is applicable $\Longrightarrow \Gamma \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
  - Unit resolution alone complete for definite clauses
    - choose positive unit-clauses first (DFS) $\Longrightarrow$ Forward chaining
    - choose negative clauses first (DFS) $\Longrightarrow$ Backward chaining
- Refutation-Complete:
  - If there is a substitution θ such that $\Gamma \models \theta\alpha$, 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 $\theta \stackrel{\text{def}}{=} mgu(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

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

- Ex: $\frac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $\Gamma \models \alpha$ in FOL:
  - convert $\Gamma \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(\Gamma \wedge \neg\alpha)$ until either
    - the empty clause is generated $\Longrightarrow \Gamma \models \alpha$
    - no more resolution step is applicable $\Longrightarrow \Gamma \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
  - Unit resolution alone complete for definite clauses
    - choose positive unit-clauses first (DFS) $\Longrightarrow$ Forward chaining
    - choose negative clauses first (DFS) $\Longrightarrow$ Backward chaining
- Refutation-Complete:
  - If there is a substitution θ such that $\Gamma \models \theta\alpha$, 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 $\theta \stackrel{\text{def}}{=} mgu(l_i, \neg m_j)$, s.t. $l_i\theta = \neg m_j\theta$:

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

- Ex: $\frac{Man(Socrates) \quad (\neg Man(x) \vee Mortal(x))}{Mortal(Socrates)}$ s.t. $\theta \stackrel{\text{def}}{=} \{x/Socrates\}$
- To prove that $\Gamma \models \alpha$ in FOL:
  - convert $\Gamma \wedge \neg\alpha$ to CNF
  - apply repeatedly resolution rule to $CNF(\Gamma \wedge \neg\alpha)$ until either
    - the empty clause is generated $\Longrightarrow \Gamma \models \alpha$
    - no more resolution step is applicable $\Longrightarrow \Gamma \not\models \alpha$
    - resource (time, memory) exhausted $\Longrightarrow$??
  - Hint: apply resolution first to unit clauses (unit resolution)
  - Unit resolution alone complete for definite clauses
    - choose positive unit-clauses first (DFS) $\Longrightarrow$ Forward chaining
    - choose negative clauses first (DFS) $\Longrightarrow$ Backward chaining
- Refutation-Complete:
  - If there is a substitution θ such that $\Gamma \models \theta\alpha$, 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

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 Colonel West is a criminal.

Example: Resolution with Definite Clauses [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
  $\forall x.((Missile(x) \wedge Owns(Nono, x)) \rightarrow Sells(West, x, Nono))$

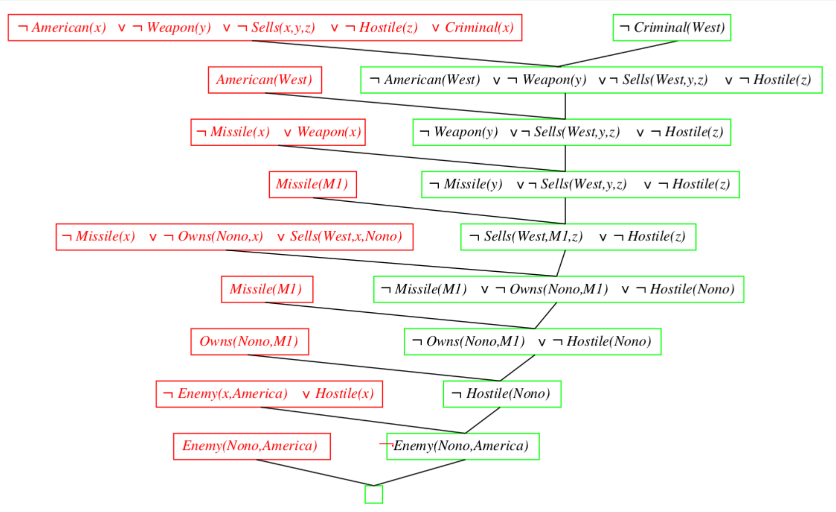
$\Longrightarrow \neg Missile(x) \vee \neg Owns(Nono, x) \vee Sells(West, x, Nono)$

- 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: Resolution with Definite Clauses



(© S. Russell & P. Norwig, AIMA)

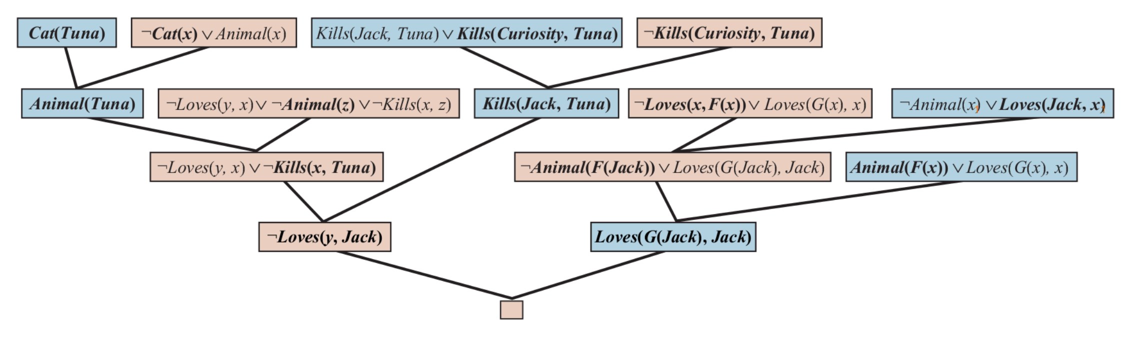
Exercise: Resolution with Definite Clauses

Resolve the problem of previous example:

1. selecting positive unit clauses first (DFS) $\Longrightarrow$ Forward chaining
2. selecting negative clauses first first (DFS) $\Longrightarrow$ Backward chaining
3. selecting unit-literals in any order first $\Longrightarrow$ Mixed chaining

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 previous exercise or AIMA book for FOL formalization and CNF-ization.)



Resolution Strategies

Saturation Calculus:

- Given N_0 : set of (implicitly universally quantified) clauses.
- Derive N_0, N_1, N_2, N_3, ... s.t. $N_{i+1} = N_i \cup \{C\}$,
  - where C is the conclusion of a resolution step from premises in N_i
- (under reasonable restrictions) is refutationally complete :

$$N_0 \models \bot \implies \bot \in N_i \text{ for some i}$$

Problem

- The resolution rule is prolific.
  - it generates many useless intermediate results
  - it may generate the same clauses in many different ways
- This motivates the introduction of resolution restrictions.

Resolution Strategies

Saturation Calculus:

- Given N_0 : set of (implicitly universally quantified) clauses.
- Derive N_0, N_1, N_2, N_3, ... s.t. $N_{i+1} = N_i \cup \{C\}$,
  - where C is the conclusion of a resolution step from premises in N_i
- (under reasonable restrictions) is refutationally complete :

$$N_0 \models \bot \implies \bot \in N_i \text{ for some i}$$

Problem

- The resolution rule is prolific.
  - it generates many useless intermediate results
  - it may generate the same clauses in many different ways
- This motivates the introduction of resolution restrictions.

Resolution Strategies

Saturation Calculus:

- Given N_0 : set of (implicitly universally quantified) clauses.
- Derive N_0, N_1, N_2, N_3, ... s.t. $N_{i+1} = N_i \cup \{C\}$,
  - where C is the conclusion of a resolution step from premises in N_i
- (under reasonable restrictions) is refutationally complete :

$$N_0 \models \bot \implies \bot \in N_i \text{ for some i}$$

Problem

- The resolution rule is prolific.
  - it generates many useless intermediate results
  - it may generate the same clauses in many different ways
- This motivates the introduction of resolution restrictions.

Resolution Strategies

Saturation Calculus:

- Given N_0 : set of (implicitly universally quantified) clauses.
- Derive $N_0, N_1, N_2, N_3, \ldots$ s.t. $N_{i+1} = N_i \cup \{C\}$,
  - where C is the conclusion of a resolution step from premises in N_i
- (under reasonable restrictions) is refutationally complete :

$$N_0 \models \bot \implies \bot \in N_i \text{ for some i}$$

Problem

- The resolution rule is prolific.
  - it generates many useless intermediate results
  - it may generate the same clauses in many different ways
- This motivates the introduction of resolution restrictions.

Resolution Restrictions

Ordered resolution

- define stable atom ordering;
- resolve only maximal literals

Hyper-Resolution

- Clauses are divided into
  - "nuclei": those with ≥ 1 negative literals
  - "electrons" : those with positive literals only
- Resolution can occur only among one nucleus and one electron
- $$Ex: \quad \dfrac{\dfrac{\neg P(x) \vee \neg Q(x) \vee R(x) \qquad Q(A) \vee C}{\neg P(A) \vee R(A) \vee C} \qquad P(A) \vee D}{R(A) \vee C \vee D}$$
- Multiple resolution steps are merged into one step
- $$Ex: \quad \dfrac{\neg P(x) \vee \neg Q(x) \vee R(x) \qquad Q(A) \vee C \qquad P(A) \vee D}{R(A) \vee C \vee D}$$

$\Longrightarrow$ Globally, can produce only electrons

Resolution Restrictions

Ordered resolution

- define stable atom ordering;
- resolve only maximal literals

Hyper-Resolution

- Clauses are divided into
  - "nuclei": those with $\geq$ 1 negative literals
  - "electrons" : those with positive literals only
- Resolution can occur only among one nucleus and one electron
- $$Ex: \frac{\dfrac{\neg P(x) \vee \neg Q(x) \vee R(x) \quad Q(A) \vee C}{\neg P(A) \vee R(A) \vee C} \quad P(A) \vee D}{R(A) \vee C \vee D}$$
- Multiple resolution steps are merged into one step
- $$Ex: \frac{\neg P(x) \vee \neg Q(x) \vee R(x) \quad Q(A) \vee C \quad P(A) \vee D}{R(A) \vee C \vee D}$$

$\Longrightarrow$ Globally, can produce only electrons

Resolution Restrictions

Ordered resolution

- define stable atom ordering;
- resolve only maximal literals

Hyper-Resolution

- Clauses are divided into
  - "nuclei": those with ≥ 1 negative literals
  - "electrons" : those with positive literals only
- Resolution can occur only among one nucleus and one electron
-

$$Ex: \quad \frac{\dfrac{\neg P(x) \vee \neg Q(x) \vee R(x) \qquad Q(A) \vee C}{\neg P(A) \vee R(A) \vee C} \qquad P(A) \vee D}{R(A) \vee C \vee D}$$

- Multiple resolution steps are merged into one step
-

$$Ex: \quad \frac{\neg P(x) \vee \neg Q(x) \vee R(x) \qquad Q(A) \vee C \qquad P(A) \vee D}{R(A) \vee C \vee D}$$

$\Longrightarrow$ Globally, can produce only electrons

Resolution Restrictions

Ordered resolution

- define stable atom ordering;
- resolve only maximal literals

Hyper-Resolution

- Clauses are divided into
  - "nuclei": those with ≥ 1 negative literals
  - "electrons" : those with positive literals only
- Resolution can occur only among one nucleus and one electron
- $$Ex: \frac{\dfrac{\neg P(x) \vee \neg Q(x) \vee R(x) \quad Q(A) \vee C}{\neg P(A) \vee R(A) \vee C} \quad P(A) \vee D}{R(A) \vee C \vee D}$$
- Multiple resolution steps are merged into one step
- $$Ex: \frac{\neg P(x) \vee \neg Q(x) \vee R(x) \quad Q(A) \vee C \quad P(A) \vee D}{R(A) \vee C \vee D}$$

$\Longrightarrow$ Globally, can produce only electrons

Resolution Restrictions

Ordered resolution

- define stable atom ordering;
- resolve only maximal literals

Hyper-Resolution

- Clauses are divided into
  - "nuclei": those with ≥ 1 negative literals
  - "electrons" : those with positive literals only
- Resolution can occur only among one nucleus and one electron
- $$Ex: \frac{\dfrac{\neg P(x) \vee \neg Q(x) \vee R(x) \quad Q(A) \vee C}{\neg P(A) \vee R(A) \vee C} \quad P(A) \vee D}{R(A) \vee C \vee D}$$
- Multiple resolution steps are merged into one step
- $$Ex: \frac{\neg P(x) \vee \neg Q(x) \vee R(x) \quad Q(A) \vee C \quad P(A) \vee D}{R(A) \vee C \vee D}$$

$\Longrightarrow$ Globally, can produce only electrons

Resolution Restrictions

Ordered resolution

- define stable atom ordering;
- resolve only maximal literals

Hyper-Resolution

- Clauses are divided into
  - “nuclei”: those with $\geq$ 1 negative literals
  - “electrons” : those with positive literals only
- Resolution can occur only among one nucleus and one electron
-

$$Ex: \frac{\dfrac{\neg P(x) \vee \neg Q(x) \vee R(x) \quad Q(A) \vee C}{\neg P(A) \vee R(A) \vee C} \quad P(A) \vee D}{R(A) \vee C \vee D}$$

- Multiple resolution steps are merged into one step
-

$$Ex: \frac{\neg P(x) \vee \neg Q(x) \vee R(x) \quad Q(A) \vee C \quad P(A) \vee D}{R(A) \vee C \vee D}$$

$\Longrightarrow$ Globally, can produce only electrons

Resolution Restrictions

Ordered resolution

- define stable atom ordering;
- resolve only maximal literals

Hyper-Resolution

- Clauses are divided into
  - "nuclei": those with ≥ 1 negative literals
  - "electrons" : those with positive literals only
- Resolution can occur only among one nucleus and one electron
- $$Ex: \frac{\dfrac{\neg P(x) \vee \neg Q(x) \vee R(x) \quad Q(A) \vee C}{\neg P(A) \vee R(A) \vee C} \quad P(A) \vee D}{R(A) \vee C \vee D}$$
- Multiple resolution steps are merged into one step
- $$Ex: \frac{\neg P(x) \vee \neg Q(x) \vee R(x) \quad Q(A) \vee C \quad P(A) \vee D}{R(A) \vee C \vee D}$$

$\Longrightarrow$ Globally, can produce only electrons

Exercise

- Solve the example of Colonel West using Hyper-Resolution strategy
- Solve the example of Curiosity & Tuna using Hyper-Resolution Strategy

Exercise

- Solve the example of Colonel West using Hyper-Resolution strategy
- Solve the example of Curiosity & Tuna using Hyper-Resolution Strategy

Outline

Exercise

Problem

Consider the following FOL formula set Γ:

1. $\forall x.\{[\forall y.(\mathrm{Child}(y) \rightarrow \mathrm{Loves}(x, y))] \rightarrow [\exists y.\mathrm{Loves}(y, x)]\}$
2. $\forall x.[\mathrm{Child}(x) \rightarrow \mathrm{Loves}(\mathrm{Mark}, x)]$
3. $\mathrm{Beats}(\mathrm{Mark}, \mathrm{Paul}) \vee \mathrm{Beats}(\mathrm{John}, \mathrm{Paul})$
4. $\mathrm{Child}(\mathrm{Paul})$
5. $\forall x.\{[\exists z.(\mathrm{Child}(z) \wedge \mathrm{Beats}(x, z))] \rightarrow [\forall y.\neg\mathrm{Loves}(y, x)]\}$

(a) Compute the CNF-ization of Γ, Skolemize & standardize variables

(b) Write a FOL-resolution inference of the query $\mathrm{Beats}(\mathrm{John}, \mathrm{Paul})$ from the CNF-ized KB

Exercise solution

CNF-ization

(a) Compute the CNF-ization of Γ, Skolemize & standardize variables

1. $\forall x.\{[\forall y.(Child(y) \rightarrow Loves(x, y))] \rightarrow [\exists y.Loves(y, x)]\}$
   $\forall x.\{[\neg\forall y.(Child(y) \rightarrow Loves(x, y))] \vee [\exists y.Loves(y, x)]\}$
   $\forall x.\{[\exists y.(Child(y) \wedge \neg Loves(x, y))] \vee [\exists y.Loves(y, x)]\}$
   $\{[(Child(F(x)) \wedge \neg Loves(x, F(x)))] \vee [Loves(G(x), x)]\}$
   1. $Child(F(x)) \vee Loves(G(x), x)$
   2. $\neg Loves(y, F(y)) \vee Loves(G(y), y)$
2. $\neg Child(z) \vee Loves(Mark, z)$
3. $Beats(Mark, Paul) \vee Beats(John, Paul)$
4. $Child(Paul)$
5. $\forall x.\{[\exists z.(Child(z) \wedge Beats(x, z))] \rightarrow [\forall y.\neg Loves(y, x)]\}$
   $\forall x.\{[\neg\exists z.(Child(z) \wedge Beats(x, z))] \vee [\forall y.\neg Loves(y, x)]\}$
   $\forall x.\{[\forall z.(\neg Child(z) \vee \neg Beats(x, z))] \vee [\forall y.\neg Loves(y, x)]\}$
   $\neg Child(z_2) \vee \neg Beats(x_2, z_2) \vee \neg Loves(y_2, x_2)$

where $F()$, $G()$ are Skolem unary functions.

Exercise solution [cont.]

Resolution

(b) Write a FOL-resolution inference of the query Beats(John, Paul) from the CNF-ized KB:

6. [1.2, 2.] $\Longrightarrow \neg \text{Child}(F(\text{Mark})) \vee \text{Loves}(G(\text{Mark}), \text{Mark})$;
7. [1.1, 6.] $\Longrightarrow \text{Loves}(G(\text{Mark}), \text{Mark})$;
8. [4, 5.] $\Longrightarrow \neg \text{Beats}(x_2, \text{Paul}) \vee \neg \text{Loves}(y_2, x_2)$;
9. [7, 8.] $\Longrightarrow \neg \text{Beats}(\text{Mark}, \text{Paul})$;
10. [3, 9.] $\Longrightarrow \text{Beats}(\text{John}, \text{Paul})$;