# Why Logic?

- General, “language-like” representation of knowledge applicable across many domains

- Provides a concrete way to formalize the notion of an inference & “common-sense knowledge”

- Allows us to represent sentences like:

  “Every CSC UG course with ≥10 students has a TA”

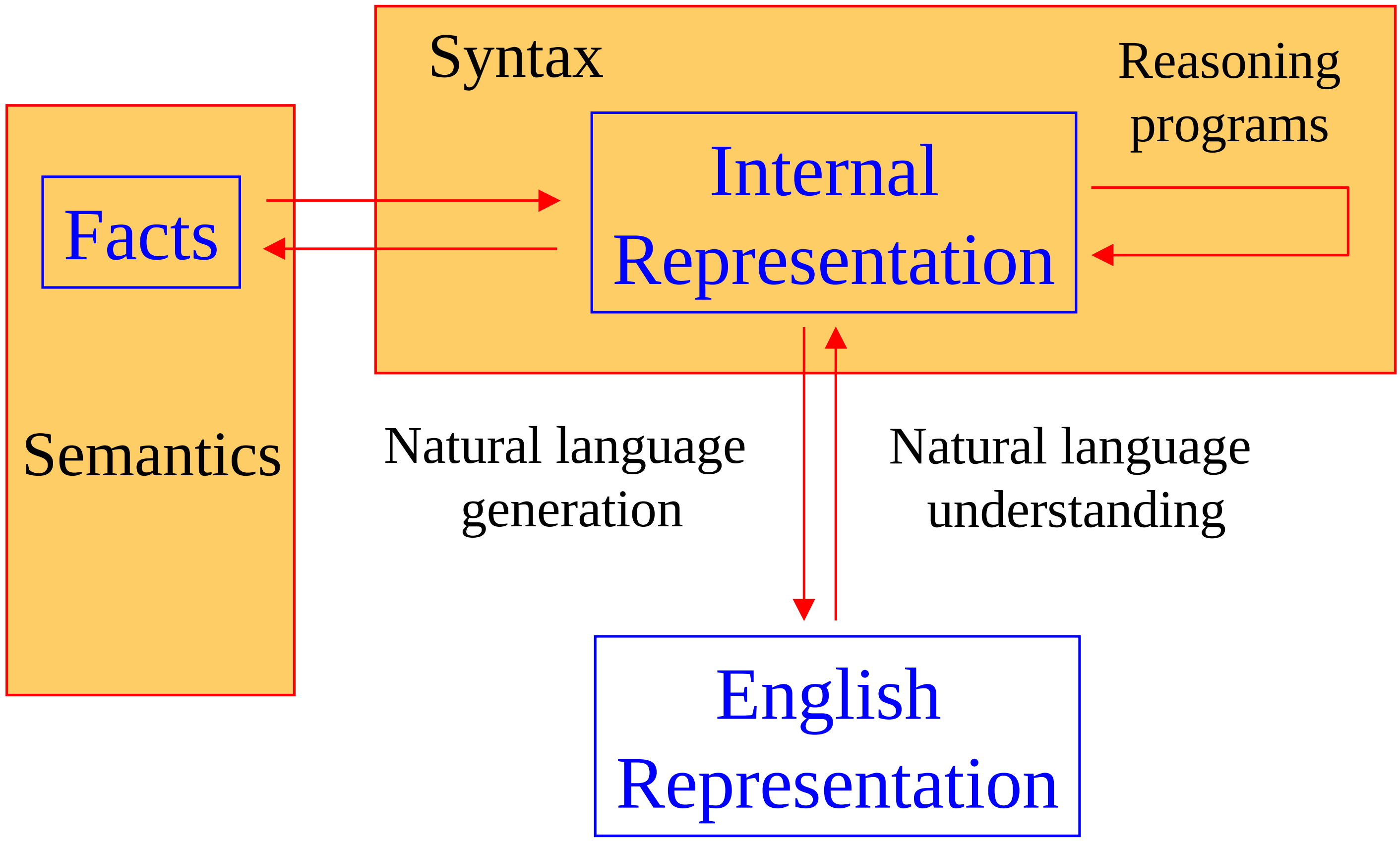
  “No man can run at 30 mph”

  “A traffic light is yellow, green, or red at any time”

  “If John doesn’t like Mary, she doesn’t like him; Mary likes John; therefore, John likes Mary”

  ...

# Facts & Representations




Propositional logic

- A string of symbols, separated by conjunctions, disjunctions & negations
- Examples:
  "Socrates is a man" $\rightarrow$
  SOCRATESMAN
  "Plato is a man" $\rightarrow$ PLATOMAN
  "Both Socrates and Plato are men" $\rightarrow$
  SOCRATESMAN $\wedge$ PLATOMAN

# First-Order Logic: Motivation

How do we represent the following sentences in propositional logic?

- Marcus was a Pompeian

Not representable in propositional logic!

- All Pompeians are Romans
- All Romans were either loyal to Caesar or hated him
- Everyone is loyal to someone
- People only try to assassinate rules they are not loyal to
- Marcus tried to assassinate Caesar
- Therefore, Marcus hated Caesar

# First-Order Logic

Instead of unanalyzed proposition symbols $P_1$, $P_2$, Q …, use predicates & terms:

Person(Mary)   Loves(Romeo, Juliet)

predicate constants
(or just "predicates")
<u>no predetermined meaning</u>

individual constants
(or just "constants")
<u>no predetermined meaning</u>

Another example:

"John gives Mary Fifi"

Give(John, Mary, Fifi) or

Give(John, Mary, Fifi, Giving-event4)
During(Giving-event4, Feb18-99)

"John will give Mary Fifi on Feb 18/99"

# First-Order Logic

More examples:

Equal(Plus(2,2), 4)

special predicate with predetermined meaning

a function with predetermined meaning

special constants with predetermined meaning

Can use infix notation:

2+2 = 4

Equal(Mother-of(Fifi), Lulu), or
Mother-of(Fifi) = Lulu

Sqrt(sin(90)) = 1

# Formulas in First-Order Logic

- Term:

  A constant (or variable, introduced later) or a function applied to an appropriate # of terms

  Examples:

  Plus(2,2), 2+2, Mary, Mother-of(Fifi), Mother-of(Mother-of(Fifi))

  ground terms (they contain no variables)

  well-formed formula

- Atomic formula (or atom, or atomic wff):

  A predicate applied to an appropriate # of terms

  Examples:

  Person(Mary), Loves(Romeo, Juliet), 2+2=4

# Formulas in First-Order Logic

- Complex formulas:

  Formed by using the connectives
  $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ as in propositional logic

  Examples:

  Eat(Fifi, Cookie1, Eat-event3) $\wedge$
  Cookie(Cookie1)

  Eat(Fifi, Cookie1, Eat-event3) $\vee$
  Eat(Lulu, Cookie1, Eat-event3)

  Eat(Fifi, Cookie1, Eat-event3) $\Rightarrow$
  Inside(Cookie1, Fifi,
  Result-state(Eat-event3))

  "If Fifi ate the cookie (in a certain eating event), then the cookie was inside Fifi at the end of the eating event"

# Quantification

Quantification allows us to express properties of entire collections of objects without referencing each of them by name

Two types of quantification:

- Existential
- Universal

Existential quantification:

$\exists x\ Dog(x)$ "there is a dog"

variable bound by $\exists$

Variables in first-order logic are placeholders for constants:

– $x$ is a free variable in $Dog(x)$

– $x$ is a bound variable in $\exists x\ Dog(x)$

# Existential Quantification (cont.)

Another example:

$\exists y\ (\text{Eat}(\text{Fifi}, y, \text{Eat-event3}) \wedge \text{Cookie}(y))$

"Fifi ate a cookie" (in a certain eating event)

Without outer brackets there is a scope ambiguity:

$\exists y\ \text{Eat}(\text{Fifi}, y, \text{Eat-event3}) \wedge \text{Cookie}(y)$

could be interpreted as

$(\exists y\ \text{Eat}(\text{Fifi}, y, \text{Eat-event3})) \wedge \text{Cookie}(y)$

variable bound by $\exists$

free variable

or as

$(\exists y\ \text{Eat}(\text{Fifi}, y, \text{Eat-event3}) \wedge \text{Cookie}(y))$

variable bound by $\exists$

# Existential Quantification (cont.)

Dot notation:

$\exists y.Eat(Fifi, y, Eat\text{-}event3) \wedge Cookie(y)$

means that $y$ has the <u>widest possible scope</u>

More examples:

$\exists y \exists e.Eat(Fifi, y, e) \wedge Cookie(y)$
$\quad \wedge During(e, Supper4)$

"Fifi ate a cookie during (a certain) supper"

$\exists x \exists y \exists e.Eat(x, y, e) \wedge Dog(x) \wedge Cookie(y)$

"A dog ate a cookie"

$\exists x.Dog(x) \wedge Owns(John, x)$

"John has a dog"

# Universal Quantification

$\forall x\ (Dog(x) \Rightarrow Animal(x))$

$\forall x. Dog(x) \Rightarrow Animal(x)$

"Every dog is an animal"

More examples:

$\forall x\ (Dog(x) \Rightarrow \exists y\ (Person(y) \wedge Owns(y,x)))$

"Every dog is owned by someone"

$(\forall x\ Has\text{-}weight(x)) \Rightarrow Has\text{-}weight(Fifi)$

"If everything has weight, then Fifi has weight"

"No whale is a fish"

$\forall x.\ Whale(x) \Rightarrow \neg Fish(x)$

$\forall x.\ Fish(x) \Rightarrow \neg Whale(x)$

$\neg \exists x.\ Whale(x) \wedge Fish(x)$

$\forall x.\ \neg(Whale(x) \wedge Fish(x))$

$\forall x.\ \neg Whale(x) \vee \neg Fish(x)$

# Quantification (cont.)

In general:

| | | |
|---|---|---|
| $\neg\exists x\ \phi$ | equiv. to | $\forall x\ \neg\phi$ |
| $\neg\forall x\ \phi$ | equiv. to | $\exists x\ \neg\phi$ |
| $\neg(\phi\vee\psi)$ | equiv. to | $\neg\phi\wedge\neg\psi$ |
| $\neg(\phi\wedge\psi)$ | equiv. to | $\neg\phi\vee\neg\psi$ |

(deMorgan's rule)

Important contrast:

"Every dog is an animal"

$\forall x.Dog(x) \Rightarrow Animal(x)$

"Some dog is a pet"

$\exists x.Dog(x) \wedge Pet(x)$

$\exists x.Dog(x) \Rightarrow Pet(x)$

"There is an <u>x</u> such that if <u>x</u> is a dog then it is a pet"

Formula is true if there is some x that is <u>not</u> a dog

# First-Order Logic: Syntax

The structure of all sentences in first-order logic is described by the following grammar:

```
Sentence → AtomicSentence
   |  Sentence Connective Sentence
   |  Quantifier Variable Sentence
   |  ¬Sentence
   |  (Sentence)

AtomicSentence → Predicate(Term,…)
   |  Term = Term

Term → Function(Term,…)
   |  Constant
   |  Variable

Connective → ∧ | ∨ | ⇒ | ⇔
Quantifier → ∃ | ∀
Constant → A | X1 | John | …
Variable → a | x | s | …
Predicate → Before | HasColor | …
Function → Mother | LeftLegOf | …
```

# Higher Order Logics

- In First-Order Logic, variables & quantifiers refer to constants (i.e., objects)

$$\exists x \ Dog(x)$$

variable
bound by $\exists$

- We can enhance the expressiveness of logic sentences by applying quantifiers to functions & relations, not just objects

$$\forall f \forall g. \ (f=g) \Leftrightarrow (\forall x \,.\, f(x)=g(x))$$

Functions of 1 variable
bound by $\forall$

"two functions are equal if and only if they have the same value for all arguments"

# More Examples

"Fifi is a pink poodle"

$$Pink(Fifi) \wedge Poodle(Fifi)$$

"Fifi is the only pink poodle"

i.e. "Fifi is a pink poodle & every pink poodle is identical to Fifi"

$$Pink(Fifi) \wedge Poodle(Fifi) \wedge \forall x.(Pink(x) \wedge Poodle(x)) \Rightarrow x=Fifi$$

"No-one likes Fifi"

i.e. "for every person x, it is not the case that x likes Fifi"

$$\forall x.Person(x) \Rightarrow \neg Likes(x, Fifi)$$

"If x eats y then y is inside x right afterwards"

$$\forall x \forall y \forall e.Eat(x,y,e) \Rightarrow Inside(y,x,Result\text{-}state(e))$$

$$\forall x \forall y \forall e \forall e'.Eat(x,y,e) \wedge Rightafter(e,e') \Rightarrow Inside(y,x,e')$$

# More Examples (cont.)

"If x is a parent of y, then x is older than y"

$$\forall x \forall y.Parent(x,y) \Rightarrow Older(x, y)$$

"If x is the mother of y, then x is a parent of y"

$$\forall x \forall y.Mother(x,y) \Rightarrow Parent(x, y)$$

"Everyone is loyal to someone"

$$\forall x \exists y.Person(x) \wedge Person(y) \wedge Loyalto(x,y)$$

or $\exists y \forall x.Person(x) \wedge Person(y) \wedge Loyalto(x,y)$ ?

"there is a person to whom everyone is loyal" $\rightarrow$ sentence is ambiguous!!

"People only try to assassinate rulers they are not loyal to"

$$\forall x \forall y.Person(x) \wedge Ruler(y) \wedge Tryassassinate(x,y) \Rightarrow \neg Loyalto(x,y)$$

or "the only thing people try to do is assassinate people they are not loyal to" $\rightarrow$ another ambiguous sentence!!

# Inference in First-Order Logic

## Premises:

1. If x is a parent of y, then x is older than y
2. If x is the mother of y, then x is a parent of y
3. Lulu is the mother of Fifi

## Conclusion:

Lulu is older than Fifi

## Mapping to first-order logic:

Premises in first-order logic:

1. $\forall x \forall y. Parent(x,y) \Rightarrow Older(x, y)$
2. $\forall x \forall y. Mother(x,y) \Rightarrow Parent(x, y)$
3. $Mother(Lulu, Fifi)$

Conclusion:

Therefore, $Older(x, y)$

Inference achieved using axioms & rules (i.e., syntactical transformations) that generalize those found in propositional logic

# Inference Rules in FOL

$$\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$$ Modus Ponens

Rule identical to the one used in propositional logic:

$$\frac{In(Mary, Garden),\ In(Mary,Garden) \Rightarrow \neg In(Mary,House)}{\neg\ In(Mary,House)}$$

$$\frac{\forall x\ \alpha}{\alpha[x/k]}$$ Universal Instantiation

$\alpha[x/k]$ : a sentence $\alpha$ with all free occurrences of variable $x$ replaced by constant $k$ (text uses $SUBST(\{x/k\}, \alpha)$.)

$$\frac{\forall x\ Has\text{-}weight(x)}{Has\text{-}weight(Fifi)}$$

Has-weight(x)[x/Fifi] is Has-weight(Fifi)

# Inference Rules in FOL (cont.)

**Premises:**

1. If x is a parent of y, then y is older than x
2. If x is the mother of y, then x is a parent of y
3. Lulu is the mother of Fifi

**Conclusion:**

Lulu is older than Fifi

Premises in first-order logic:

1. $\forall x \forall y . Parent(x,y) \Rightarrow Older(x, y)$
2. $\forall x \forall y . Mother(x,y) \Rightarrow Parent(x, y)$
3. $Mother(Lulu, Fifi)$

Proof:

1. $$\frac{\forall x \forall y . Mother(x,y) \Rightarrow Parent(x, y)}{\forall y . Mother(Lulu,y) \Rightarrow Parent(Lulu, y)}$$
2. $$\frac{\forall y . Mother(Lulu,y) \Rightarrow Parent(Lulu, y)}{Mother(Lulu,Fifi) \Rightarrow Parent(Lulu, Fifi)}$$
3. $$\frac{Mother(Lulu,Fifi), Mother(Lulu,Fifi) \Rightarrow Parent(Lulu, Fifi)}{Parent(Lulu,Fifi)}$$
4. Derive $Older(Lulu,Fifi)$ in 3 more steps

# Inference Rules in FOL (cont.)

Previous inference easier with strengthened rule:

Generalized Modus Ponens

$$\frac{\alpha_1[\underline{x}/\underline{k}],\ldots,\alpha_n[\underline{x}/\underline{k}],\ \forall x_1\ldots\forall x_m.(\alpha_1\wedge\ldots\wedge\alpha_n) \Rightarrow \beta}{\beta[\underline{x}/\underline{k}]}$$

$\underline{x}/\underline{k} : \{x_1/k_1,\ldots, x_m/k_m\}$

A proof of Older(Lulu, Fifi) using GMP:

| | |
|---|---|
| 1. Mother(Lulu, Fifi) | given |
| 2. Alive(Lulu) | given |
| 3. $\forall x \forall y.Mother(x,y) \Rightarrow Parent(x, y)$ | given |
| 4. $\forall x \forall y.(Parent(x,y) \wedge Alive(x)) \Rightarrow Older(x, y)$ | given |
| 5. Parent(Lulu, Fifi) | 1,3, GMP |
| 6. Older(Lulu, Fifi) | 5,2,4, GMP |

This use of GMP is called forward-chaining

# Inference Rules in FOL (cont.)

Another style of proof is to "reason backward" (backward-chaining):

Start with a goal (to be proved) & then derive new sub-goals until we have sub-goals known to be true

$\longrightarrow$ similar to problem reduction

Backward-chaining proof of Older(Lulu, Fifi) from premices:

1. Mother(Lulu, Fifi)
2. Alive(Lulu)
3. $\forall x \forall y.\text{Mother}(x,y) \Rightarrow \text{Parent}(x, y)$
4. $\forall x \forall y.(\text{Parent}(x,y) \wedge \text{Alive}(x)) \Rightarrow \text{Older}(x, y)$

Goal: (i) Older(Lulu, Fifi)

Match (i) against RHS of (4)

Subgoals: (ii) Parent(Lulu, Fifi) (iii) Alive(Lulu)

Match (iii) against (2) True

Match (ii) against RHS of (3)

Subgoal: (iv) Mother(Lulu, Fifi)

Match (iv) against (1)

# Inference Rules in FOL

Another style of proof is to "reason backward" (backward-chaining):

Start with a goal (to be proved) & then derive new sub-goals until we have sub-goals known to be true

$\longrightarrow$ similar to problem reduction

Backward-chaining proof of Older(Lulu, Fifi) from premices:

1. Mother(Lulu, Fifi)
2. Alive(Lulu)
3. $\forall x \forall y.\text{Mother}(x,y) \Rightarrow \text{Parent}(x, y)$
4. $\forall x \forall y.(\text{Parent}(x,y) \wedge \text{Alive}(x)) \Rightarrow \text{Older}(x, y)$

Goal: (i) Older(Lulu, Fifi)

Match (i) against RHS of (4)

Subgoals: (ii) Parent(Lulu, Fifi) (iii) Alive(Lulu)

Match (iii) against (2) True

Match (ii) against RHS of (3)

Subgoal: (iv) Mother(Lulu, Fifi)

Match (iv) against (1)

# Inference Rules in FOL

Example inference rule in First-Order Logic

Generalized Modus Ponens

$$\frac{\alpha_1[\underline{x}/\underline{k}],\ldots,\alpha_n[\underline{x}/\underline{k}],\ \forall x_1\ldots\forall x_m.(\alpha_1\wedge\ldots\wedge\alpha_n)\Rightarrow\beta}{\beta[\underline{x}/\underline{k}]}$$

$\underline{x}/\underline{k}$ :$\{x_1/k_1,\ldots,\ x_m/k_m\}$

A proof of Older(Lulu, Fifi) using GMP:

| | |
|---|---|
| 1. Mother(Lulu, Fifi) | given |
| 2. Alive(Lulu) | given |
| 3. $\forall x\forall y$.Mother(x,y) $\Rightarrow$ Parent(x, y) | given |
| 4. $\forall x\forall y$.(Parent(x,y)$\wedge$Alive(x)) $\Rightarrow$ Older(x, y) | given |
| 5. Parent(Lulu, Fifi) | 1,3, GMP |
| 6. Older(Lulu, Fifi) | 5,2,4, GMP |

This use of GMP is called forward-chaining

# Resolution: Motivation

- Steps in inferencing (e.g., forward-chaining)
    1. Define a set of inference rules
    2. Define a set of axioms
    3. Repeatedly choose one inference rule & one or more axioms (or premices) to derive new sentences until the conclusion sentence is formed

- Basic requirement:

    Rules + axioms should constitute a complete proof system

- Observation:

    Automated inferencing could be a lot more efficient & easy to implement if there was just a <u>single</u> inference rule in the proof system!

# Resolution

- Resolution (Robinson, 1965):

  A form of inference that relies on a single rule to prove the truth or falsity of logic sentences

- Because of its simplicity, efficiency & completeness properties, resolution has dominated reasoning in AI

Key characteristics:

- Resolution produces proofs by refutation:

  "To prove a statement, assume that the negation of the statement is true & try to arrive at a contradiction"

- Simplicity achieved by forcing inference rule to operate on sentences that have a very special form called Clause Normal Form (CNF)

- Completeness achieved because every logic sentence can be converted to CNF

# The Resolution Rule

Resolution relies on the following rule:

$$\frac{\neg\alpha \Rightarrow \beta,\ \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma} \qquad \text{Resolution rule}$$

equivalently,

$$\frac{\alpha \vee \beta,\ \neg\beta \vee \gamma}{\alpha \vee \gamma} \qquad \text{Resolution rule}$$

Applying the resolution rule:

1. Find two sentences that contain the same literal, once in its positive form & once in its negative form:

CNF sentences

$\boxed{\text{summer} \vee \text{winter},\ \neg\text{winter} \vee \text{cold}}$

2. Use the resolution rule to eliminate the literal from both sentences

$\boxed{\text{summer} \vee \text{cold}}$

# The Resolution Rule (cont.)

A resolution example:

parent clauses: $\boxed{\text{at-home}}$ $\boxed{\neg\text{at-home}}$

resolvent: $\square$

empty clause
(falsity, contradiction)

Another example:

$\boxed{\text{at-home} \vee \text{at-work}}$ $\boxed{\neg\text{at-home}}$

$\boxed{\text{at-work}}$

Observations:

- Resolution reduces the length of parent clauses by one literal
- Resolution applied after first converting all sentences to CNF form:
  - Disjunctions only
  - Negations of atoms only

# Resolution in Propositional Logic

Basic steps for proving a proposition S:

1. Convert all propositions in premises to CNF

| | | |
|---|---|---|
| $p$ | | $p$ |
| $(p \wedge q) \Rightarrow r$ | $\neg(p \wedge q) \vee r$ | $\neg p \vee \neg q \vee r$ |
| $(s \vee t) \Rightarrow q$ | $\neg(s \vee t) \vee q$ | $\neg s \vee q$ |
| | | $\neg t \vee q$ |
| $t$ | $t$ | $t$ |
| | | CNF |

2. Negate S & convert result to CNF

3. Add negated S to premises

4. Repeat until contradiction or no progress is made:
   a. Select 2 clauses (call them parent clauses)
   b. Resolve them together
   c. If resolvent is the empty clause, a contradiction has been found (i.e., S follows from the premises)
   d. If not, add resolvent to the premises

# Resolution in Propositional Logic

Premises:

$p$
$(p \wedge q) \Rightarrow r$
$(s \vee t) \Rightarrow q$

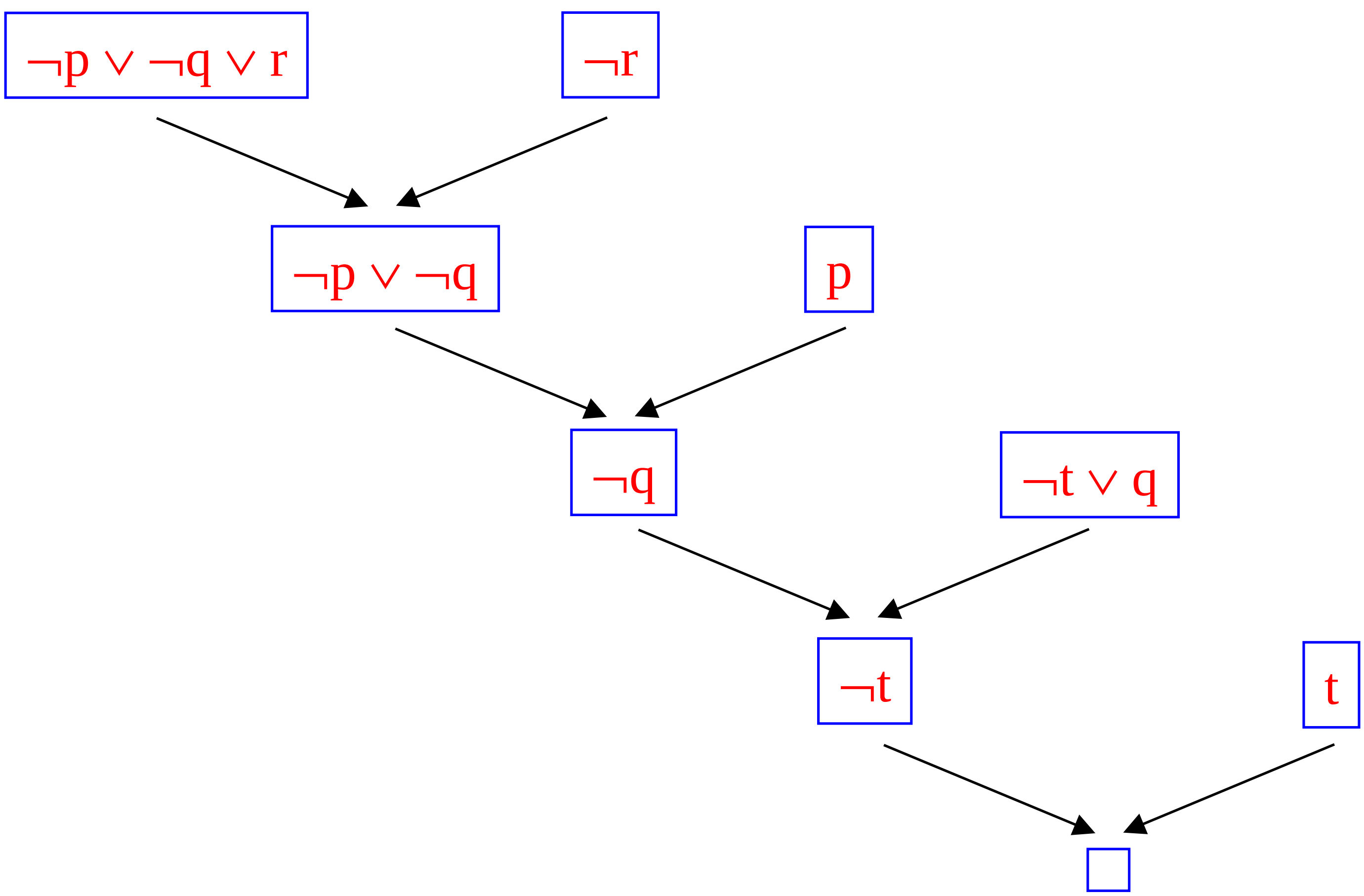
$t$

→

$p$
$\neg p \vee \neg q \vee r$
$\neg s \vee q$
$\neg t \vee q$
$t$

CNF

A resolution proof of r:



# Resolution in First-Order Logic

In propopositional logic:

$\boxed{\text{at-home} \vee \text{at-work}}$ $\boxed{\neg\text{at-home}}$

$\boxed{\text{at-work}}$

In first-order logic:

$\boxed{\forall x.\text{at-home}(x) \vee \text{at-work}(x)}$ $\boxed{\neg\text{at-home}(y)}$

$\boxed{?}$

To generalize resolution proofs to FOL
we must account for

- Predicates
- Unbound variables
- Existential & universal quantifiers

# Clause Form in First-Order Logic

- Disjunctions only
- Negations of atoms only

  $\neg P(A,B)$
- No quantifiers:
  - universal quantification implicit

    $\forall x.P(x) \quad \rightarrow \quad P(x)$
  - existential quantification replaced by Skolem constants/functions

    $\exists x.P(x) \quad \rightarrow \quad P(E)$

    $\forall y \exists x.P(x,y) \quad \rightarrow \quad P(E(y),y)$

| Ordinary FOL | | Clause Form |
|---|---|---|
| $P(A)$ | none | $P(A)$ |
| $\neg\neg Q(A,B)$ | $\neg\neg$ elimination | $Q(A,B)$ |
| $\neg(P(A) \wedge Q(B,C))$ | deMorgan | $\neg P(A) \vee \neg Q(B,C)$ |
| $\neg(P(A) \vee Q(B,C))$ | deMorgan, $\wedge$ dropping | $\boxed{\neg P(A)}, \boxed{\neg Q(B,C)}$ |

2 unit clauses

# Clause Form in First-Order Logic

| Ordinary FOL | | Clause Form |
|---|---|---|
| $P(A) \Rightarrow Q(B,C)$ | $\xrightarrow{\Rightarrow \text{ elimination}}$ | $\neg P(A) \vee Q(B,C)$ |
| $\neg(P(A) \Rightarrow Q(B,C))$ | $\xrightarrow[\text{deMorgan, } \wedge \text{ drop}]{\Rightarrow \text{ elimination}}$ | $P(A), \neg Q(B,C)$ |
| $P(A) \wedge (Q(B,C) \vee R(D))$ | $\xrightarrow{\wedge \text{ drop}}$ | $P(A), Q(B,C) \vee R(D)$ |
| $P(A) \vee (Q(B,C) \wedge R(D))$ | $\xrightarrow[\vee \text{ distribution}]{\wedge \text{ drop}}$ | $P(A) \vee Q(B,C)$, $P(A) \vee R(D)$ |
| $\forall x.P(x)$ | $\xrightarrow{\forall \text{ drop}}$ | $P(x)$ |
| $\forall x.P(x) \Rightarrow Q(x,A)$ | $\xrightarrow[\forall \text{ drop}]{\Rightarrow \text{ elimination}}$ | $\neg P(x) \vee Q(x,A)$ |
| $\exists x.P(x)$ | $\xrightarrow{\text{skolemization}}$ | $P(E)$, where E is a new constant |
| $P(A) \Rightarrow \exists x.Q(x)$ | $\xrightarrow[\text{skolemization}]{\Rightarrow \text{ elimination}}$ | $\neg P(A) \vee Q(F)$ |
| $\neg \forall x.P(x)$ <br> $\exists x.\neg P(x)$ | $\xrightarrow[\text{skolemization}]{\text{deMorgan}}$ | $\neg P(G)$ |

# Clause Form in First-Order Logic

**Ordinary FOL** | **Clause Form**

$\neg\exists x.P(x)$

$\xrightarrow{\text{deMorgan}} \forall x.\neg P(x)$

$\xrightarrow[\forall\text{ drop}]{} \neg P(G)$

$\neg(\exists x.P(x) \wedge \forall x.Q(x))$

$\xrightarrow{\text{variable rename}} \neg(\exists x.P(x) \wedge \forall y.Q(y))$

$\xrightarrow{\text{deMorgan}} \neg\exists x.P(x) \vee \neg\forall y.Q(y)$

$\xrightarrow{\text{deMorgan}} \forall x.\neg P(x) \vee \exists y.\neg Q(y)$

$\xrightarrow[\text{skolemization}]{\forall\text{ drop}} \neg P(x) \vee \neg Q(H)$

$\forall x\exists y.P(x,y)$

$\xrightarrow{\text{fun. skolemization}} \forall x.P(x,K(x))$

$\xrightarrow{\forall\text{ drop}} P(x,K(x))$

$\forall x\forall y\exists z.P(x,y,z) \xrightarrow[\forall\text{ drop}]{\text{skolemization}} P(x,y,L(x,y))$

# Clause Form in First-Order Logic

Ordinary FOL | Clause Form

$\forall x.P(x) \Rightarrow \exists y.Q(x,y) \xrightarrow[\text{skol., } \forall \text{ drop}]{\Rightarrow \text{ elimination}} \neg P(x) \vee Q(x,M(x))$

$(\forall x.P(x)) \Rightarrow \exists y.P(y)$

$\xrightarrow{\Rightarrow \text{ elimination}} (\neg\forall x.P(x)) \vee \exists y.P(y)$

$\xrightarrow{\text{deMorgan}} \exists x.\neg P(x) \vee \exists y.P(y)$

$\xrightarrow[\text{skolemization}]{} \neg P(N) \vee P(O)$

# Conversion to Clause Form

Steps in general case:

1. Rename all variables so that all quantifiers bind distinct variables
2. $\Rightarrow$-elimination
3. deMorgan ($\neg\vee$, $\neg \wedge$, $\neg\forall$, $\neg\exists$)
4. Skolemization ($\exists$-elimination)
5. $\forall$-dropping
6. $\vee$-distribution
7. $\wedge$-dropping

# Resolution in First-Order Logic

In propopositional logic:

$\boxed{\text{at-home} \vee \text{at-work}}$ $\boxed{\neg\text{at-home}}$

$\boxed{\text{at-work}}$

In first-order logic:

To generalize resolution proofs to FOL
we must account for

- Predicates
- Unbound variables
- Existential & universal quantifiers

Idea: First convert sentences to clause form
Then unify variables

UNIFY

$\boxed{\forall x.\boxed{\text{at-home}(x) \vee \text{at-work}(x)}}$ $\boxed{\neg\text{at-home}(y)}$

$\boxed{?}$

# Resolution in First-Order Logic

Resolution examples:

at-home(x) ∨ at-work(x)    ¬at-home(Mary)

Unifier: {x/Mary}

at-work(Mary)

Parent(x,y) ⇒ Older(x,y)    Parent(Lulu,Fifi)

{x/Lulu,y/Fifi}

Older(Lulu,Fifi)

Loves(x,Mother-of(x)) ∨ Psychotic(x)

¬Loves(Bill,y) ∨ Sends-flowers(Bill,y)

{x/Bill}
{y/Mother-of(Bill)}

Psychotic(Bill) ∨ Sends-flowers(Bill,Mother-of(Bill))

# Resolution Steps

Resolution steps for 2 clauses containing

P(arg.list1), ¬P(arg.list2)

1. Make the variables in the 2 clauses distinct

2. Find the "most general unifier" of arg.list1 & arg.list2:

   go through the lists "in parallel," making substitutions for variables only, so as to make the 2 lists the same

3. Make the substitutions corresponding to the m.g.u. throughout both clauses

4. The resolvent is the clause consisting of all the resulting literals except P & ¬P

# Unification

Can we unify the following clauses?

| | | |
|---|---|---|
| Loves(x,x)<br>¬Loves(Bill,Paula) | {x/Bill}?? | NO--Can't substitute Bill for Paula or vice-versa |
| Loves(Mother-of(x),x)<br>¬Loves(Father-of(y),y) | | NO--Can't make Mother-of(x) & Father-of(x) the same by substituting for a <u>variable</u> |
| Loves(Mother-of(x),x)<br>¬Loves(Mother-of(Bill),Bill) | {x/Bill} | YES! |
| Loves(y,Mother-of(y))<br>¬Loves(x,x) | {x/y}?? | NO--Can't substitute Mother-of(y) for y (would give infinite regress) |

# Resolution in First-Order Logic

In propositional logic:

$\boxed{\text{at-home} \vee \text{at-work}}$ $\boxed{\neg\text{at-home}}$

$\boxed{\text{at-work}}$

In first-order logic:

To generalize resolution proofs to FOL
we must account for

- Predicates
- Unbound variables
- Existential & universal quantifiers

Idea: First convert sentences to clause form
Then unify variables

UNIFY

$\boxed{\forall x.\text{at-home}(x) \vee \text{at-work}(x)}$ $\boxed{\neg\text{at-home}(y)}$

$\boxed{\text{at-work}(x)}$

# Resolution in First-Order Logic

Basic steps for proving a conclusion S given premises

$\text{Premise}_1, \ldots, \text{Premise}_n$

(all expressed in FOL):

1. Convert all sentences to CNF
2. Negate conclusion S & convert result to CNF
3. Add negated conclusion S to the premise clauses
4. Repeat until contradiction or no progress is made:
    a. Select 2 clauses (call them parent clauses)
    b. Resolve them together, performing all required unifications
    c. If resolvent is the empty clause, a contradiction has been found (i.e., S follows from the premises)
    d. If not, add resolvent to the premises

If we succeed in Step 4, we have proved the conclusion

# Resolution Examples

Example 1:

- If something is intelligent, it has common sense
- Deep Blue does not have common sense
- Prove that Deep Blue is not intelligent

1. $\forall x. I(x) \Rightarrow H(x)$
2. $\neg H(D)$

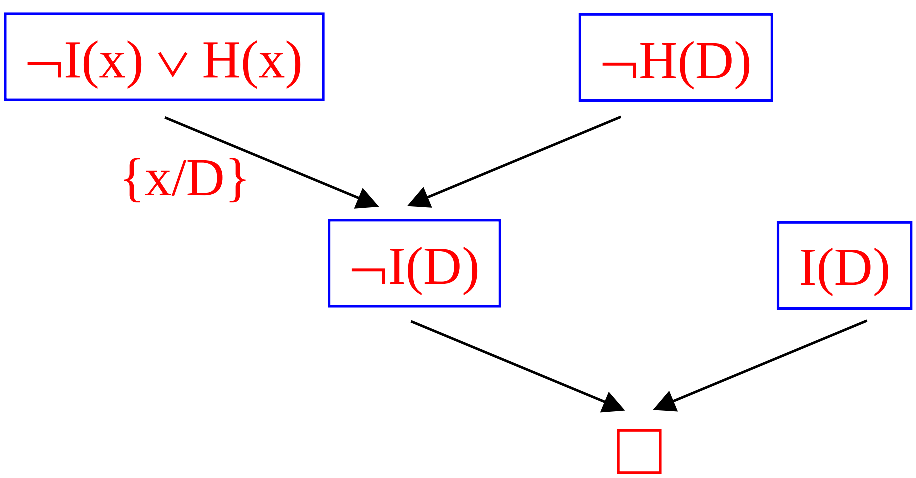
Conclusion: $\neg I(D)$
Denial: C3: $I(D)$

$\longrightarrow$

C1: $\neg I(x) \vee H(x)$
C2: $\neg H(D)$

CNF

A resolution proof of $\neg I(D)$:




Proof also written as:

2nd literal, 1st clause

C4: $\neg I(D)$ r[C1b, C2]

C5: $\square$ r[C3, C4]

# Resolution Examples (cont.)

Example 2:

Premises:

Mother(Lulu, Fifi)

Alive(Lulu)

$\forall x\ \forall y.\text{Mother}(x,y) \Rightarrow \text{Parent}(x,y)$

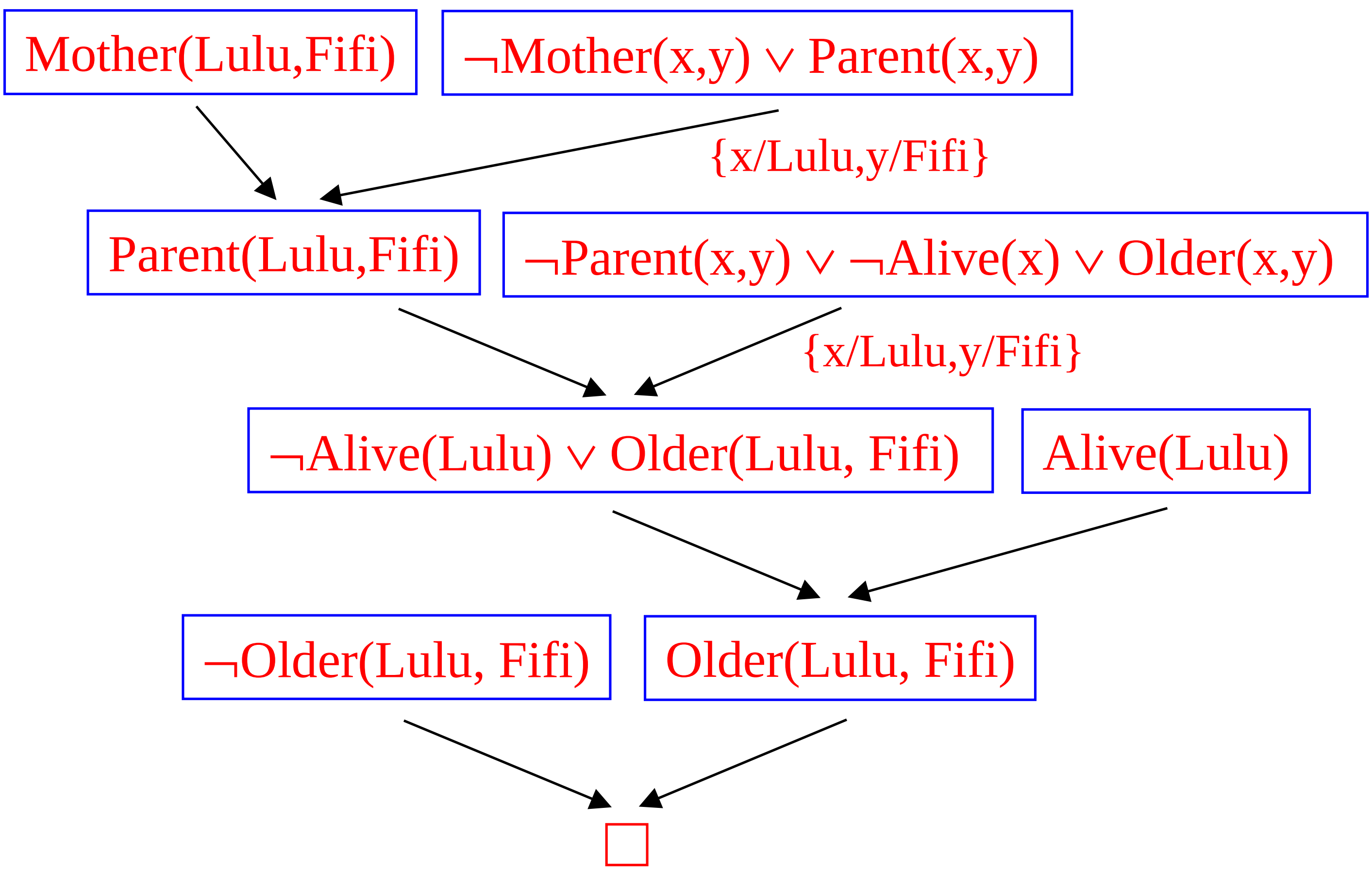
$\forall x\ \forall y.(\text{Parent}(x,y) \wedge \text{Alive}(x)) \Rightarrow \text{Older}(x,y)$

Prove:

Older(Lulu, Fifi)

Denial:

$\neg$Older(Lulu, Fifi)



# Resolution Examples (cont.)

Could also have written the proof as:

| | | |
|---|---|---|
| C1. | Mother(Lulu,Fifi) | given |
| C2. | Alive(Lulu) | given |
| C3. | $\neg$Mother(x,y) $\vee$ Parent(x,y) | given |
| C4. | $\neg$Parent(x,y) $\vee$ $\neg$Alive(x) $\vee$ Older(x,y) | given |
| C5. | $\neg$Older(Lulu, Fifi) | denial of concl. |
| C6. | Parent(Lulu,Fifi) | r[C1,C3a] |
| C7. | $\neg$Alive(Lulu) $\vee$ Older(Lulu, Fifi) | r[C6,C4a] |
| C8. | Older(Lulu, Fifi) | r[C8,C5] |
| C9. | $\square$ | |

Proof consists of 4 resolution steps: longer than the proof with GMP
because we can only resolve 2 clauses at once using this form of resolution

# Resolution Examples (cont.)

Example 3:

- Suppose the desired conclusion had been "Something is older than Fifi"
  ∃x.Older(x, Fifi)
- Denial:
  ¬∃x.Older(x, Fifi)
  also written as: ∀x.¬Older(x, Fifi)
  in clause form: ¬Older(x, Fifi)
- Last proof step would have been

C8: Older(Lulu,Fifi)    C5': ¬Older(x, Fifi)

{x/Lulu}

□

Don't make mistake of <u>first</u> forming clause from conclusion & <u>then</u> denying it:

- Conclusion:
  ∃x.Older(x, Fifi)
  clause form: Older(C, Fifi)
  denial: ¬Older(C, Fifi)

Cannot unify Lulu,C!!

# Resolution for Question-Answering

- So far, resolution was used to just prove logic sentences
- Resolution's unification mechanism allows us to answer questions as well:
  - Consider again the proof of "Something is older than Fifi"
    $\exists x.Older(x, Fifi)$
  - Denial clause:
    $\neg Older(x, Fifi)$
  - Substitution made in disproof:
    $\{x/Lulu\}$
  - So Lulu is the "something" that's older than Fifi.
    $\rightarrow$Answers question "what is older than Fifi?"

In general, to answer
"what x has such-and-such properties?"

- Prove "there exists an $x$ with such-and-such properties"
- Extract substitution for $x$

# Question-Answering

Example 1:

"Who is Lulu older than?"

- Prove that

  "there is an x such that Lulu is older than x"
- In FOL form:

  $\exists x.Older(Lulu, x)$
- Denial:

  $\neg\exists x.Older(Lulu, x)$

  $\forall x.\neg Older(Lulu, x)$

  in clause form: $\neg Older(Lulu, x)$
- Successful proof gives

  $\{x/Fifi\}$ [Verify!!]

Example 2:

"What is older than what?"

- In FOL form:

  $\exists x\ \exists y.Older(x, y)$
- Denial:

  $\neg\exists x\exists y.Older(x, y)$

  in clause form: $\neg Older(x, y)$
- Successful proof gives

  $\{x/Lulu, y/Fifi\}$ [Verify!!]

# Getting Multiple Answers

- Assume additional facts:

  Father(BowWow, Fifi)

  $\neg$Father(x, y) $\vee$ Parent(x,y)

  Alive(BowWow)

- We can then answer $\exists$x.Older(x, Fifi) using

  {x/Lulu} or {x/BowWow}

  (i.e., 2 distinct proofs exist)

Q: Is it possible to find all answers to a given question using the resolution rule?

Ans: Yes, if the premises in the knowledge base are all Horn clauses

$$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \vee B$$

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \Rightarrow B$$

Achieved by finding all ways to <u>refute</u> a query

# Getting Multiple Answers (cont.)

To find all ways of refuting $\neg$Older(x, Fifi):

- Find unit clauses this resolves with (if any), adding substitutions for successful refutations to **`Answers`**

  If Older(Fang, Fifi) was in KB, we would have
  **`Answers = Answers ∪ {x/Fang}`**

- Find clauses of the form

  $$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \vee \text{Older}(x, \text{Fifi})$$

  and resolve

- If successful, with unifier $\theta$, recursively find all refutations of the corresponding antecedent instances ($\neg A_1, \neg A_2, \ldots, \neg A_n$)

- “Compose” the substitutions for these refutations with $\theta$ and add to **`Answers`**

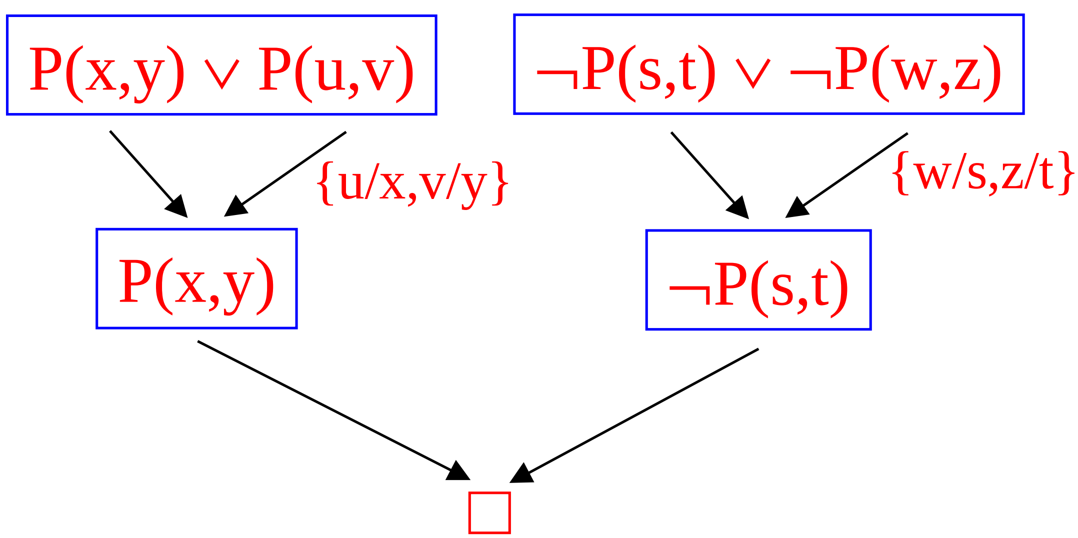
# Factoring

- Resolution is “not quite” refutation-complete

  e.g. $P(x,y) \vee P(u,v)$ and $\neg P(s,t) \vee \neg P(w,z)$ are clearly contradictory, yet we can’t derive $\Box$

- Factoring:

  Allows us to unify 2 literals of the same clause



# Equality

- Suppose we are given:

  Older(Lulu, Fifi)

  ¬Older(x,x)

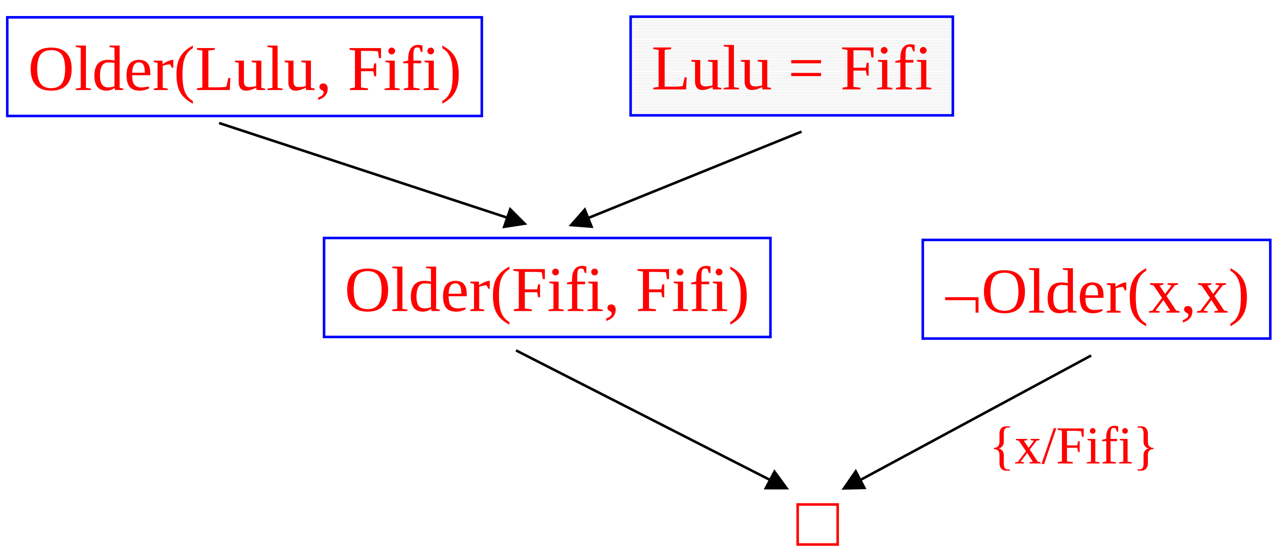
  resolution cannot be applied here

- Now, what if we know that Lulu & Fifi refer to the same entity?

  Need an additional rule & axioms to treat equality

  Paramodulation: essentially, substitution of equals (but with unification)

- Proving ¬(Lulu = Fifi):



# Resolution Strategies

In a general KB, there may be many resolutions that can be applied at a given step

| | | |
|---|---|---|
| C1. | Mother(Lulu,Fifi) | given |
| C2. | Alive(Lulu) | given |
| C3. | ¬Mother(x,y) ∨ Parent(x,y) | given |
| C4. | ¬Parent(x,y) ∨ ¬Alive(x) ∨ Older(x,y) | given |
| C5. | ¬Older(Lulu, Fifi) | denial of concl. |
| C6. | Parent(Lulu,Fifi) | r[C1,C3a] |
| C7. | ¬Alive(Lulu) ∨ Older(Lulu, Fifi) | r[C6,C4a] |
| C8. | Older(Lulu, Fifi) | r[C8,C5] |
| C9. | □ | |

We can use specific resolution strategies to ensure that we do not perform “useless” resolutions

# Resolution Strategies

- Backward chaining strategy:

  Reason backwards from a goal
  (used for finding multiple answers to a query)

- Unit resolution:

  One of the parent clauses is always chosen to contain a single literal

$$\boxed{\text{at-home}(x) \lor \text{at-work}(x)} \qquad \boxed{\neg\text{at-home}(y)}$$

$$\boxed{\text{at-work}(x)}$$

Idea: Length of resolvent always decreases by 1
$\rightarrow$ gets closer to empty clause
(i.e., unit resolution is a Greedy method)

Caveat: Unit resolution is not complete!

# Resolution Strategies (cont.)

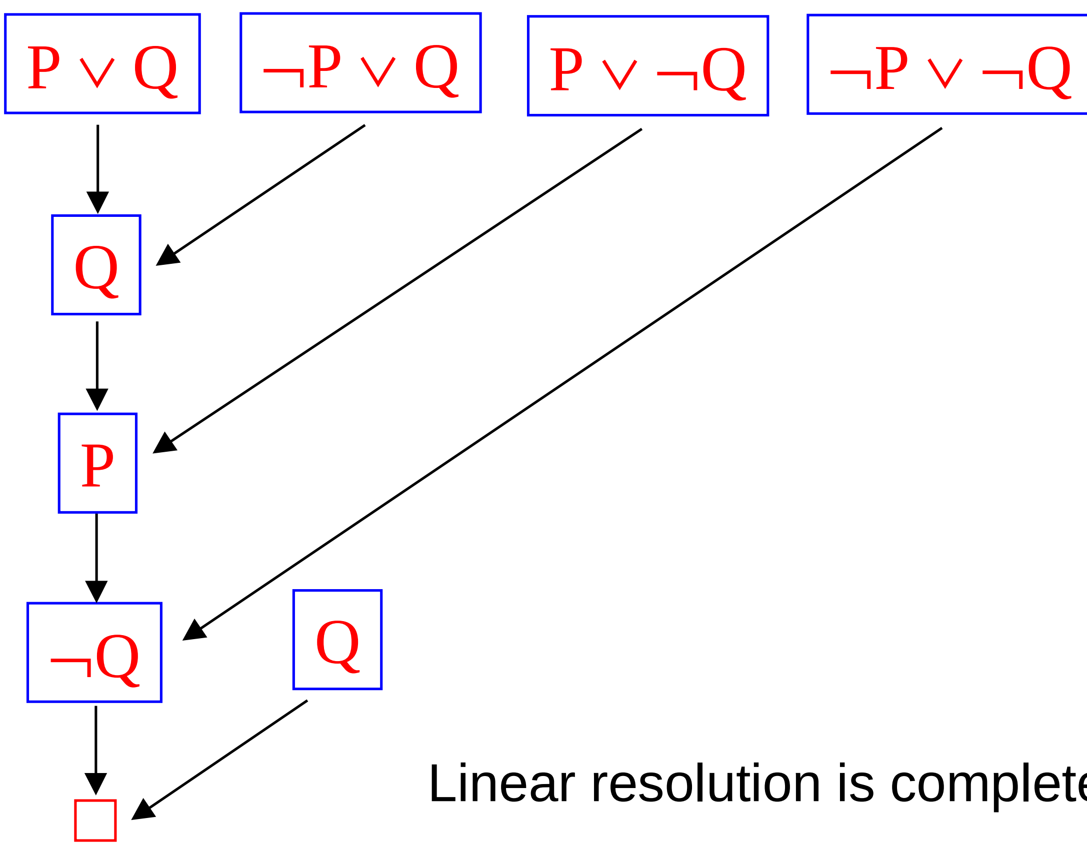
- Input resolution:

One of the parent clauses is contained in the original KB

Input resolution is equivalent to unit resolution (and hence also incomplete)

- Linear resolution:

Each parent is a linear resolvent, i.e., is either in the initial KB or is an ansestor of the other parent




Linear resolution is complete

# Resolution Strategies (cont.)

- Set-of-support resolution:

  Given a set of clauses $\Gamma$, a set of support resolvent of $\Gamma$ is a resolvent whose parents are either clauses of $\Gamma$ or descendants of such clauses

  Set-of-support resolution: always use a denial clause or a descendant of a denial clause as one parent

  Idea: “Focus” the proof on using the denial clause(s) to derive a contradiction rather than grinding arbitrary KB facts together

# Logic Programming

Robert Kowalski's equation:

$\boxed{\text{Programming = Logic + Control}}$

In logic programming, algorithms are created by augmenting logical sentences with information to control the inference process (Russell & Norvig)

An FOL definition of the list member function:

$\forall x \forall l.\text{Member}(x, [x|l])$
$\forall x \forall y \forall l.\text{Member}(x, l) \Rightarrow \text{Member}(x, [y|l])$

Logic programming can be thought of as a "declarative language"

Program = sequence declarations

Control = implicit

Program execution = proof

e.g., prove member(3, [2,1,3])

# Programming in Prolog

- Developed in the early 70's

- It is the most popular logic programming language (in Europe, was even more popular than Lisp)

- It is an interpreted language

- Prolog programs are:
  - sequences of logical sentences
  - only Horn clauses are allowed:

    $Member(x, l) \Rightarrow Member(x, [y|l])$
  - terms can be constant symbols, variables, functional terms
  - syntactically distinct terms assumed to be distinct objects
    (e.g., $A$ cannot be unified with $F(x)$)
  - Uses "negation-as-failure" operator:

    $not\ P$ is considered true if language fails to prove $P$

# Programming in Prolog (cont.)

- Syntax:

  $\forall x \forall l. Member(x, [x|l])$

  written as

  ```
  > member(X, [X | L])
  ```

  $\forall x \forall y \forall l. Member(x, l) \Rightarrow Member(x, [y|l])$

  written as

  ```
  > member(X, [Y | L]):-
    member(X, L)
  ```

- To check whether 1 is a member of list [1,2,3] we simply type

  ```
  > member(1,[1,2,3])
  ```

- We can enumerate all members of [1,2,3] by typing

  ```
  > member(X,[1,2,3])
  X=1 (press Enter)
  X=2 etc
  ```