

Inference in first-order logic

Chapter 9, Sections 1–6

Dieter Fox

Proofs

Sound inference: find α such that $KB \models \alpha$.
Proof process is a **search**, operators are inference rules.

E.g., Modus Ponens (MP)

$$\frac{\alpha, \quad \alpha \Rightarrow \beta}{\beta} \quad \frac{At(Joe, UW) \quad At(Joe, UW) \Rightarrow OK(Joe)}{OK(Joe)}$$

E.g., And-Introduction (AI)

$$\frac{\frac{\alpha \quad \beta}{\alpha \wedge \beta} \quad \frac{OK(Joe) \quad CSMajor(Joe)}{OK(Joe) \wedge CSMajor(Joe)}}{OK(Joe) \wedge CSMajor(Joe)}$$

E.g., Universal Elimination (UE)

$$\frac{\frac{\forall x \quad \alpha}{\alpha\{x/\tau\}} \quad \frac{\forall x \quad At(x, UW) \Rightarrow OK(x)}{At(Pat, UW) \Rightarrow OK(Pat)}}{\alpha\{x/\tau\} \quad At(Pat, UW) \Rightarrow OK(Pat)}$$

τ must be a ground term (i.e., no variables)

- ◇ Proofs
- ◇ Unification
- ◇ Generalized Modus Ponens
- ◇ Forward and backward chaining
- ◇ Completeness
- ◇ Resolution

Example proof

Bob is a buffalo	1.	$Buffalo(Bob)$
Pat is a pig	2.	$Pig(Pat)$
Buffaloes outrun pigs	3.	$\forall x, y \quad Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
Bob outruns Pat		

Example proof

Bob is a buffalo	1. $Buf\ falo(Bob)$
Pat is a pig	2. $Pig(Pat)$
Buffaloes outrun pigs	3. $\forall x,y\ Buf\ falo(x) \wedge Pig(y) \Rightarrow Faster(x,y)$
Bob outruns Pat	
AI 1 & 2	4. $Buf\ falo(Bob) \wedge Pig(Pat)$

Example proof

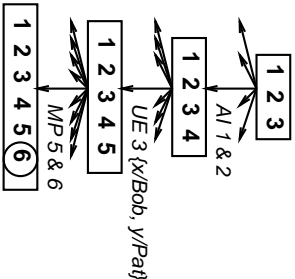
Bob is a buffalo	1. $Buf\ falo(Bob)$
UE 3, $\{x/Bob,y/Pat\}$	2. $Pig(Pat)$
Buffaloes outrun pigs	3. $\forall x,y\ Buf\ falo(x) \wedge Pig(y) \Rightarrow Faster(x,y)$
Bob outruns Pat	
AI 1 & 2	4. $Buf\ falo(Bob) \wedge Pig(Pat)$
UE 3, $\{x/Bob,y/Pat\}$	5. $Buf\ falo(Bob) \wedge Pig(Pat) \Rightarrow Faster(Bob,Pat)$
MP 6 & 7	6. $Faster(Bob,Pat)$

Example proof

Bob is a buffalo	1. $Buf\ falo(Bob)$
Pat is a pig	2. $Pig(Pat)$
Buffaloes outrun pigs	3. $\forall x,y\ Buf\ falo(x) \wedge Pig(y) \Rightarrow Faster(x,y)$
Bob outruns Pat	
AI 1 & 2	4. $Buf\ falo(Bob) \wedge Pig(Pat)$
UE 3, $\{x/Bob,y/Pat\}$	5. $Buf\ falo(Bob) \wedge Pig(Pat) \Rightarrow Faster(Bob,Pat)$

Search with primitive inference rules

Operators are inference rules
States are sets of sentences
Goal test checks state to see if it contains query sentence



AI, UE, MP is a common inference pattern

Problem: branching factor huge, esp. for UE

Idea: find a substitution that makes the rule premise match some known facts
⇒ a single, more powerful inference rule

Unification

p	q	σ
$Knows(John, x)$	$Knows(John, Jane)$	
$Knows(John, x)$	$Knows(y, OJ)$	
$Knows(John, x)$	$Knows(y, Mother(y))$	

p	q	σ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, OJ)$	
$Knows(John, x)$	$Knows(y, Mother(y))$	

Unification

p	q	σ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, OJ)$	
$Knows(John, x)$	$Knows(y, Mother(y))$	

p	q	σ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, OJ)$	
$Knows(John, x)$	$Knows(y, Mother(y))$	

Unification

Unification

p	q	σ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, OJ)$	$\{x/John, y/OJ\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$

Idea: Unify rule premises with known facts, apply unifier to conclusion
 E.g., if we know q and
 then we conclude

$Knows(John, x) \Rightarrow Likes(John, x)$
 $Likes(John, Jane)$
 $Likes(John, OJ)$
 $Likes(John, Mother(John))$

Soundness of GMP

Need to show that

$$p_1', \dots, p_n', (p_1 \wedge \dots \wedge p_n \Rightarrow q) \models q\sigma$$

provided that $p_i'\sigma = p_i\sigma$ for all i

Lemma: For any definite clause p , we have $p \models p\sigma$ by UE

- $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \models (p_1 \wedge \dots \wedge p_n \Rightarrow q)\sigma = (p_1\sigma \wedge \dots \wedge p_n\sigma \Rightarrow q\sigma)$
- $p_1', \dots, p_n' \models p_1' \wedge \dots \wedge p_n' \models p_1'\sigma \wedge \dots \wedge p_n'\sigma$
- From 1 and 2, $q\sigma$ follows by simple MP

Generalized Modus Ponens (GMP)

$$\frac{p_1', p_2', \dots, p_n', (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{q\sigma}$$

where $p_i'\sigma = p_i\sigma$ for all i

$$\begin{aligned} \text{E.g. } p_1' &= \text{Faster}(\text{Bob}, \text{Pat}) \\ p_2' &= \text{Faster}(\text{Pat}, \text{Steve}) \\ p_1 \wedge p_2 &\Rightarrow q = \text{Faster}(x, y) \wedge \text{Faster}(y, z) \Rightarrow \text{Faster}(x, z) \\ \sigma &= \{x/\text{Bob}, y/\text{Pat}, z/\text{Steve}\} \\ q\sigma &= \text{Faster}(\text{Bob}, \text{Steve}) \end{aligned}$$

GMP used with KB of **definite clauses** (exactly one positive literal):
 either a single atomic sentence or
 (conjunction of atomic sentences) \Rightarrow (atomic sentence)
 All variables assumed universally quantified

Forward chaining

When a new fact p is added to the KB
 for each rule such that p unifies with a premise
 if the other premises are **known**
 then add the conclusion to the KB and continue chaining

Forward chaining is **data-driven**
 e.g., inferring properties and categories from percepts

Forward chaining example

Add facts 1, 2, 3, 4, 5, 7 in turn.

Number in [] = unification literal; \surd indicates rule firing

1. $Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
2. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
3. $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$
4. $Buffalo(Bob)$

Forward chaining example

Add facts 1, 2, 3, 4, 5, 7 in turn.

Number in [] = unification literal; \surd indicates rule firing

1. $Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
2. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
3. $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$
4. $Buffalo(Bob)$ [1a, \times]

Forward chaining example

Add facts 1, 2, 3, 4, 5, 7 in turn.

Number in [] = unification literal; \surd indicates rule firing

1. $Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
2. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
3. $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$
4. $Buffalo(Bob)$ [1a, \times]
5. $Pig(Pat)$

Forward chaining example

Add facts 1, 2, 3, 4, 5, 7 in turn.

Number in [] = unification literal; \surd indicates rule firing

1. $Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
2. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
3. $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$
4. $Buffalo(Bob)$ [1a, \times]
5. $Pig(Pat)$ [1b, \surd]

Forward chaining example

Add facts 1, 2, 3, 4, 5, 7 in turn.

Number in [] = unification literal; \surd indicates rule firing

1. $Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
2. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
3. $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$
4. $Buffalo(Bob)$ [1a, \times]
5. $Pig(Pat)$ [1b, \surd] \rightarrow 6. $Faster(Bob, Pat)$ [3a, \times], [3b, \times]

Forward chaining example

Add facts 1, 2, 3, 4, 5, 7 in turn.

Number in [] = unification literal; \surd indicates rule firing

1. $Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
2. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
3. $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$
4. $Buffalo(Bob)$ [1a, \times]
5. $Pig(Pat)$ [1b, \surd] \rightarrow 6. $Faster(Bob, Pat)$ [3a, \times], [3b, \times]
5. $Pig(Pat)$ [2a, \times]

Forward chaining example

Add facts 1, 2, 3, 4, 5, 7 in turn.

Number in [] = unification literal; \surd indicates rule firing

1. $Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
2. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
3. $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$
4. $Buffalo(Bob)$ [1a, \times]
5. $Pig(Pat)$ [1b, \surd] \rightarrow 6. $Faster(Bob, Pat)$ [3a, \times], [3b, \times]
5. $Pig(Pat)$ [2a, \times]
7. $Slug(Steve)$

Forward chaining example

Add facts 1, 2, 3, 4, 5, 7 in turn.

Number in [] = unification literal; \surd indicates rule firing

1. $Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
2. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
3. $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$
4. $Buffalo(Bob)$ [1a, \times]
5. $Pig(Pat)$ [1b, \surd] \rightarrow 6. $Faster(Bob, Pat)$ [3a, \times], [3b, \times]
5. $Pig(Pat)$ [2a, \times]
7. $Slug(Steve)$ [2b, \surd]

Forward chaining example

Add facts 1, 2, 3, 4, 5, 7 in turn.

Number in [] = unification literal; \surd indicates rule firing

1. $Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
2. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
3. $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$
4. $Buffalo(Bob)$ [1a, \times]
5. $Pig(Pat)$ [1b, \surd] \rightarrow 6. $Faster(Bob, Pat)$ [3a, \times], [3b, \times]
5. $Pig(Pat)$ [2a, \times]
7. $Slug(Steve)$ [2b, \surd]
 \rightarrow 8. $Faster(Pat, Steve)$

Forward chaining example

Add facts 1, 2, 3, 4, 5, 7 in turn.

Number in [] = unification literal; \surd indicates rule firing

1. $Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
2. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
3. $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$
4. $Buffalo(Bob)$ [1a, \times]
5. $Pig(Pat)$ [1b, \surd] \rightarrow 6. $Faster(Bob, Pat)$ [3a, \times], [3b, \times]
5. $Pig(Pat)$ [2a, \times]
7. $Slug(Steve)$ [2b, \surd]
 \rightarrow 8. $Faster(Pat, Steve)$ [3a, \times], [3b, \surd]
 \rightarrow 9. $Faster(Bob, Steve)$ [3a, \times], [3b, \times]

Forward chaining example

Add facts 1, 2, 3, 4, 5, 7 in turn.

Number in [] = unification literal; \surd indicates rule firing

1. $Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
2. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
3. $Faster(x, y) \wedge Faster(y, z) \Rightarrow Faster(x, z)$
4. $Buffalo(Bob)$ [1a, \times]
5. $Pig(Pat)$ [1b, \surd] \rightarrow 6. $Faster(Bob, Pat)$ [3a, \times], [3b, \times]
5. $Pig(Pat)$ [2a, \times]
7. $Slug(Steve)$ [2b, \surd]
 \rightarrow 8. $Faster(Pat, Steve)$ [3a, \times], [3b, \surd]

Backward chaining

When a query q is asked
if a matching fact q' is known, return the unifier
for each rule whose consequent q' matches q
attempt to prove each premise of the rule by backward chaining

(Some added complications in keeping track of the unifiers)

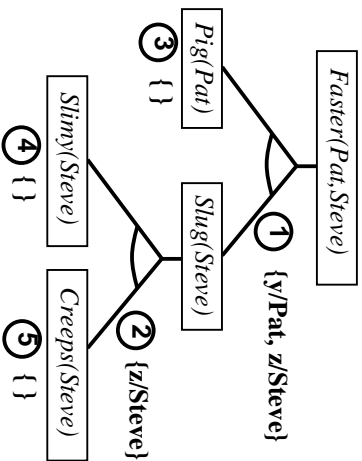
(More complications help to avoid infinite loops)

Two versions: find **any** solution, find **all** solutions

Backward chaining is the basis for **logic programming**, e.g., Prolog

Backward chaining example

1. $Pig(y) \wedge Slug(z) \Rightarrow Faster(y, z)$
2. $Slimy(z) \wedge Creeps(z) \Rightarrow Slug(z)$
3. $Pig(Pat)$
4. $Slimy(Steve)$
5. $Creeps(Steve)$



Resolution

Entailment in first-order logic is only **semidecidable**:

can find a proof of α if $KB \models \alpha$

cannot always prove that $KB \not\models \alpha$

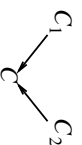
Cf. Halting Problem: proof procedure may be about to terminate with success or failure, or may go on for ever

Resolution is a **refutation** procedure:

to prove $KB \models \alpha$, show that $KB \wedge \neg\alpha$ is unsatisfiable

Resolution uses $KB, \neg\alpha$ in CNF (conjunction of clauses)

Resolution inference rule combines two clauses to make a new one:



Inference continues until an **empty clause** is derived (contradiction)

Completeness in FOL

Procedure i is complete if and only if

$$KB \vdash_i \alpha \quad \text{whenever} \quad KB \models \alpha$$

Forward and backward chaining are **complete for Horn KBs** but incomplete for general first-order logic

E.g., from

$$\begin{aligned} PhD(x) &\Rightarrow HighlyQualified(x) \\ \neg PhD(x) &\Rightarrow EarlyEarnings(x) \\ HighlyQualified(x) &\Rightarrow Rich(x) \\ EarlyEarnings(x) &\Rightarrow Rich(x) \end{aligned}$$

should be able to infer $Rich(Me)$, but FC/BC won't do it

Does a complete algorithm exist?

Resolution inference rule

Basic propositional version:

$$\frac{\alpha \vee \beta, \quad \neg\beta \vee \gamma}{\alpha \vee \gamma} \quad \text{or equivalently} \quad \frac{\neg\alpha \Rightarrow \beta, \quad \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

Full first-order version:

$$\frac{p_1 \vee \dots \vee p_j \quad \dots \vee p_m, \quad q_1 \vee \dots \vee q_k \quad \dots \vee q_n}{(p_1 \vee \dots \vee p_{j-1} \vee p_{j+1} \quad \dots \vee p_m \vee q_1 \quad \dots \vee q_{k-1} \vee q_{k+1} \quad \dots \vee q_n)\sigma}$$

where $p_j\sigma = \neg q_k\sigma$

For example,

$$\frac{\neg Rich(x) \vee Unhappy(x), \quad Rich(Me)}{Unhappy(Me)}$$

with $\sigma = \{x/Me\}$

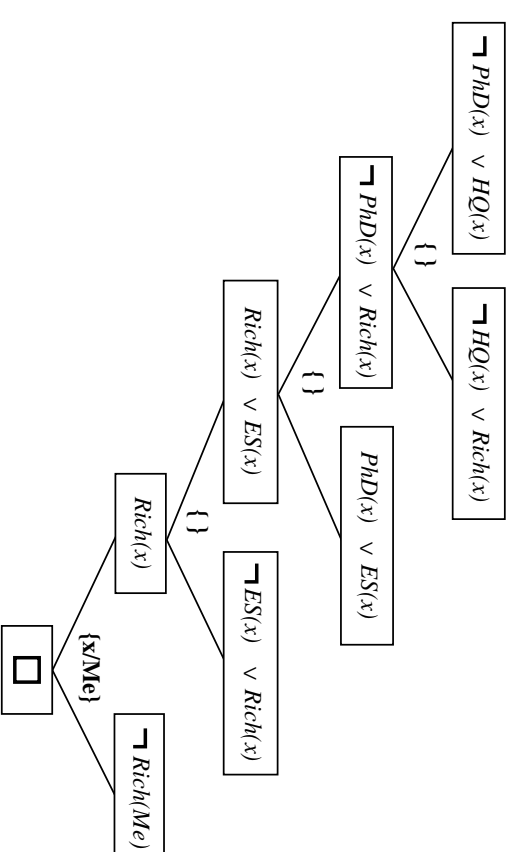
Resolution proof

To prove α :

- negate it
- convert to CNF
- add to CNF KB
- infer contradiction

E.g., to prove $Rich(me)$, add $\neg Rich(me)$ to the CNF KB

- $\neg PhD(x) \vee HighlyQualified(x)$
- $PhD(x) \vee EarlyEarnings(x)$
- $\neg HighlyQualified(x) \vee Rich(x)$
- $\neg EarlyEarnings(x) \vee Rich(x)$



Resolution proof

Resolution in practice

Resolution is complete and usually necessary for mathematics

Automated theorem provers are starting to be useful to mathematicians and have proved several new theorems