Industrial-strength inference

Chapter 9.5–6, Chapters 8.1 and 10.2–3

Outline

◇ Completeness

◇ Resolution

◇ Logic programming

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 <u>complete for Horn KBs</u>
but incomplete for general first-order logic

E.g., from

$$\begin{array}{l} PhD(x) \;\Rightarrow\; HighlyQualified(x) \\ \neg PhD(x) \;\Rightarrow\; EarlyEarnings(x) \\ HighlyQualified(x) \;\Rightarrow\; Rich(x) \\ EarlyEarnings(x) \;\Rightarrow\; Rich(x) \end{array}$$

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

Does a complete algorithm exist?

A brief history of reasoning

450B.C.	Stoics	propositional logic, inference (maybe)
322B.C.	Aristotle	"syllogisms" (inference rules), quantifiers
1565	Cardano	probability theory (propositional logic + uncertainty)
1847	Boole	propositional logic (again)
1879	Frege	first-order logic
1922	Wittgenstein	proof by truth tables
1930	Gödel	$\exists$ complete algorithm for FOL
1930	Herbrand	complete algorithm for FOL (reduce to propositional)
1931	Gödel	$\neg\exists$ complete algorithm for arithmetic
1960	Davis/Putnam	"practical" algorithm for propositional logic
1965	Robinson	"practical" algorithm for FOL—resolution

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:

$$\begin{tikzcd} C_1 \arrow[dr] & & C_2 \arrow[dl] \\ & C & \end{tikzcd}$$

Inference continues until an empty clause is derived (contradiction)

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:

$$\frac{\begin{array}{c} p_1 \vee \ldots\ p_j\ \ldots \vee p_m, \\ q_1 \vee \ldots\ q_k\ \ldots \vee q_n \end{array}}{(p_1 \vee \ldots\ p_{j-1} \vee p_{j+1}\ \ldots p_m \vee q_1 \ldots\ q_{k-1} \vee q_{k+1}\ \ldots \vee q_n)\sigma}$$

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

For example,

$$\frac{\begin{array}{l} \neg Rich(x) \vee Unhappy(x) \\ Rich(Me) \end{array}}{Unhappy(Me)}$$

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

Conjunctive Normal Form

Literal = (possibly negated) atomic sentence, e.g., $\neg Rich(Me)$

Clause = disjunction of literals, e.g., $\neg Rich(Me) \vee Unhappy(Me)$

The KB is a conjunction of clauses

Any FOL KB can be converted to CNF as follows:

1. Replace $P \Rightarrow Q$ by $\neg P \vee Q$
2. Move $\neg$ inwards, e.g., $\neg \forall x\, P$ becomes $\exists x\, \neg P$
3. Standardize variables apart, e.g., $\forall x\, P \vee \exists x\, Q$ becomes $\forall x\, P \vee \exists y\, Q$
4. Move quantifiers left in order, e.g., $\forall x\, P \vee \exists x\, Q$ becomes $\forall x \exists y\, P \vee Q$
5. Eliminate $\exists$ by Skolemization (next slide)
6. Drop universal quantifiers
7. Distribute $\wedge$ over $\vee$, e.g., $(P \wedge Q) \vee R$ becomes $(P \vee Q) \wedge (P \vee R)$

Skolemization

$\exists x\, Rich(x)$ becomes $Rich(G1)$ where $G1$ is a new "Skolem constant"

$\exists\, k\;\; \frac{d}{dy}(k^y) = k^y$ becomes $\frac{d}{dy}(e^y) = e^y$

More tricky when $\exists$ is inside $\forall$

E.g., "Everyone has a heart"

$$\forall\, x\;\; Person(x) \;\Rightarrow\; \exists\, y\;\; Heart(y) \land Has(x, y)$$

Incorrect:

$$\forall\, x\;\; Person(x) \;\Rightarrow\; Heart(H1) \land Has(x, H1)$$

Correct:

$$\forall\, x\;\; Person(x) \;\Rightarrow\; Heart(H(x)) \land Has(x, H(x))$$

where H is a new symbol ("Skolem function")

Skolem function arguments: all enclosing universally quantified variables

Resolution proof

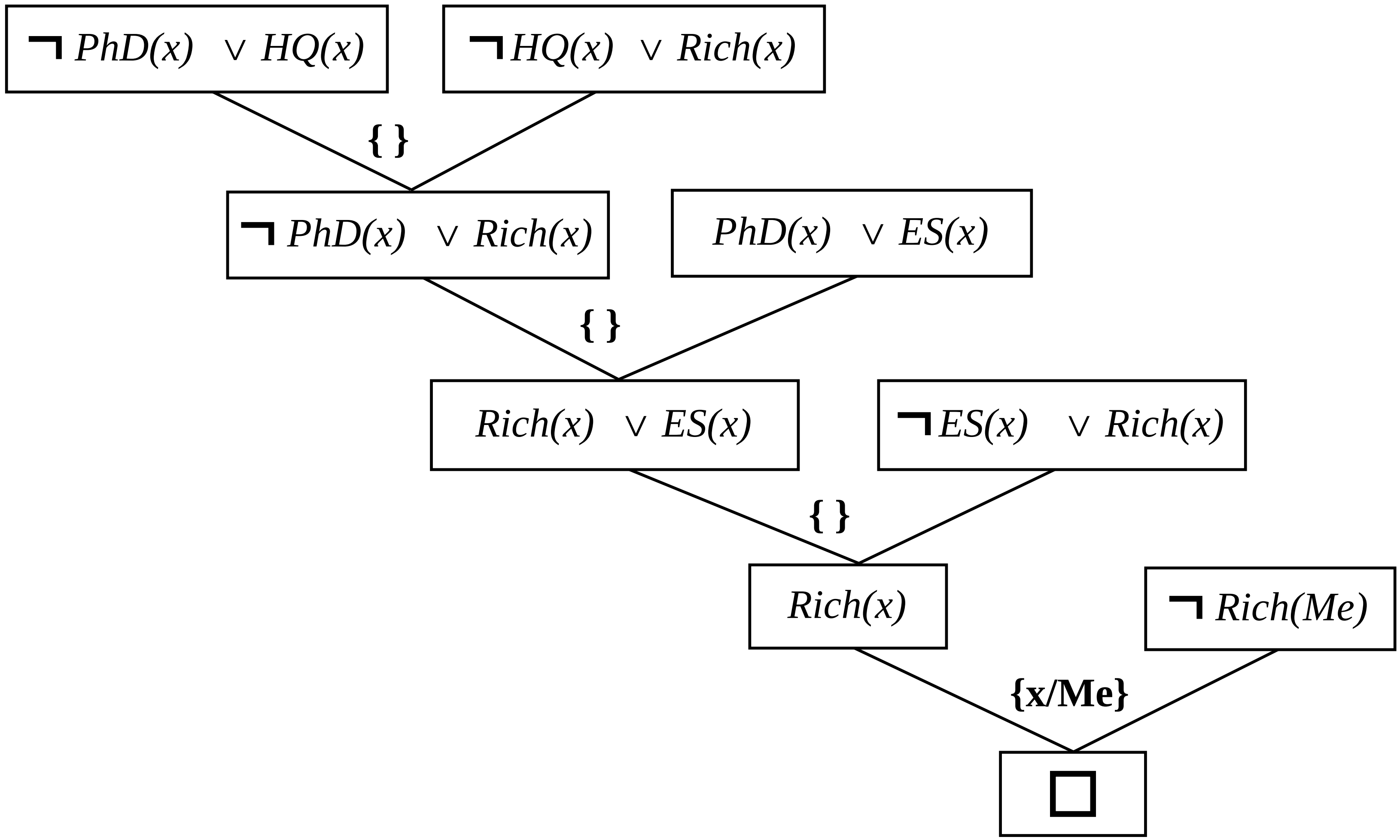
To prove α:

– 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

$$\neg PhD(x) \lor HighlyQualified(x)$$
$$PhD(x) \lor EarlyEarnings(x)$$
$$\neg HighlyQualified(x) \lor Rich(x)$$
$$\neg EarlyEarnings(x) \lor Rich(x)$$

Resolution proof



Logic programming

Sound bite: computation as inference on logical KBs

	Logic programming	Ordinary programming
1.	Identify problem	Identify problem
2.	Assemble information	Assemble information
3.	Tea break	Figure out solution
4.	Encode information in KB	Program solution
5.	Encode problem instance as facts	Encode problem instance as data
6.	Ask queries	Apply program to data
7.	Find false facts	Debug procedural errors

Should be easier to debug $Capital(NewYork, US)$ than $x := x + 2$!

Prolog systems

Basis: backward chaining with Horn clauses + bells & whistles
Widely used in Europe, Japan (basis of 5th Generation project)
Compilation techniques ⇒ 10 million LIPS

Program = set of clauses = `head :- literal`$_1$`, ... literal`$_n$`.`
Efficient unification by <u>open coding</u>
Efficient retrieval of matching clauses by direct linking
Depth-first, left-to-right backward chaining
Built-in predicates for arithmetic etc., e.g., `X is Y*Z+3`
Closed-world assumption ("negation as failure")
  e.g., `not PhD(X)` succeeds if `PhD(X)` fails

Prolog examples

Depth-first search from a start state `X`:

```
dfs(X) :- goal(X).
dfs(X) :- successor(X,S),dfs(S).
```

No need to loop over `S`: `successor` succeeds for each

Appending two lists to produce a third:

```
append([],Y,Y).
append([X|L],Y,[X|Z]) :- append(L,Y,Z).
```

```
query:   append(A,B,[1,2]) ?
answers: A=[]    B=[1,2]
         A=[1]   B=[2]
         A=[1,2] B=[]
```