Topics in Artiflcial Intelligence

CS424 | Fall 1999

Lecture #4/5

Gregory Dudek

Proof Theory

Purely syntactic rules for deriving the logical consequences of a set of sentences.

We write: $KB \vdash \alpha$, i.e., α can be **deduced** from KB or α is **provable** from KB.

Key property:

Both in propositional and in flrst-order logic we have a proof theory (\calculus") such that:

$\vdash$ **and** $\models$ **are equivalent.**

Proof Theory

If $KB \vdash \alpha$ implies $KB \models \alpha$, we say the proof theory is **sound**.

If $KB \models \alpha$ implies $KB \vdash \alpha$, we say the proof theory is **complete**.

Why so remarkable / important?

Soundness and Completeness

Allows computer to ignore semantics and

\just push symbols"!

In propositional logic, truth tables cumbersome (at least).

In flrst-order, models can be inflnite!

Proof theory: One or more **inference rules** with

zero or more axioms/tautologies

to \get things going.").

Example Proof Theory

One rule of inference: **Modus Ponens**

From α and $\alpha \Rightarrow \beta$ it follows that β.

Semantic soundness easily verifled. (truth table)

Axiom schemas: **not wffs, but a way to generate axioms**

(As. I) $\alpha \Rightarrow (\beta \Rightarrow \alpha)$

(As. II) $((\alpha \Rightarrow (\beta \Rightarrow \gamma)) \Rightarrow ((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \beta)))$.

(As. III) $(\neg\alpha \Rightarrow \beta) \Rightarrow (\neg\alpha \Rightarrow \neg\beta) \Rightarrow \alpha$.

Note: α, β, γ stand for arbitrary sentences. So,
inflnite collection of axioms.

Now, α can be **deduced** from a set of sentences '

if there exists a sequence of applications of

modus ponens

that leads from ' to α (possibly using the axioms).

One can prove that:

Modus ponens with the above axioms will generate exactly

all (and only those) statements logically **entailed** by '.

So, we have a way of generating entailed statements

in a purely syntactic manner!

(Sequence is called a proof. Finding it can be hard . . .)

Example Proof

Lemma. For any α, we have $\vdash (\alpha \Rightarrow \alpha)$.

Proof.

$(\alpha \Rightarrow (\alpha \Rightarrow \alpha) \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$, (Ax. II)

$\alpha \Rightarrow (\alpha \Rightarrow \alpha) \Rightarrow \alpha$, (Ax. I)

$(\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$, (M. P.)

$\alpha \Rightarrow \alpha \Rightarrow \alpha)$ (Ax. I)

$\alpha \Rightarrow \alpha$ (M.P.)

Alternative: more e−cient using resolution.

Example Proof Theory

One rule of inference: **Modus Ponens**

From α and $\alpha \Rightarrow \beta$ it follows that β.

Semantic soundness easily verifled. (truth table)

Axiom schemas:

(Ax. I) $\alpha \Rightarrow (\beta \Rightarrow \alpha)$

(Ax. II) $((\alpha \Rightarrow (\beta \Rightarrow \gamma)) \Rightarrow ((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \gamma)))$.

(Ax. III) $(\neg\alpha \Rightarrow \beta) \Rightarrow (\neg\alpha \Rightarrow \neg\beta) \Rightarrow \alpha$.

Note: α, β, γ stand for arbitrary sentences. So, inflnite collection of axioms.

Now, α can be **deduced** from a set of sentences '

ifi there exists a sequence of applications of **modus ponens** that leads from ' to α (possibly using the axioms).

One can prove that:

Modus ponens with the above axioms will generate exactly all (and only those) statements logically **entailed** by '.

So, we have a way of generating entailed statements

in a purely syntactic manner!

(Sequence is called a proof. Finding it can be hard ...)

Example Proof

Lemma. 1) For any α, we have $\vdash (\alpha \Rightarrow \alpha)$.

Proof.

$(\alpha \Rightarrow (\alpha \Rightarrow \alpha) \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$, (Ax. II)

$\alpha \Rightarrow (\alpha \Rightarrow \alpha) \Rightarrow \alpha$, (Ax. I)

$(\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$, (M. P.)

$\alpha \Rightarrow \alpha \Rightarrow \alpha)$ (Ax. I)

$\alpha \Rightarrow \alpha$ (M.P.)

Another Example Proof

Lemma. 2) For any α and β, we have $\beta, \neg\beta \vdash \alpha$.

Proof.

$(\neg\alpha \Rightarrow \beta) \Rightarrow (\neg\alpha \Rightarrow \neg\beta) \vdash \alpha$, (Ax. III)

β, (hyp.)

$\beta \Rightarrow \neg\alpha \Rightarrow \beta$, (Ax. I)

$\neg\alpha \Rightarrow \beta$, (M.P.)

$(\neg\alpha \Rightarrow \neg\beta) \Rightarrow \alpha$, (M.P.)

$\neg\beta$, (hyp.)

$\neg\beta \Rightarrow \neg\alpha \Rightarrow \neg\beta$, (Ax. I)

$\neg\alpha \Rightarrow \neg\beta$, (M.P.)

α (M.P.)

Key Properties

We have the following properties (also for flrst-order logic):

the following three conditions are equivalent:

(I) $\Gamma \models \alpha$

(II) $\Gamma \vdash \alpha$

(III) $\Gamma, \neg\alpha$ is inconsistent (can be refuted).

(I) is semantic; (II) syntactic, and (III) at high-level semantic

but we have a nice syntactic automatic procedure procedure: resolution.

What common proof techinque does III represent?

Resolution

First need canonical form: \clausal".

Conjunction of disjunctions (clauses) / CNF

Ex.: $\neg(P \Rightarrow Q) \vee (R \Rightarrow P)$.

$\neg(\neg P \vee Q) \vee (\neg R \vee P)$

$(P \wedge \neg Q) \vee (\neg R \vee P)$ (Morgan's law)

$(P \vee \neg R \vee P) \wedge (\neg Q \vee \neg R \vee P)$ (assoc. and distr. laws)

$(P \vee \neg R) \wedge (\neg Q \vee \neg R \vee P)$

$\{(P \vee \neg R), (\neg Q \vee \neg R \vee P)\}$,

What can you say about the length of the CNF?

Given a CNF, a **single** inference rule (and no axioms) will allow us to determine inconsistency.

So, using property III (above) and resolution, we have a sound and complete proof procedure for propositional logic
(can be extended to flrst-order).

The Resolution Rule (clausal form)

We saw it (on blackboard) last class.

From $\alpha \vee p$ and $\neg p \vee \beta$, we can derive:

$\alpha \vee \beta$ (α and β are disjunctions of literals (literal = prop. vars or its negation).

: $\neg\alpha \Rightarrow p$ and $p \Rightarrow \beta$

gives

$\neg\alpha \Rightarrow \beta$.

It's a \chaining rule."

We can derive the empty clause via resolution ifi

the set of clauses is inconsistent.

Method relies on property III. It's refutation complete.

Note that method does not generate theorems from scratch.

E.g. we have $P \wedge R \models (P \vee R)$, but we can't get $(P \vee R)$ from $\{\{P\}, \{R\}\}$.

But, given $\{\{P\}, \{R\}\}$ and the negation of $P \vee R$, we get the set $\{\{P\}, \{R\}, \{\neg P\}, \{\neg R\}\}$. Resolving on this set gives empty clause. Thus contradiction. Thus proof.

May seem cumbersome but note that can be easily automated. Just \smash" clauses till empty clause or no more new clauses.

Guaranteed sound and (refutation) complete.

Q. Why is method with axioms more di–cult to implement?

What about length of resolution proof?

Consider Pigeon-Hole (PH) problem: Formula encodes that you cannot place n + 1 pigeons in n holes (one per hole).

Cook / Karp around 1971/72. Resolved by Armin Haken 1985

Related to NP vs. $co - NP$ questions.

PH takes **exponentially** many steps! (no matter in what order.)

PH hidden in many practical problems. Makes thm. proving expensive. Partly, led to recent move to model-based methods (NP-complete).

Pigeon-Hole Principle

$P_{i,j}$ for Pigeon i in hole j.

$P_{1,1} \vee P_{1,2} \vee P_{1,3} \ldots P_{1,n}$

$P_{2,1} \vee P_{2,2} \vee P_{2,3} \ldots P_{2,n}$

$\ldots$

$P_{(n+1),1} \vee P_{(n+2),2} \vee P_{(n+3),3} \ldots P_{(n+1),n}$

and ??

$(\neg P_{1,1} \vee \neg P_{1,2})$, $(\neg P_{1,1} \vee \neg P_{1,3})$, $(\neg P_{1,1} \vee \neg P_{1,4})$

$\ldots$

$(\neg P_{1,(n-1)} \vee \neg P_{1,n})$,

$(\neg P_{2,1} \vee \neg P_{2,2}) \ldots (\neg P_{2,(n-1)} \vee \neg P_{2,n})$

etc.

$(\neg P_{1,1} \vee \neg P_{2,1})$, $(\neg P_{1,1} \vee \neg P_{3,1})$, $\ldots$

$(\neg P_{1,2} \vee \neg P_{2,2})$, $(\neg P_{1,2} \vee \neg P_{3,2})$, etc.

Resolution proof of inconsistency requires at least an exponential number of clauses, no matter in what order how you resolve things!

\Method can't count."

A More Concise Formulation

$\forall x \exists y (x \in Pigeons)(y \in Holes) IN(x, y)$

$\forall x \forall x' \forall y (IN(x, y) \wedge IN(x', y) \ldots$??

$\forall x \forall y \forall y' (IN(x, y) \wedge IN(x, y') \ldots$??

$Pigeons = \{p_1, p_2, \ldots p_{n+1}\}$,

$Holes = \{p_1, p_2, \ldots p_n\}$.

We have first-order logic with some set-theory notation.

Notation only.

Alternatively, we can state for $x \in Pigeons$ as ??

Q. Any easier to determine inconsistency?

Basic idea: axiomatic / knowledge-based / declarative approach

allows us a very large range of queries / conclusions.

Procedural (e.g., standard program, \simulation") would

require separate procedure for each possible question (almost).

Of course, getting the axioms / facts right can be tricky!

And, reasoning can be computationally very hard!

Again, \what's meant by embodying knowledge about the world?"

Example:

1) $On(A, Fl) \Rightarrow Clear(B)$

2) $(Clear(B) \wedge Clear(C)) \Rightarrow On(A, Fl)$

3) $Clear(B) \vee Clear(A)$

4) $Clear(B)$

5) $Clear(C)$

One interpretation:

U is the set { `A, B, C, Floor` }.

1) mapping constant symbols to elements of U.

e.g., A to `A`, B to `B`, C to `C`

and Fl to `Floor`

Could we have mapped Fl to `A`??

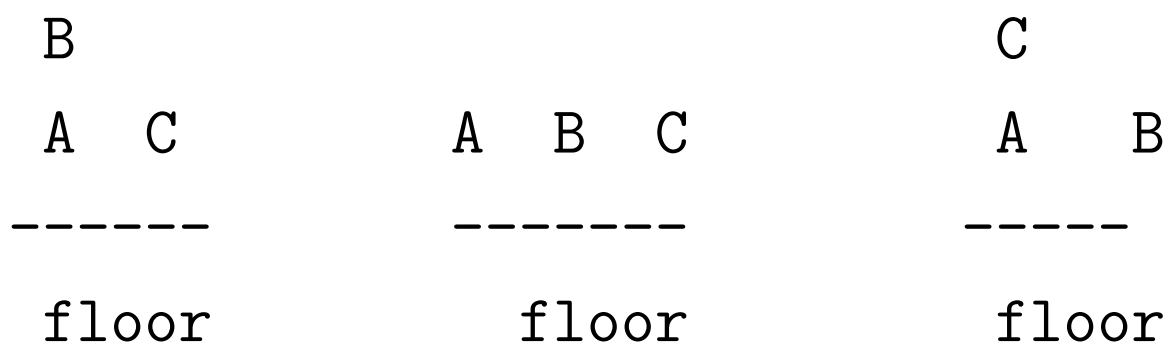
2) mapping of relation symbol On to relation on U.

e.g., On = { `[ B, A ], [ A, Floor ], [C, Floor ]` }.

3) mapping of relation (property) $Clear$ to a unary rel. on U.

e.g., $Clear$ = { `[ B ], [ C ]` }.

Yet others . . .



Including completely difierent interpretations!

E.g., use integers for domain. (Lowenheim 1915)

Try to add su–cient axioms (facts) to rule out unwanted models. E.g., add $clear(A)$.

Terms | - a logical expressions that refers to

an object. Constant symbols are terms. Functions applied to constant symbols. $FatherOf(John)$. Also, **variables** are terms (later) and functions applied to variables or other terms.

The interepretation is given by whatever the Constant or Function maps to in U (vars later).

If no vars, called **atomic terms**.

Atomic sentences: predicate symbol applied to atomic terms.

E.g. $Married(FatherOf(Richard), MotherOf(John))$

Evaluated to **true** if predicate symbol holds between the objects referred to by the arguments.

Complex sentences | add **logical connectives**.

E.g. $Older(John, 30) \Rightarrow Older(Jane, 29)$

Quantifiers

Universal Quantification $\forall$ |

E.g., $\forall x \;\; Cat(x) \Rightarrow (x)$

Think of as:

$(Cat(Spot) \Rightarrow Mammal(Spot)) \;\; \wedge$

$(Cat(Felix) \Rightarrow Mammal(Felix)) \;\; \wedge$

$(Cat(John) \Rightarrow Mammal(John)) \;\; \wedge$

$\ldots$

Intuition: Expand over all object symbols.

Existential Quantiflcation $\exists$ |

E.g., $\exists x \;\; Sister(x, Spot) \wedge Cat(x)$

Think of as:

$(Sister(Spot, Spot) \wedge Cat(Spot)) \quad \vee$

$(Sister(Rebecca, Spot) \wedge Cat(Spot)) \quad \vee$

$(Sister(Felix, Spot) \wedge Cat(Spot)) \quad \vee$

$\ldots$

Intuition: Expand over all object symbols.

Equality = |

E.g. $father(John) = Henry$

True ifi refer to same object of U in interpretation.

(identity relation)

See reference materials for more discussion and flne details.

E.g. can't switch quantiflers around.

Compare $\forall x \exists y Loves(x, y)$ vs.

Compare $\exists x \forall y Loves(x, y)$

Graph Coloring

Graph: N nodes, K colors.

- 1) $\forall i \ \ (1 \leq i \leq N) \ \ \exists j \ \ (1 \leq j \leq K) \ \ Color(i, j)$

  $\forall i, j, l \ \ (1 \leq i \leq N) \ \ (1 \leq j, l \leq K)$

  $[(Color(i, j) \wedge Color(i, l)) \Rightarrow (j = l)]$
- 2) $\forall i, j \ \ (1 \leq i, j \leq N) \ \ [(i \neq j) \Rightarrow$

  $(Edge(i, j) \Rightarrow$

  $[\neg \exists \ \ k (1 \leq k \leq K) \ \ (Color(i, k) \wedge Color(j, k))])]$

alternative:

- 3) $\forall i, j \;\; (1 \leq i, j \leq N) \;\; [(i \neq j) \Rightarrow$
  $(Edge(i, j) \Rightarrow [\forall \;\; k(1 \leq k \leq K) \;\; (\neg Color(i, k) \vee \neg Color(j, k))])]$

Now actual graph given by, e.g.,:

- 4) $Edge(1, 3), Edge(2, 4), Edge(5, 6) \ldots$ etc.

reasoning: 3 & 4 gives e.g.:

$$\forall\ k(1 \leq k \leq K)\ \ (\neg Color(1,k) \vee \neg Color(3,k))$$

uses \unifIcation" $\{i/1, j/3\}$with Modus Ponens (p. 269 R&N).

For K = 5, we get:

$$(\neg Color(1,1) \vee \neg Color(3,1)), (\neg Color(1,2) \vee \neg Color(3,2)),$$

$$\ldots (\neg Color(1,5) \vee \neg Color(3,5))$$

in propositional form.

uses Universal Elimination, e.g., substitute $\{k/1\}$, etc.

So far, we've considered various flrst-order formalizations.

How do we reason with them? Derive new info?

A. Use **resolution** as in propositional case

From $(\alpha \vee p) \wedge (\neg p \vee \beta)$

conclude $\alpha \vee \beta$ until you reach contradiction.

Need some extra \tricks" to deal with quantiflers

and variables.

Example

Jack owns a dog.

Every dog owner is an animal lover.

No animal lover kills an animal.

Either Jack or Curiosity killed the cat, who is named Tuna.

Did Curiosity kill the cat?

Original Sentences (Plus Background Knowledge)

1. $\exists x : Dog(x) \wedge Owns(Jack, x)$
2. $\forall x(\exists y Dog(y) \wedge Owns(x, y)) \rightarrow AnimalLover(x)$
3. $\forall x AnimalLover(x) \rightarrow \forall y Animal(y) \rightarrow \neg Kills(x, y)$
4. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
5. $Cat(Tuna)$
6. $\forall x Cat(x) \rightarrow Animal(x)$

Clausal Form

1. $Dog(D)$ (D is the function that flnds Jack's dog)
2. $Owns(Jack, D)$
3. $\neg Dog(S(x)) \vee \neg Owns(x, S(x)) \vee AnimalLover(x)$
4. $\neg AnimalLover(w) \vee \neg Animal(y) \vee \neg Kills(w, y)$
5. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
6. $Cat(Tuna)$
7. $\neg Cat(z) \vee Animal(z)$

\Tricks"

- **unifIcation:** needed to match variables and terms
  between clauses that look similar
  See DAA text pp. 103-107.
- **normalization:** put in **clausal form**
  move quantifiers / $\wedge$ / $\vee$ etc.
  and **Skolemization** | remove $\exists$ by giving
  an arbitrary, but unique name to the object in question.
  E.g. D for the dog owned by Jack.
  See DAA pp. 96-96.

Unification

UNIFY (P,Q) takes two atomic sentences P and Q and returns a substitution that makes P and Q look the same.

Rules for substitutions:

- Can replace a variable by a constant.
- Can replace a variable by a variable.
- Can replace a variable by a function expression, as long as the function expression does not contain the variable.

Unifier: a substitution that makes two clauses resolvable.

$v_1 \rightarrow C; v_2 \rightarrow v_3; v_4 \rightarrow f(...)$

1. To resolve two clauses, two literals must match exactly, except that one is negated. Sometimes the literals match exactly as they are, but other times one can be made to match the other by an appropriate substitution.

2. This requires unifIcation.

3. Denote substitutions as shown. variable v1 is replaced by the constant c; variable v4 is replaced by the function f and its arguments.

Unification

$Knows(John, x) \rightarrow Hates(John, x)$

$Knows(John, Jim)$

$Knows(y, Leo)$

$Knows(y, Mother(y))$

$Knows(x, Jane)$

UNIFY($Knows(John, x)$,$Knows(John, Jim)$) =

UNIFY($Knows(John, x)$,$Knows(y, Leo)$) =

UNIFY($Knows(John, x)$,$Knows(y, Mother(y))$) =

UNIFY($Knows(John, x)$,$Knows(x, Jane)$) =

UNIFY($Knows(John, x)$,$Knows(John, Jim)$) = $\{x/Jim\}$

UNIFY($Knows(John, x)$,$Knows(y, Leo)$) = $\{x/Leo, y/John\}$

UNIFY($Knows(John, x)$,$Knows(y, Mother(y))$) =
$\{y/John, x/Mother(John)\}$

UNIFY($Knows(John, x)$,$Knows(x, Jane)$) = $fail$

- Want to use the KB to flnd out who John hates.
- Need to flnd those sentences that unify with Knows(John,x) and then apply the unifler to Hates(John,x).
- Remember that x and y are universally quantifled
- Last one fails because x can't take on both the value John and the value Jane But intuitively we know that everyone john knows he hates and everyone knows Jane so we should be able to infer that John hates Jane.
- This is why we required every variable to have a separate name. Knows(John,x) and Knows(y,Jane) works.

Most General Unifler

In cases where there is more than one substitution choose the one that makes the least commitment about the bindings.

UNIFY($Knows(John, x), Knows(y, z)$)

$= \{y/John, x/z\}$

or $\{y/John, x/z, z/Freda\}$

or $\{y/John, x/John, z/John\}$

or ....