Logic, Data Examples and Learning

Lecture 2/5: Fitting algorithms for logical concept classes

Balder ten Cate

Tsinghua University, May 13, 2026

2. From Examples to Concepts

Recap of Lecture 1

Last time, we mostly looked at the direction

$$\text{concepts} \longrightarrow \text{examples}.$$

The guiding questions were:

- How can examples explain what a concept means?
- Can a finite set of examples identify a concept uniquely?
- How does the answer depend on the concept class?

Along the way, we saw both encouraging examples and strong limitations.

How Lecture 1 Ended

The final case study was the fragment $\mathrm{FO}_{\exists,\wedge,\top}[\mathbf{S}]$.

- This concept class is monotone with respect to the homomorphism preorder.
- We wanted to find finite sets of examples that uniquely characterize a given formula.
- This led us to study which finite structures have a *frontier* in the homomorphism lattice.
- A classical theorem of Erdős implies that frontiers do *not* always exist.
- On the positive side, there is a criterion (c-acyclicity) telling us exactly when frontiers *do* exist (and hence, when a $\mathrm{FO}_{\exists,\wedge,\top}$-formula is a uniquely characterized by a finite set of examples).

Today: Reverse the Direction

Today we move in the opposite direction:

$$\text{labeled examples} \longrightarrow \text{concepts}.$$

Starting from positive and negative examples, we want to construct a concept that fits them.

This sounds simple, but it very quickly becomes algorithmic.

Questions for Today

Given labeled examples, we can ask:

- Does a fitting concept exist?
- Can we construct one efficiently?
- If there are many fitting concepts, which one should we prefer?
- When the syntax has a size measure, can we find a *small* fitting concept?

The first half of today is centered around these questions.

The Fitting Problem

Let $\mathsf{C} = (C, \mathcal{X}, \lambda)$ be a concept class and let E be a set of labeled examples.

Definition (fitting)

A concept $c \in C$ *fits* E if it classifies every example in E correctly.

This leads to three basic computational tasks:

Fitting-Verification	given E and c, decide whether c fits E
Fitting-Existence	given E, decide whether some c fits E
Fitting-Construction	given consistent E, output a fitting c

A Small Terminology Note

In this course we use the word *algorithm* informally.

- Think: a clear finite procedure that takes an input and produces an output.
- A fully formal definition could be given using Turing machines.
- We will not do that here: it is a bit tedious, and for our purposes unnecessary.

In practice, you will usually know an algorithm when you see one.

Why the Algorithm Matters

A fitting concept need not be unique.

- Sometimes there are infinitely many fitting concepts.
- Different fitting algorithms may return very different outputs.
- Some outputs are more useful than others:
  - more canonical as a candidate for fitting,
  - smaller,
  - easier to compute with,
  - perhaps more likely to generalize.

So today is not just about *existence*, but about *desirable fitting algorithms*.

A First List of Desiderata

For a fitting algorithm `alg`, some natural desiderata are:

- **Efficiency:** ideally, polynomial-time.
- **Extremality:** perhaps return the most specific or most general fitting concept.
- **Small size:** prefer concise fitting concepts when the syntax has a size measure.
- **Generalizability:** fitting the observed data should still say something about unseen data.

Today we focus mainly on the first three.

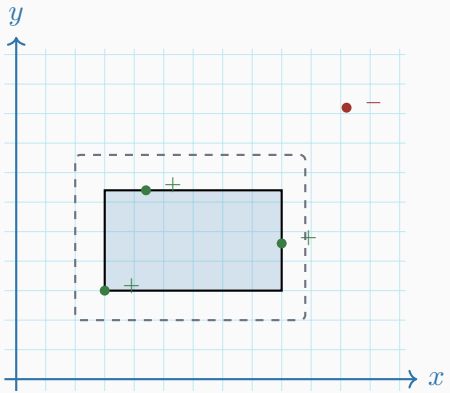
Case Study 1: RECT^{2D}

Definition (RECT^{2D})

$$\text{RECT}^{2D} = (C, \mathbb{R}^2, \lambda)$$

- $C = \{\underline{\text{rect}}(x_1, x_2, y_1, y_2) \mid x_1, x_2, y_1, y_2 \in \mathbb{R}\}$,
- $\lambda(\underline{\text{rect}}(x_1, x_2, y_1, y_2))$ is the set of points (x, y) with

$$x_1 \leq x \leq x_2 \quad \text{and} \quad y_1 \leq y \leq y_2.$$

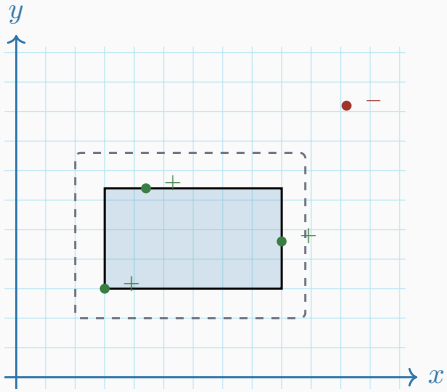


Many Fitting Rectangles

The sample on the right admits many fitting rectangles.

- The solid rectangle is the smallest one containing all positive examples.
- The dashed rectangle also fits the sample.
- In fact, there are infinitely many fitting rectangles.

Which one is *better*? At this point, that is not so clear.



Standard Fitting Algorithm for RECT^{2D}

The fitting problem for RECT^{2D} is easy:

1. Compute the smallest rectangle enclosing the positive examples.
2. Check whether that rectangle contains any negative example.

Canonical candidate for fitting

If any rectangle fits the sample, then the smallest enclosing rectangle already fits it.

So both fitting existence and fitting construction are in polynomial time for RECT^{2D}.

Pseudo-code: Standard Algorithm for RECT^{2D}

```
def fit_rect(E):
    positives = positive_points(E)
    negatives = negative_points(E)
    if positives == []: return EMPTY_RECTANGLE

    (x_min, y_min) = positives[0]
    x_max, y_max = x_min, y_min

    for (x, y) in positives[1:]:
        x_min = min(x_min, x); x_max = max(x_max, x)
        y_min = min(y_min, y); y_max = max(y_max, y)

    for (x, y) in negatives:
        if x_min <= x <= x_max and y_min <= y <= y_max):
            return "no fitting rectangle"

    return rect(x_min, x_max, y_min, y_max)
```

We only make a small number of passes through the input.

What Is Good About the RECT^{2D} Algorithm?

The standard algorithm for RECT^{2D}:

- runs in polynomial time;
- returns a canonical candidate for fitting: if any rectangle fits, this one fits;
- returns the *most specific* fitting rectangle.

But it does *not* yet force a meaningful notion of *small size*: every fitting concept is just one rectangle.

Case Study 2: U-RECT^{2D}

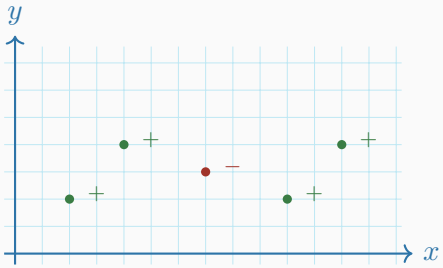
Definition (U-RECT^{2D})

A concept in U-RECT^{2D} is a finite union

$$c_1 \cup \cdots \cup c_\ell$$

where each c_i is a rectangle.

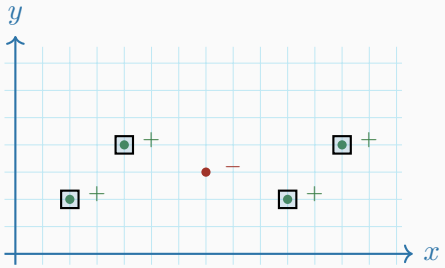
Its *size* is the number ℓ of component rectangles.



Trivial Fitting Algorithm for U-RECT^{2D}

For U-RECT^{2D}, fitting existence is again easy on consistent inputs:

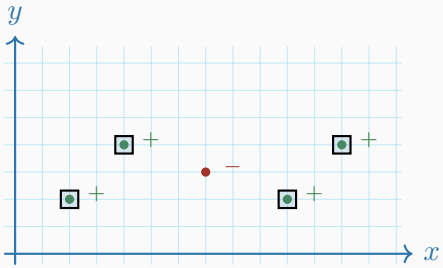
- If the same point occurs both positively and negatively, no fitting concept exists.
- Otherwise, create one point-sized rectangle for each positive example and take the union.



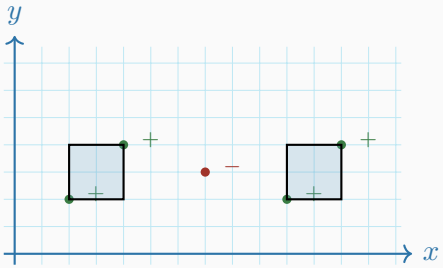
This is a polynomial-time fitting algorithm.

Not All Fitting Unions Are Equally Good

On this sample, the trivial algorithm returns a concept of size 4.



Trivial fitting: size 4



Better fitting: size 2

Why U-RECT^{2D} Is Different

For U-RECT^{2D}, the question “which fitting concept is better?” is much clearer.

- There may still be many fitting concepts.
- But now the syntax carries a natural size measure.
- So it is natural to prefer a fitting concept using fewer rectangles.

This is our first serious encounter with *optimization* inside the fitting problem.

Forward Reference to Lecture 3

Smaller fitting concepts are attractive for several reasons:

- they are easier to communicate and inspect;
- they are often easier to compute with;
- and, in many settings, they have a better chance of generalizing.

In the next lecture, we will make that last point precise.

From Fitting to a Decision Problem

For U-RECT^{2D}, a natural optimization goal is to minimize the number of rectangles.

The associated *decision* problem is:

Size-bounded fitting existence for U-RECT^{2D}

Input: a consistent sample E and a number k.

Question: is there a fitting U-RECT^{2D}-concept of size at most k?

This is the version that will connect naturally to complexity theory.

Intermezzo: Decision Problems, Reductions, and NP-completeness

Complexity Primer: The Pieces We Need

Topics

- PTIME
- NP
- reductions
- NP-completeness
- beyond NP

Decision problem

- Input
- Output: yes / no

Example: Graph 3-Colorability

Input: a graph $G = (V, E)$.

Output: yes iff G is 3-colorable.

PTIME and NP

PTIME (also known as P)

The decision problems solvable in time $f(n)$ for some polynomial f, where n is the size of the input.

NP

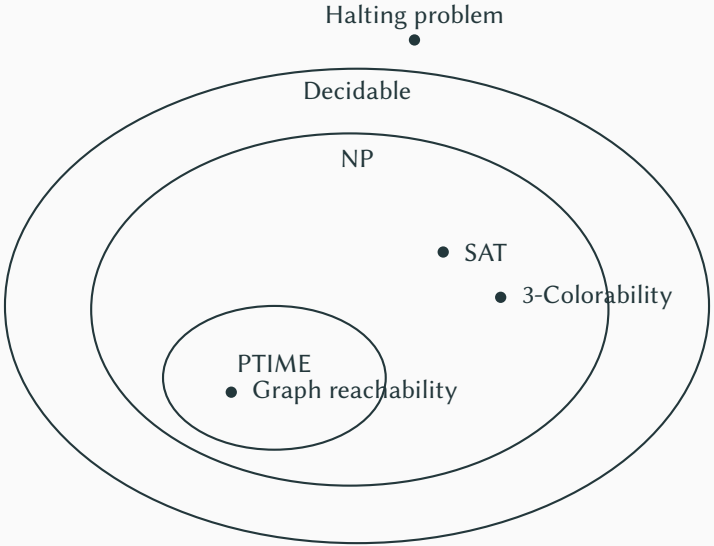
The decision problems that can be phrased as follows:

- input: an instance X;
- output: yes iff there exists $\underbrace{\text{a witness } Y}_{\text{object of polynomial size}}$ such that $\underbrace{\text{something}}_{\text{polynomial-time condition}}$ holds.

Examples of Decision Problems

1. **Has Duplicates**
   Input: a list of natural numbers.
   Output: yes if the list contains duplicates.
   This is in PTIME.
2. **Graph 3-Colorability** is in NP.
3. **Propositional Satisfiability** is in NP.
4. **Graph Reachability**
   Input: $G = (V, E)$ and vertices v_0, v_1.
   Output: yes if v_1 is reachable from v_0.
   This is in NP, in fact already in PTIME.

A Mental Picture



Polynomial-Time Reductions

Reduction from decision problem P_1 to decision problem P_2

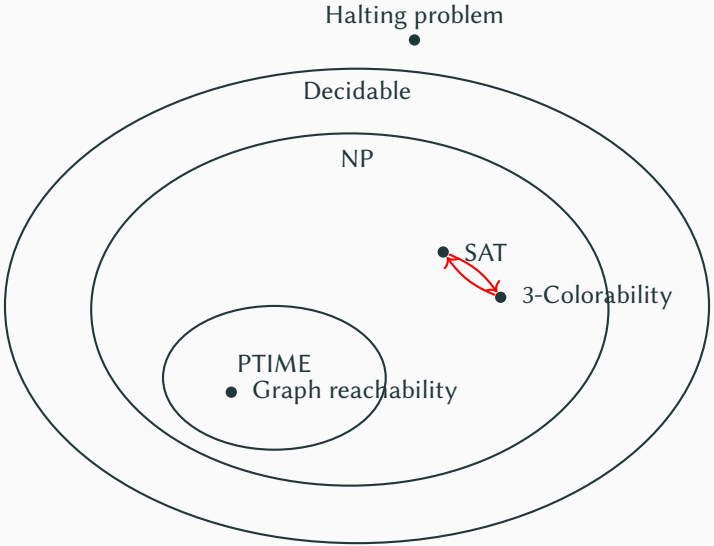
A polynomial-time reduction is an algorithm that takes an instance X of P_1 and outputs an instance Y of P_2 such that

$$X \text{ is a yes-instance } \iff Y \text{ is a yes-instance.}$$

Why reductions matter

If there is a polynomial-time reduction from P_1 to P_2 and P_2 is in PTIME, then P_1 is in PTIME as well.

The Same Picture, After Reductions



NP-hard and NP-complete

NP-hard

A decision problem is NP-hard if every problem in NP has a polynomial-time reduction to it.

NP-complete

NP-complete means: in NP and NP-hard.

Example

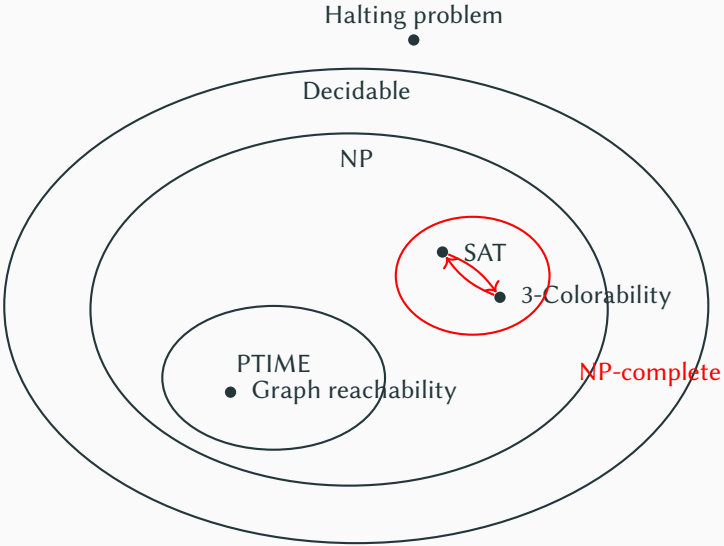
Standard examples of NP-complete problems are:

- Graph 3-Colorability,
- Propositional Satisfiability.

Historical Remark: NP-completeness

- In 1971, Cook and Levin showed that SAT is NP-complete.
- In 1972, Karp showed that many natural combinatorial problems are NP-complete as well.
- Since the 1970s, reductions and NP-completeness have become the standard language for explaining why exact efficient algorithms are unlikely.

The Same Picture, Now with NP-completeness



coNP and Beyond

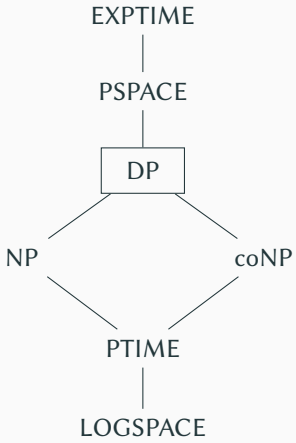
coNP

A decision problem is in coNP if its complement is in NP.

Example

Examples:

- non-3-colorability,
- propositional unsatisfiability.



Back to Our Running Problem

For U-RECT^{2D}, the exact optimization problem

"find a smallest fitting union of rectangles"

is naturally connected to the decision problem

"is there one of size at most k?"

This is the lens through which NP-hardness will enter.

Optimization Intermezzo: Set Cover

Decision Problems versus Optimization Problems

A *decision problem* asks for a yes/no answer.

- Example: does there exist a fitting U-RECT^{2D}-concept of size at most k?

An *optimization problem* asks for the best solution according to some objective.

- Example: find a fitting U-RECT^{2D}-concept of minimum size.

The second is often harder than the first.

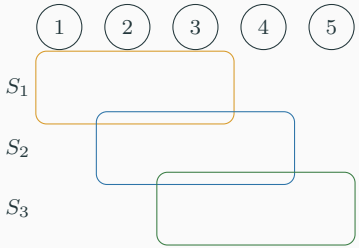
Set Cover

Definition (Set Cover)

A set-cover instance is a pair $(U, \mathcal{S})$ where

- U is a finite set,
- $\mathcal{S}$ is a collection of subsets of U whose union is U.

A solution is a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ that still covers U.



Set Cover: Toy Example

Example

Let

$$U = \{1, 2, 3, 4, 5\} \qquad \text{and} \qquad \mathcal{S} = \big\{\{1, 2, 3\}, \{2, 3, 4\}, \{3, 4, 5\}\big\}.$$

Then the optimal solution has size 2, for instance by choosing

$$\{1, 2, 3\} \quad \text{and} \quad \{3, 4, 5\}.$$

Greedy Set Cover Algorithm

The greedy idea is very simple:

1. Start with no chosen sets and with all elements of U still uncovered.
2. Repeatedly choose a set from $\mathcal{S}$ that covers as many uncovered elements as possible.
3. Mark those elements as covered, and continue.
4. Stop when every element of U has been covered.

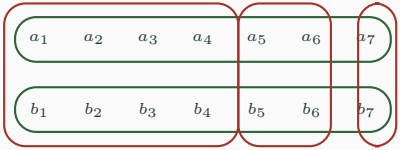
It runs in polynomial time, but it need not be optimal.

Greedy Can Be Suboptimal

- Universe:

$$U = \{a_1, \ldots, a_7, b_1, \ldots, b_7\}$$

- Available sets:
  - the two row-sets $\{a_1, \ldots, a_7\}$ and $\{b_1, \ldots, b_7\}$,
  - and the three column-blocks shown in red.
- Greedy chooses the large left red block first: it covers 8 elements, more than either row-set.
- So greedy ends up with 3 sets, while the optimal solution uses only 2.



Two Classic Facts about Set Cover

Let $n = |U|$ and let ℓ_{opt} be the size of an optimal cover.

- The decision problem

  $$\text{“is there a cover of size at most } k\text{?”}$$

  is NP-complete.
- The greedy algorithm returns a cover of size at most

  $$(\ln n + 1) \cdot \ell_{opt}.$$

- This logarithmic factor is essentially best possible: up to lower-order terms, there is a matching approximability lower bound under standard complexity assumptions.

Historical Remark: Set Cover and Approximation

- In 1972, Set Cover already appeared in Karp's list of classic NP-complete problems.
- In 1974, Johnson showed that greedy gives a logarithmic approximation ratio.
- In 1975, Lovász gave a short LP-based analysis, and in 1979 Chvátal extended it further.
- In 1998, Feige showed that this logarithmic barrier is essentially tight.

So Set Cover became one of the early flagship examples in the theory of approximation algorithms.

How U-RECT^{2D} Turns into Set Cover

Fix a consistent sample E for U-RECT^{2D}.

Set-cover view

- Universe U: the positive examples in E.
- Available sets: all subsets of U that can be covered by a single rectangle while avoiding every negative example.

Then:

- a union of k rectangles fits E
- iff the associated set-cover instance has a cover of size k.

Greedy Fitting Algorithm for U-RECT^{2D}

Once we cast the problem as set cover, we can run the greedy set-cover algorithm.

Result

There is a polynomial-time fitting algorithm for U-RECT^{2D} that outputs a concept of size at most

$$(\ln n + 1) \cdot \ell_{opt},$$

where n is the number of positive input examples.

This is already much better than the trivial algorithm when $\ell_{opt} \ll n$.

Where We Are

Concept class	Algorithm	PTIME	Size guarantee
RECT^{2D}	smallest enclosing rectangle	yes	canonical candidate / most specific
U-RECT^{2D}	point-per-positive	yes	size $O(n)$
U-RECT^{2D}	greedy via set cover	yes	size $\leq (\ln n + 1)\ell_{opt}$

So already with RECT^{2D} and U-RECT^{2D}, we see three themes:

- canonical candidates for fitting,
- exact optimization,
- approximation algorithms.

Takeaway for the First Half

The fitting problem is not just

"find some concept that fits."

It is also about choosing the *right* fitting algorithm and the *right* criterion of quality.

For RECT^{2D}, there is a simple exact algorithm producing a canonical candidate for fitting.

For U-RECT^{2D}, the natural optimization problem is already hard, and approximation enters immediately.

Applying the Same Questions to Logical Concept Classes

Two Case Studies

We now return to two concept classes from Lecture 1:

- $\mathsf{PL}_{\wedge,\top}[\mathsf{PROP}]$
- $\mathsf{FO}_{\exists,\wedge,\top}[\mathbf{S}]$

In both cases, we will ask the same questions as before:

- Is there a natural canonical candidate for fitting?
- Can we compute it efficiently?
- If it is not size-optimal, can we do better algorithmically?

Recall:

- concepts are conjunctions of proposition letters;
- the class is monotone with respect to $\leq_b$;
- canonical examples and canonical concepts both exist;
- greatest lower bounds exist and are computed by bitwise AND.

Canonical concept of a truth assignment t

Take the conjunction of exactly those proposition letters p with $t(p) = 1$.

Standard Fitting Algorithm for $\mathsf{PL}_{\wedge,\top}[\mathrm{PROP}]$

```
def fit_and_top(E):
    safe = set(PROP)

    for t in positive_examples(E):
        for p in PROP:
            if t[p] == 0:
                safe.remove(p)

    phi = conjunction(safe)

    for t in negative_examples(E):
        if satisfies(t, phi):
            return "no fitting formula"

    return phi
```

This is clearly polynomial time.

Equivalent View: Greatest Lower Bound of the Positive Examples

Let $E^+ = \{t_1, \ldots, t_m\}$ be the positive examples.

Step 1

Compute

$$t_{\text{glb}} = glb(t_1, \ldots, t_m),$$

i.e. the bitwise AND of the positive examples.

Step 2

Return the canonical concept of t_{glb}.

This is the same algorithm in more structural language.

Why This Candidate Is Correct

Any fitting conjunction can only use proposition letters that are true in *every* positive example.

Therefore:

- it must satisfy t_{glb};
- its set of satisfying assignments contains the satisfying assignments of the canonical concept of t_{glb}.

Conclusion

If any $\text{PL}_{\wedge,\top}[\text{PROP}]$-formula fits, then the canonical concept of t_{glb} fits.

So this is a *canonical candidate for fitting*, and in fact the most specific fitting conjunction.

But It Need Not Be Small

Consider the sample over $\{p_1, \ldots, p_5\}$:

label	example
$+$	$(1, 1, 1, 0, 0)$
$+$	$(1, 1, 0, 0, 1)$
$-$	$(1, 0, 0, 0, 0)$

The standard algorithm returns

$$p_1 \wedge p_2,$$

because both p_1 and p_2 are true in all positive examples.

But the smaller formula p_2 already fits.

Can We Find a Small Fitting Conjunction?

Let $P \subseteq \text{PROP}$ be the set of *safe-to-use* proposition letters, meaning: true in every positive example.

For each $p \in P$, define

$$S_p := \{t \in E^- \mid t(p) = 0\}.$$

Set-cover view

A subset $X \subseteq P$ yields a fitting conjunction

$$\phi_X := \bigwedge_{p \in X} p$$

if and only if the sets $\{S_p \mid p \in X\}$ cover all negative examples.

So *minimum-size fitting* becomes a Set-Cover problem.

Main Results for $\mathsf{PL}_{\wedge,\top}[\mathsf{PROP}]$

- Computing a minimum-size fitting conjunction is NP-hard.
- The standard fitting algorithm is polynomial time, but may return a much larger conjunction than necessary.
- By reducing to Set Cover and running the greedy Set-Cover algorithm, we obtain a polynomial-time fitting algorithm that returns a conjunction of size at most

$$(\ln |E^-| + 1) \cdot \ell_{opt},$$

where ℓ_{opt} is the minimum possible size.

Summary: $\mathrm{PL}_{\wedge,\top}[\mathrm{PROP}]$

goal / algorithm	PTIME?	guarantee
standard candidate	yes	canonical candidate; most specific
minimum-size fitting	no (NP-hard)	exact optimum
greedy fitting via Set Cover	yes	size $\leq (\ln \lvert E^- \rvert + 1)\ell_{opt}$

$\mathsf{FO}_{\exists,\wedge,\top}[\mathbf{S}]$: Relevant Properties

Recall:

- the class is monotone with respect to the homomorphism pre-order $\leq_{\mathrm{hom}}$;
- canonical examples and canonical concepts exist;
- greatest lower bounds exist in $\leq_{\mathrm{hom}}$;
- those greatest lower bounds are given by direct products.

So structurally, this case looks a lot like $\mathsf{PL}_{\wedge,\top}[\mathsf{PROP}]$ — but the examples are now finite structures.

Quick Homomorphism Quiz: Directed Cycles

C_2

C_3

C_4

Which of these structures homomorphically map to which?

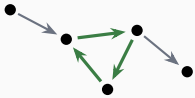
Fact

For directed cycles,

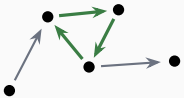
$$C_n \leq_{\text{hom}} C_m \quad \text{iff} \quad m \text{ divides } n.$$

So among these three, the only non-trivial homomorphism is $C_4 \to C_2$.

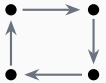
A First Fitting Example



positive



positive



negative

In both positive examples, we can visibly spot a directed triangle. In the negative example, we cannot.

So if we take $A = C_3$, then

$$A \leq_{\text{hom}} B_1, \qquad A \leq_{\text{hom}} B_2, \qquad A \not\leq_{\text{hom}} N.$$

Therefore the canonical concept of C_3 fits this sample.

Recasting the Fitting Problem

For $\mathsf{FO}_{\exists,\wedge,\top}[\mathbf{S}]$, fitting can be rephrased entirely on the example side.

Goal

Find a finite structure A such that:

- $A \leq_{\text{hom}} B$ for every positive example B;
- $A \not\leq_{\text{hom}} N$ for every negative example N.

Then the canonical concept of A is fitting.

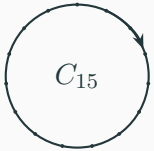
Conversely, every fitting formula gives such a structure via its canonical example.

A More Interesting Example

Suppose now that the positive examples are C_3 and C_5, while the negative example is C_2.

We want a structure A such that

$$A \leq_{\text{hom}} C_3, \qquad A \leq_{\text{hom}} C_5, \qquad A \not\leq_{\text{hom}} C_2.$$

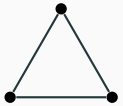


A good answer is C_{15}.
Indeed, 15 is a common multiple of 3 and 5, but not of 2.

First Hiccup: Even Verification Is NP-hard

Think of ordinary graphs as structures with one *symmetric* binary edge relation.

Fix the sample with the single positive example K_3.
Given any graph G, let ϕ_G be the canonical concept of G.
Then:



K_3

$$\phi_G \text{ fits } \{(K_3, +)\} \quad \Longleftrightarrow \quad G \leq_{\text{hom}} K_3.$$

But $G \leq_{\text{hom}} K_3$ means exactly that G is 3-colorable.
So already testing whether a given candidate formula fits a sample is NP-hard.

Second Hiccup: Smallest Fittings Can Be Exponentially Large

Take positive examples

$$C_{p_2}, C_{p_3}, \ldots, C_{p_{n+1}}$$

for the first odd primes, and take the negative example C_2.

- A fitting concept exists: use the canonical concept of the cycle

$$C_{p_2 p_3 \cdots p_{n+1}}.$$

- But that cycle already has exponential size in n.

So here the *smallest* fitting concept can already be exponentially large.

Greatest Lower Bounds = Direct Products

For structures A and B over the same schema, the direct product $A \times B$ has:

- domain $\mathrm{Dom}(A) \times \mathrm{Dom}(B)$;
- a fact $R((a_1, b_1), \ldots, (a_k, b_k))$ exactly when both

$$R(a_1, \ldots, a_k) \in A \quad \text{and} \quad R(b_1, \ldots, b_k) \in B.$$

Key fact

$A \times B$ is the greatest lower bound of A and B under $\leq_{\mathrm{hom}}$.

In particular, for directed cycles, $C_3 \times C_5$ is (up to isomorphism) C_{15}.

Standard Fitting Strategy for $\mathsf{FO}_{\exists,\wedge,\top}[\mathbf{S}]$

The same recipe as before now becomes:

1. take the direct product of the positive examples;
2. return its canonical concept;
3. test it on the negative examples.

Guarantee

If any $\mathsf{FO}_{\exists,\wedge,\top}[\mathbf{S}]$-concept fits, then this canonical candidate fits.

It is again the most specific fitting concept — but the direct product may be exponentially large.

Summary: $\mathrm{FO}_{\exists,\wedge,\top}[\mathbf{S}]$

algorithm / goal	guarantee	caveat
direct product + canonical concept	canonical candidate; most specific	may be exponential in size
minimum-size fitting	much harder	even smallest fittings may be exponential

Where This Places the First Three Lectures

- **Lecture 1:** from concepts to examples
  using, among other things, some finite model theory.
- **Lecture 2:** from examples to concepts
  with the focus mostly on algorithms and complexity.
- **Lecture 3:** fitting algorithms that provably generalize from the input data
  using probability theory and combinatorics.