Exploring structures of inferential mechanisms through simplistic digital circuits

25 October 2025, AIC workshop @ ECAI 2025

10th International Workshop on Artificial Intelligence and Cognition

Giovanni Sileno
g.sileno@uva.nl

University of Amsterdam

Jean-Louis Dessalles
jean-louis.dessalles@telecom-paris.fr

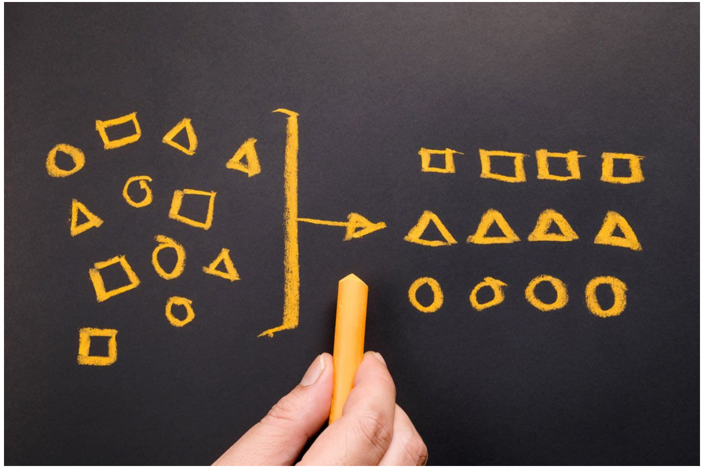
Télécom Paris

Inferential mechanisms in cognitive studies

- Cognitive studies have since long distinguished several types of cognitive mechanisms, by conducting distinct modelling efforts and different types of experiments.

Inferential mechanisms in cognitive studies

- Cognitive studies have since long distinguished several types of cognitive mechanisms, by conducting distinct modelling efforts and different types of experiments.

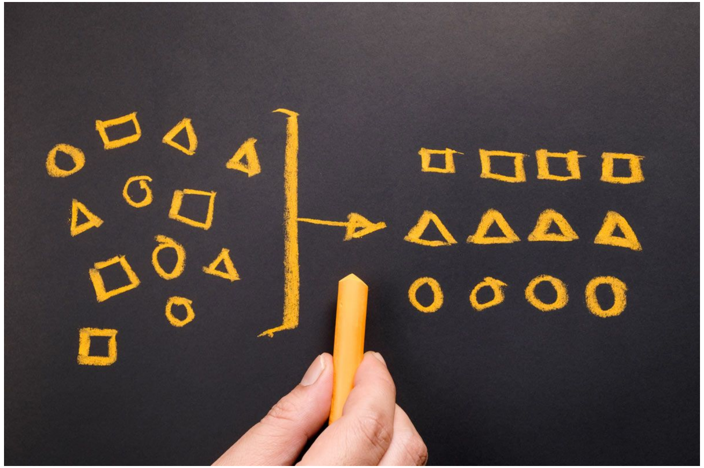


categorization

the process grouping objects, events, or situations, on the basis of shared characteristics

Inferential mechanisms in cognitive studies

- Cognitive studies have since long distinguished several types of cognitive mechanisms, by conducting distinct modelling efforts and different types of experiments.



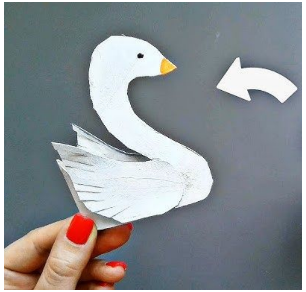
categorization

the process grouping objects, events, or situations, on the basis of shared characteristics

via prototype theory, exemplar theory, rule-based, knowledge-based, Bayesian models…

Inferential mechanisms in cognitive studies

- Cognitive studies have since long distinguished several types of cognitive mechanisms, by conducting distinct modelling efforts and different types of experiments.

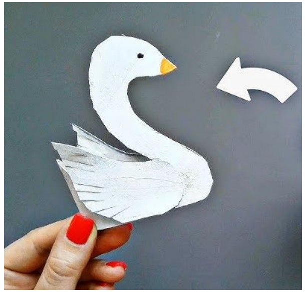


induction

the process drawing general rules/models from observations

Inferential mechanisms in cognitive studies

- Cognitive studies have since long distinguished several types of cognitive mechanisms, by conducting distinct modelling efforts and different types of experiments.



induction

the process drawing general rules/models from observations

via associative (co-occurrence), descriptive (similarity), Bayesian models, …

Inferential mechanisms in cognitive studies

- Cognitive studies have since long distinguished several types of cognitive mechanisms, by conducting distinct modelling efforts and different types of experiments.

deductive reasoning

categorization

causal/diagnostic inference

analogical reasoning

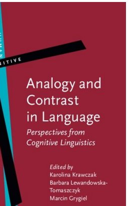
abduction

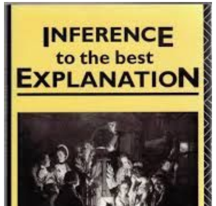
contrast

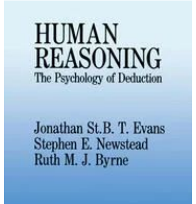
induction

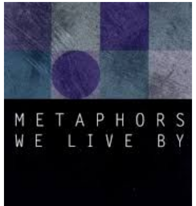
conceptual merge/blending

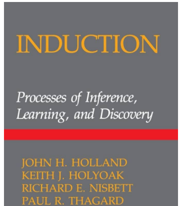
....

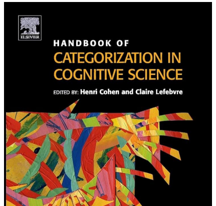


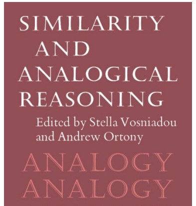


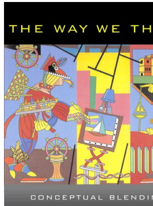






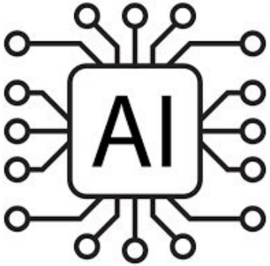






Inferential mechanisms in AI

- The same dispersion can be observed TRADITIONALLY in artificial intelligence, research and practice….



categorization

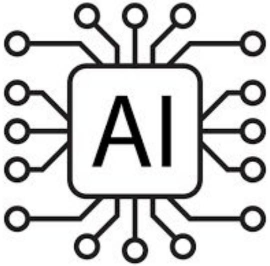
the process grouping objects, events, or situations, on the basis of shared characteristics

induction

the process drawing general rules/models from observations

Inferential mechanisms in AI

- The same dispersion can be observed TRADITIONALLY in artificial intelligence, research and practice….



categorization

the process grouping objects, events, or situations, on the basis of shared characteristics

symbolic AI: *rule-based systems, decision trees, formal concept analysis (FCA), …* **sub-symbolic AI:** *neural networks, support vector machines, clustering, …*

induction

the process drawing general rules/models from observations

symbolic AI: *inductive logic programming, version-space, and explanation-based learning*
sub-symbolic IA: *all machine learning methods (including deep learning and generative AI methods)*

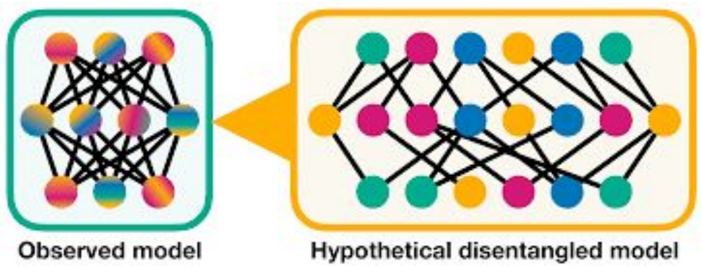
Inferential mechanisms in AI

- The same dispersion can be observed TRADITIONALLY in artificial intelligence, research and practice…. TODAY, with transformer architectures, there is a general belief that we can induce essentially all inferential mechanisms having adequate data, so the problem is now to extract them!

Inferential mechanisms in AI

- The same dispersion can be observed TRADITIONALLY in artificial intelligence, research and practice…. TODAY, with transformer architectures, there is a general belief that we can induce essentially all inferential mechanisms having adequate data, so the problem is now to extract them!

MECHANISTIC INTERPRETABILITY



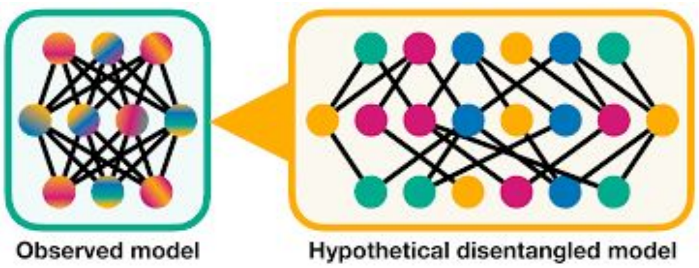
reverse engineer the neural network to identify "circuits" implementing inferential functions

Bereska, Leonard, and Efstratios Gavves. "Mechanistic interpretability for AI safety--a review." *arXiv preprint arXiv:2404.14082* (2024). https://leonardbereska.github.io/blog/2024/mechinterpreview/

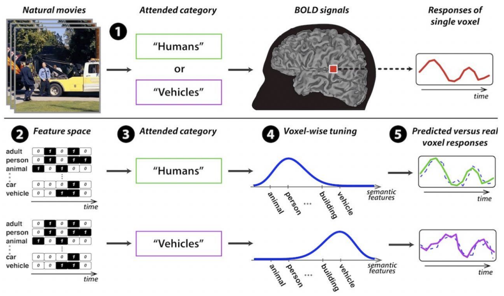
Inferential mechanisms in AI

- The same dispersion can be observed TRADITIONALLY in artificial intelligence, research and practice…. TODAY, with transformer architectures, there is a general belief that we can induce essentially all inferential mechanisms from adequate data, so the problem is now to extract them!

MECHANISTIC INTERPRETABILITY



reverse engineer the neural network to identify "circuits" implementing inferential functions

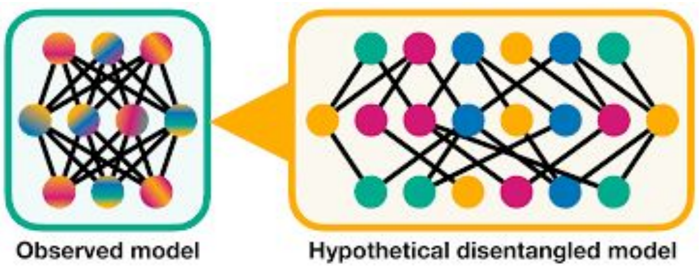


affinity with Computational Neuro-Science

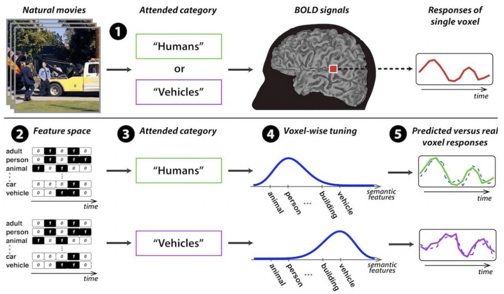
Inferential mechanisms in AI

- The same dispersion can be observed TRADITIONALLY in artificial intelligence, research and practice…. TODAY, with transformer architectures, there is a general belief that we can induce essentially all inferential mechanisms from adequate data, so the problem is now to extract them!

MECHANISTIC INTERPRETABILITY



reverse engineer the neural network to identify "circuits" implementing inferential functions



affinity with Computational Neuro-Science

<u>still none of these approaches say where all inferential mechanisms come from!</u>

Restarting from scratch…

- Rather then reverse engineering the brain, we could investigate an inferential system from its minimal core. But what would that be?

Restarting from scratch…

- Rather then reverse engineering the brain, we could investigate an inferential system from its minimal core. But what would that be?
- **Intuition:** several inferential mechanisms have been reproduced with success through symbolic AI methods ⇒ they *SHOULD* get some aspect of the cognitive functions right.

Restarting from scratch…

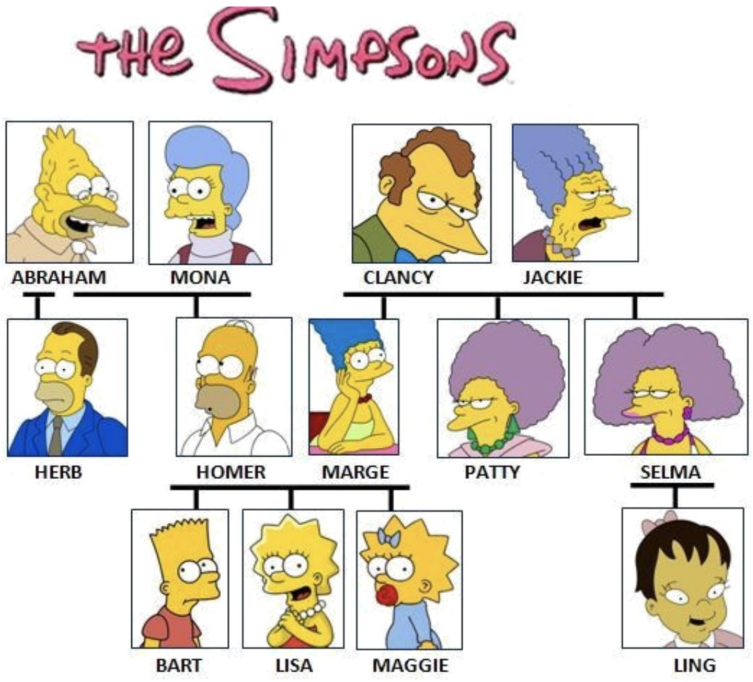
- Rather then reverse engineering the brain, we could investigate an inferential system from its minimal core. But what would that be?
- **Intuition:** several inferential mechanisms have been reproduced with success through symbolic AI methods ⇒ they *SHOULD* get some aspect of the cognitive functions right.
- **This is a weaker assumption than hypotheses like *Language of Thought* (LoT) or *Physical Symbol System* (PSS)!**

Outline of presentation

- Under which constraints logic rules can be interpreted as valid digital circuits?
  - Which inferential constructs can be constructed from these constraints?
    - Generalizing this analysis, we will find an unexpectedly unifying schema!

Let's take a classic symbolic tool for computational inference: logic programming

```
…
parent(marge, lisa).
parent(marge, bart).
parent(marge, maggie).
parent(homer, lisa).
parent(homer, bart).
parent(homer, maggie).
parent(abraham, homer).
parent(abraham, herb).
parent(mona, homer).

child(X,Y) :- parent(Y,X).

?- child(lisa, marge).
true
```

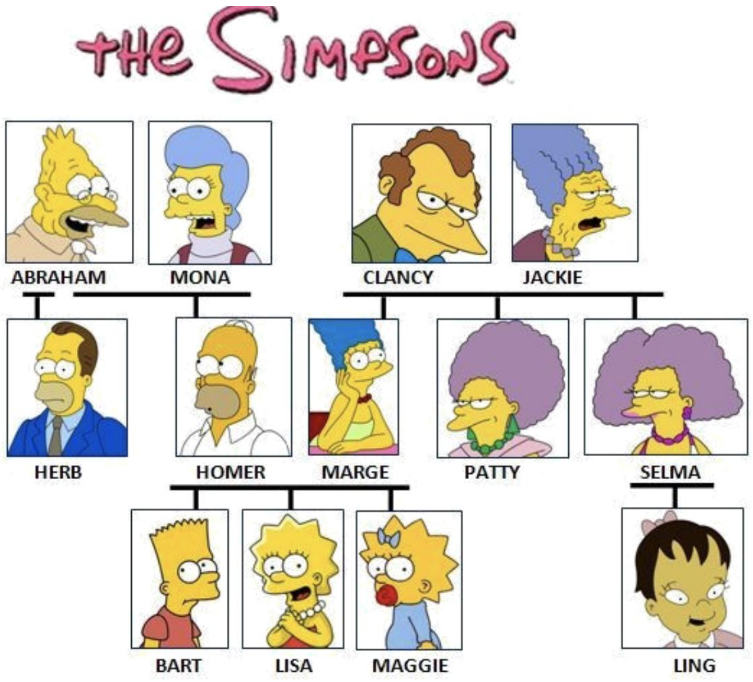


Let's take a classic symbolic tool for computational inference: logic programming

```
…
parent(marge, lisa).
parent(marge, bart).
parent(marge, maggie).
parent(homer, lisa).
parent(homer, bart).
parent(homer, maggie).
parent(abraham, homer).
parent(abraham, herb).
parent(mona, homer).

orphan(X) :- not parent(Y, X).

?- orphan(abraham).
true
```

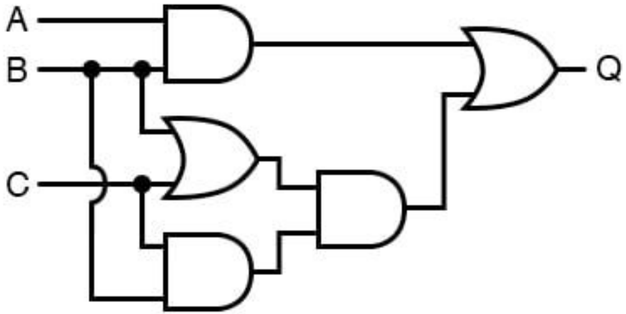
DEFAULT NEGATION

creating
knowledge
out of "ignorance"



Going electrical!

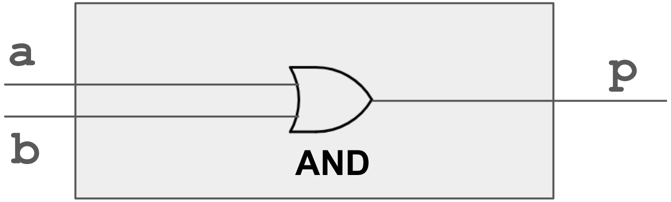
- In order to drop implicit assumptions holding with symbolic methods, let’s think in terms of **digital circuits**: activation at the end maps just to an electrical feedforward mechanism!



Going electrical!

- It seems easy to associate logic rules to logic ports.

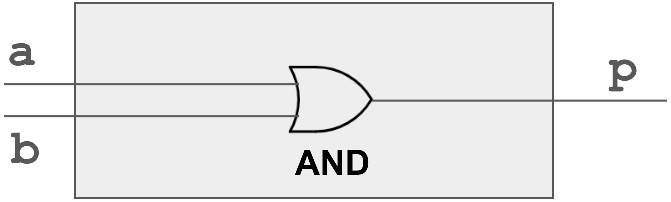
```
p :- a, b.
```



Going electrical!

- It seems easy to associate logic rules to logic ports.

```
p :- a, b.
```



- Yet, in propositional logic, the **contrapositive** holds:

$a \land b \rightarrow p$ ⟺ $\neg p \rightarrow \neg a \lor \neg b$

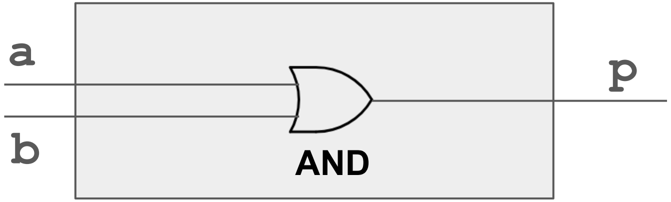
CLASSIC NEGATION

assertions of denial (*it is the case that not …*)

Going electrical!

- It seems easy to associate logic rules to logic ports.

```
p :- a, b.
```



- Yet, in propositional logic, the **contrapositive** holds:

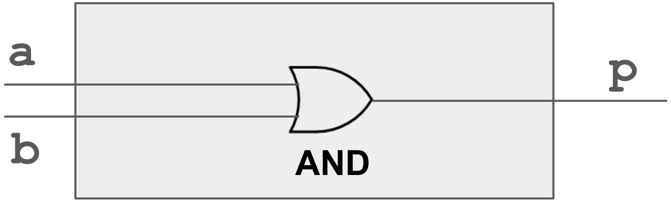
$a \land b \rightarrow p \iff \neg p \rightarrow \neg a \lor \neg b$

CLASSIC NEGATION
assertions of denial (*it is the case that not …*)

This dual circuit is not directly implementable!
The output is not a single port, and it is non-deterministic.

Going electrical!

```
p :- a, b.
-a :- -p, b.
-b :- -p, a.
```



$a \wedge b \rightarrow p \iff \neg p \rightarrow \neg a \vee \neg b$

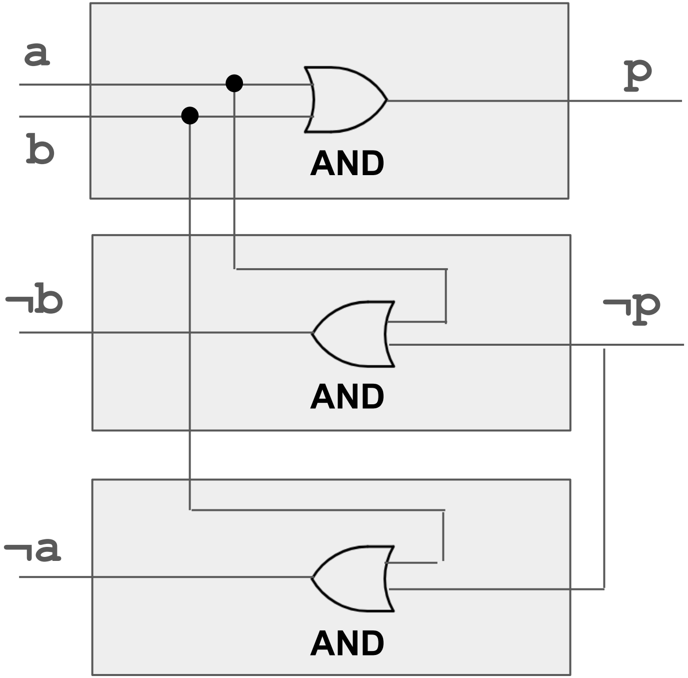
The only valid circuits can be constructed **removing this indeterminism**.

Going electrical!

```
p :- a, b.
-a :- -p, b.
-b :- -p, a.
```

$a \land b \rightarrow p$ ⟺ $\neg p \rightarrow \neg a \lor \neg b$

The only valid circuits can be constructed **removing this indeterminism**.

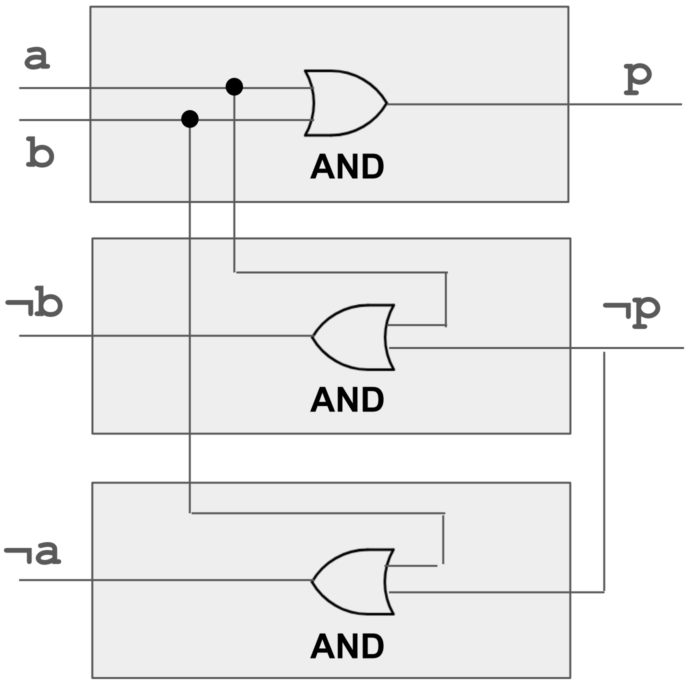


Going electrical!

```
p :- a, b.
-a :- -p, b.
-b :- -p, a.
```

$a \land b \rightarrow p$ ⟺ $\neg p \rightarrow \neg a \lor \neg b$

Semantically, however, logic interacts with the space of **all possible models/states.**



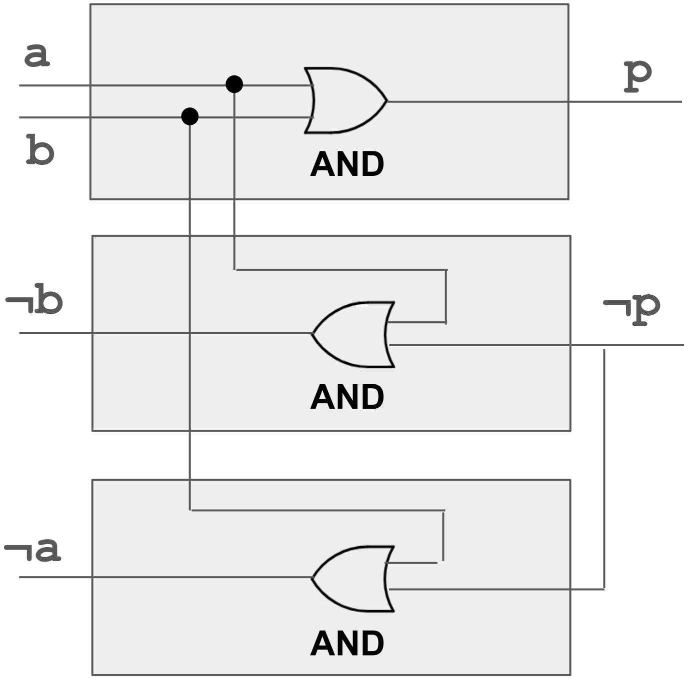
Going electrical!

```
p :- a, b.
-a :- -p, b.
-b :- -p, a.
```

$a \land b \to p$ ⟷ $\neg p \to \neg a \lor \neg b$

Semantically, however, logic interacts with the space of **all possible models/states.**

How to recreate this electrically?



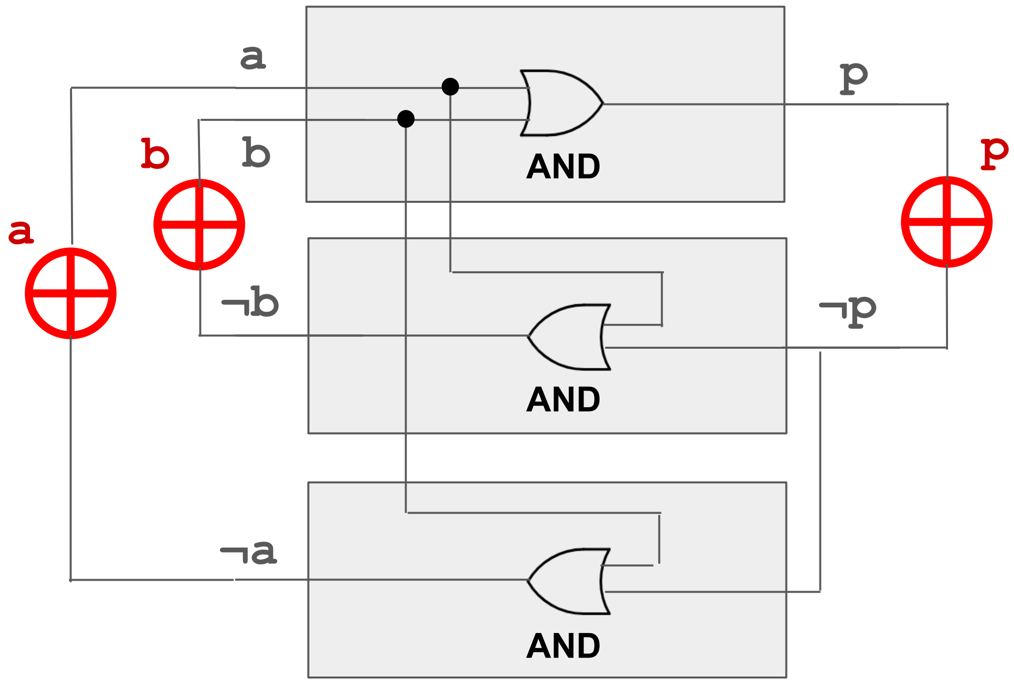
Going electrical!

deterministic machinery
reifying constraints

```
p :- a, b.
-a :- -p, b.
-b :- -p, a.
```

```
1{a; -a}1.
1{b; -b}1.
1{p; -p}1.
```

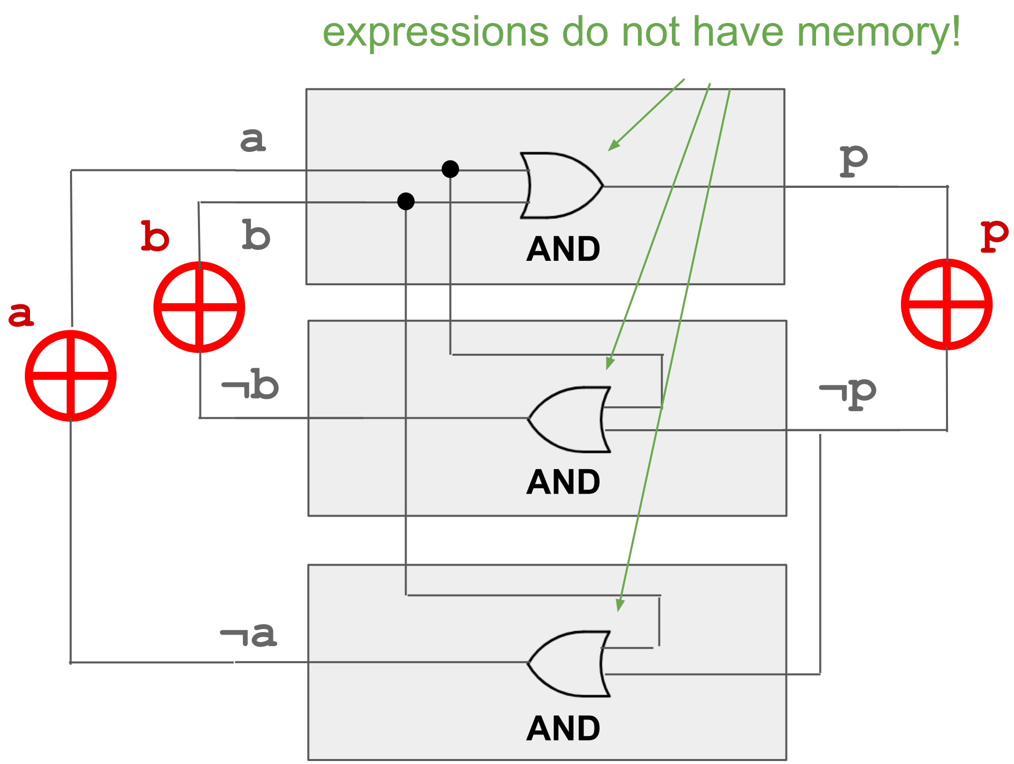
combinatorial exploration



ASP syntax, see https://potassco.org/clingo/run/

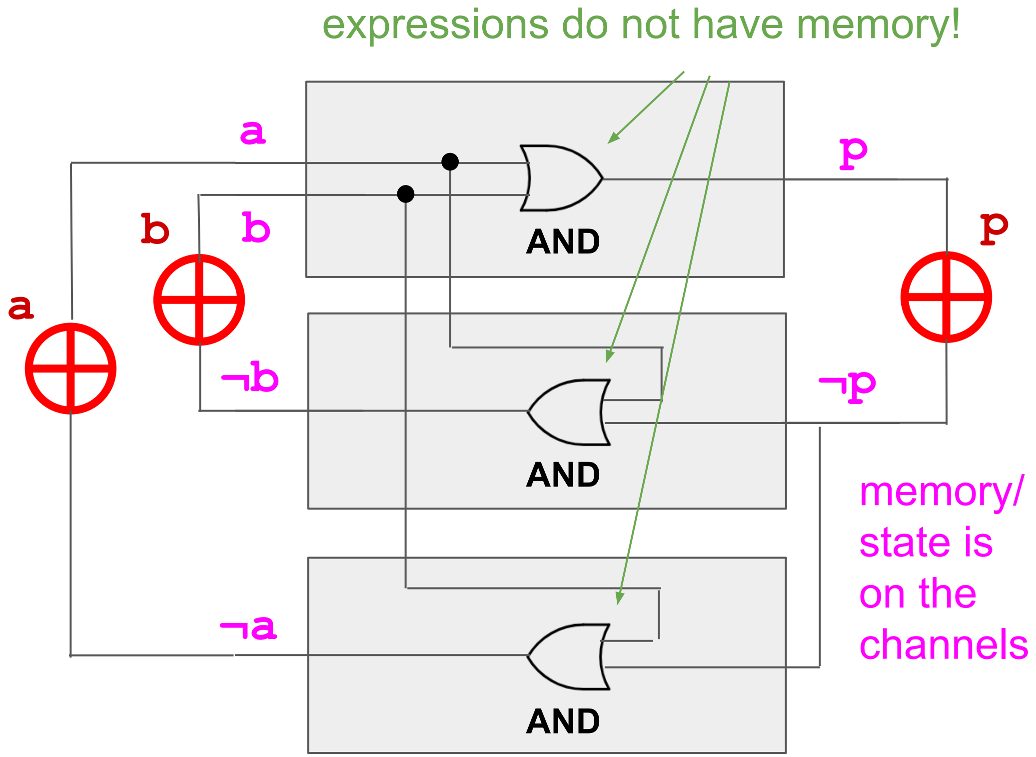
What we learn by going electrical? (1)

```
p :- a, b.
-a :- -p, b.
-b :- -p, a.

1{a; -a}1.
1{b; -b}1.
1{p; -p}1.
```



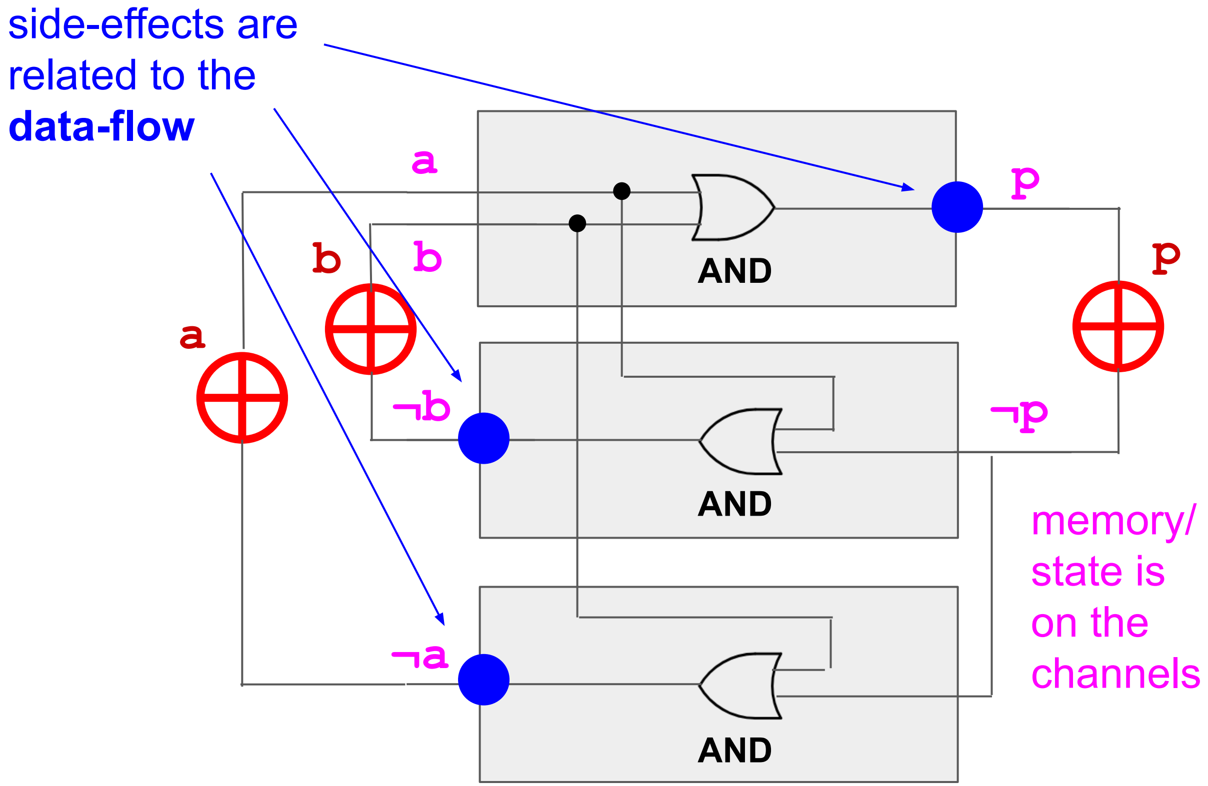
What we learn by going electrical? (1)

```
p :- a, b.
-a :- -p, b.
-b :- -p, a.

1{a; -a}1.
1{b; -b}1.
1{p; -p}1.
```



What we learn by going electrical? (2)

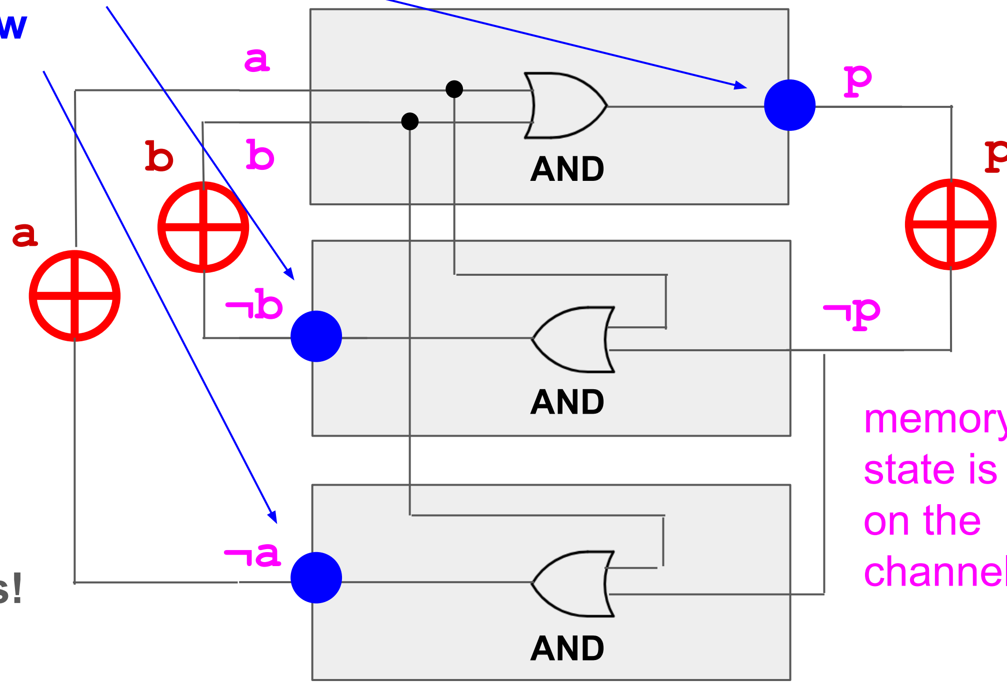
```
p :- a, b.
-a :- -p, b.
-b :- -p, a.

1{a; -a}1.
1{b; -b}1.
1{p; -p}1.
```



What we learn by going electrical? (2)

side-effects are related to the **data-flow**

```
p  :- a, b.
-a  :- -p, b.
-b  :- -p, a.

1{a; -a}1.
1{b; -b}1.
1{p; -p}1.
```

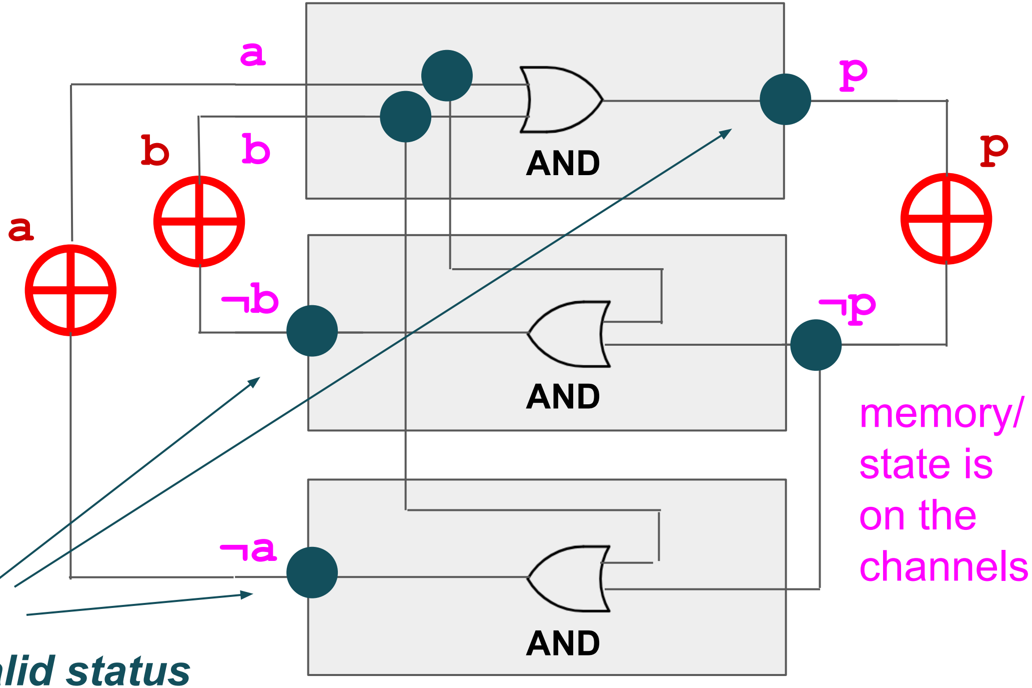


memory/ state is on the channels

conditionals are not simple operators!
implications are topological!

What we learn by going electrical? (3)

```
p :- a, b.
-a :- -p, b.
-b :- -p, a.

1{a; -a}1.
1{b; -b}1.
1{p; -p}1.
```

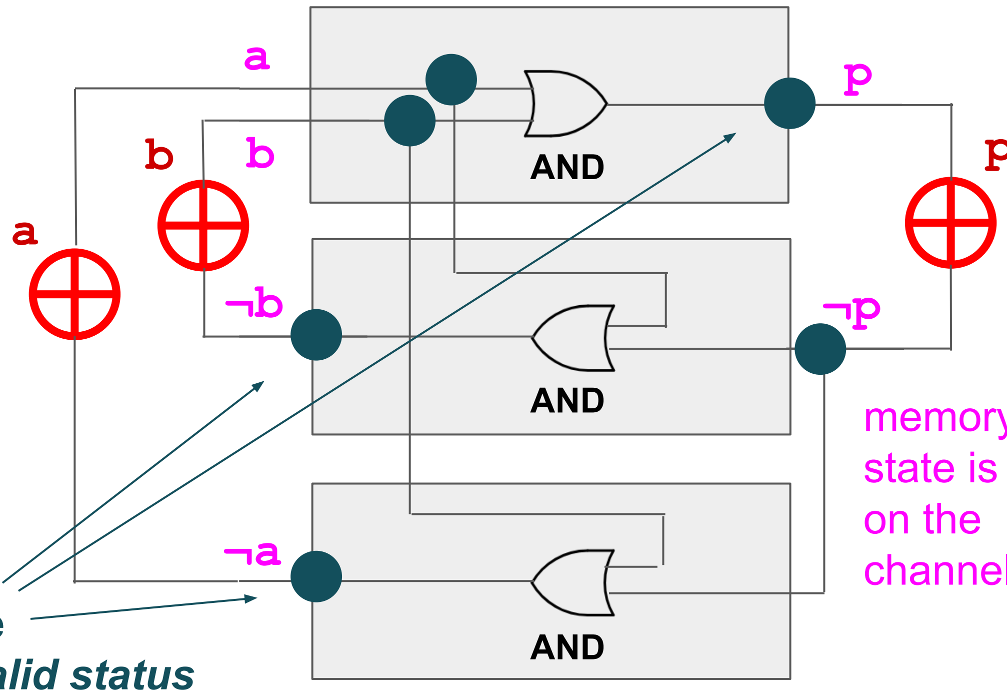


channels may receive
opposite signals: ***invalid status***

What we learn by going electrical? (3)

```
p :- a, b.
-a :- -p, b.
-b :- -p, a.

1{a; -a}1.
1{b; -b}1.
1{p; -p}1.
```

we should purge all models with contradictions

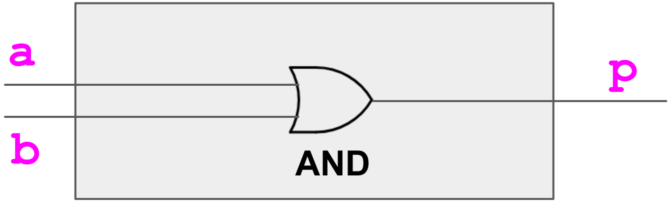


memory/ state is on the channels

channels may receive opposite signals: ***invalid status***

Going electrical: conclusions!

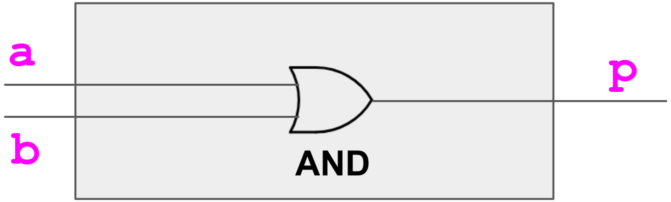
```
p :- a, b.
```



logic rules as in LP are more sound specifications of activation mechanisms. Yet, we should not use default negation!

Going electrical: conclusions!

```
p :- a, b.
```



logic rules as in LP are more sound specifications of activation mechanisms. Yet, we should not use default negation!

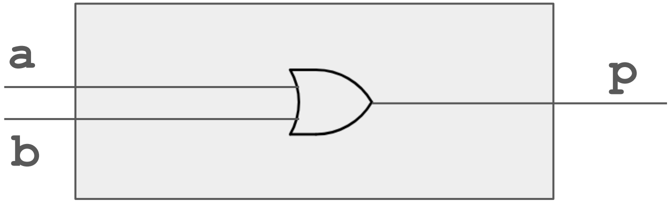
We have some indication on **minimal activation mechanisms**: **inputs can be composed only with OR ";" and AND "," ..** *but what about outputs?*

A suite of dependencies! (abusing the standard LP notation)

```
p :- a, b.
p :- a; b.
p, q :- a.
p; q :- a.
```

1. Conjunction in body

```
p :- a, b.
```



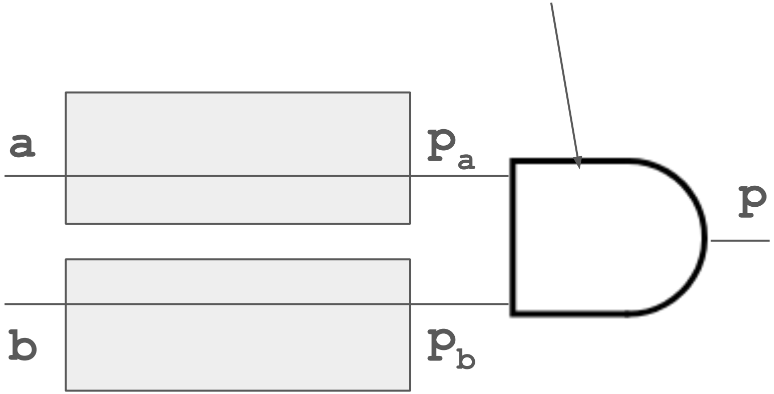
2. Disjunction in body

```
p :- a; b.
```

⟺

```
p :- a.
p :- b.
```

this acts like an accumulation bus

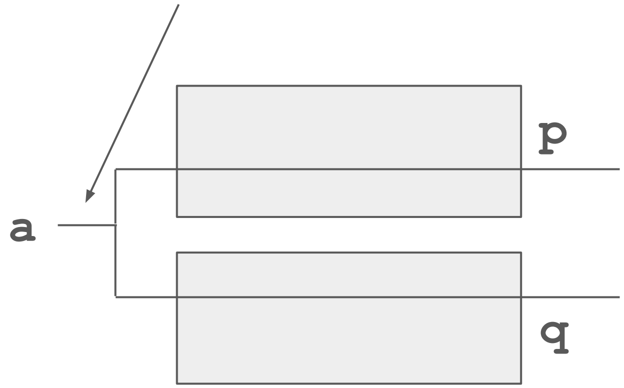


3. Conjunction in head

`p, q :- a.` ⟺

```
p :- a.
q :- a.
```

this acts like a broadcast bus



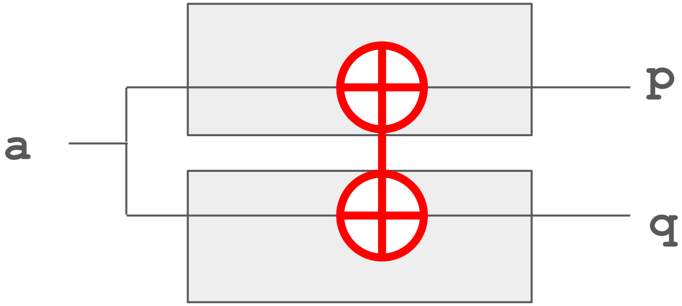
This form is also expressed in the contrapositive of (2)

$a \land b \rightarrow p$ ⟺ $\neg p \rightarrow \neg a \lor \neg b$

4. Disjunction in head

```
p; q :- a.
```

non-deterministic!



This form is also expressed in the contrapositive of (1)

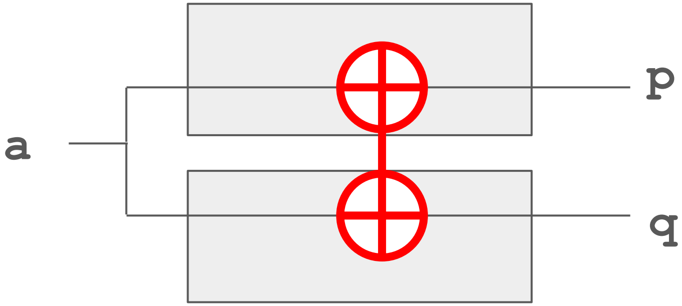
$a \lor b \rightarrow p \iff \neg p \rightarrow \neg a \land \neg b$

(If made deterministic, it would be just as form 3.)

4. Disjunction in head

```
p; q :- a.
```

non-deterministic!



This form is also expressed in the contrapositive of (1)

$a \lor b \rightarrow p$ ⟺ $\neg p \rightarrow \neg a \land \neg b$

~~(If made deterministic, it would be just as form 3.)~~

But not if interpreted as a XOR!

From propositions to predicates…

- So far we considered only the propositional case. However, features are always about some entity, just as predicates are always about something.
- Let us consider how these dependencies would appear in common usages in logic programming.
- We will consider two cases:
  - **Unary predicates** (concerning only a single entity)
  - **Binary predicates** (relating two distinct entities)

Unary predicates (abusing the standard LP notation)

```
p(X) :- a(X), b(X).
p(X) :- a(X); b(X).
p(X), q(X) :- a(X).
p(X); q(X) :- a(X).
```

Unary predicates

```
p(X) :- a(X), b(X).
p(X) :- a(X); b(X).
p(X), q(X) :- a(X).
p(X); q(X) :- a(X).
```

(1) relation defining new concepts:

```
angrydog(X) :- dog(X), angry(X).
```

(2) relation defining taxonomical relations:

```
mammal(X) :- dog(X); cat(X).
```

(3) relation activating back the source concepts:

```
dog(X), angry(X) :- angrydog(X).
```

(4) non-deterministic relation activating alternative choices:

```
dog(X); cat(X) :- mammal(X).
```

Unary predicates

```
p(X) :- a(X), b(X).
p(X) :- a(X); b(X).
p(X), q(X) :- a(X).
p(X); q(X) :- a(X).
```

(1) relation defining new concepts:

```
angrydog(X) :- dog(X), angry(X).
```

(2) relation defining taxonomical relations:

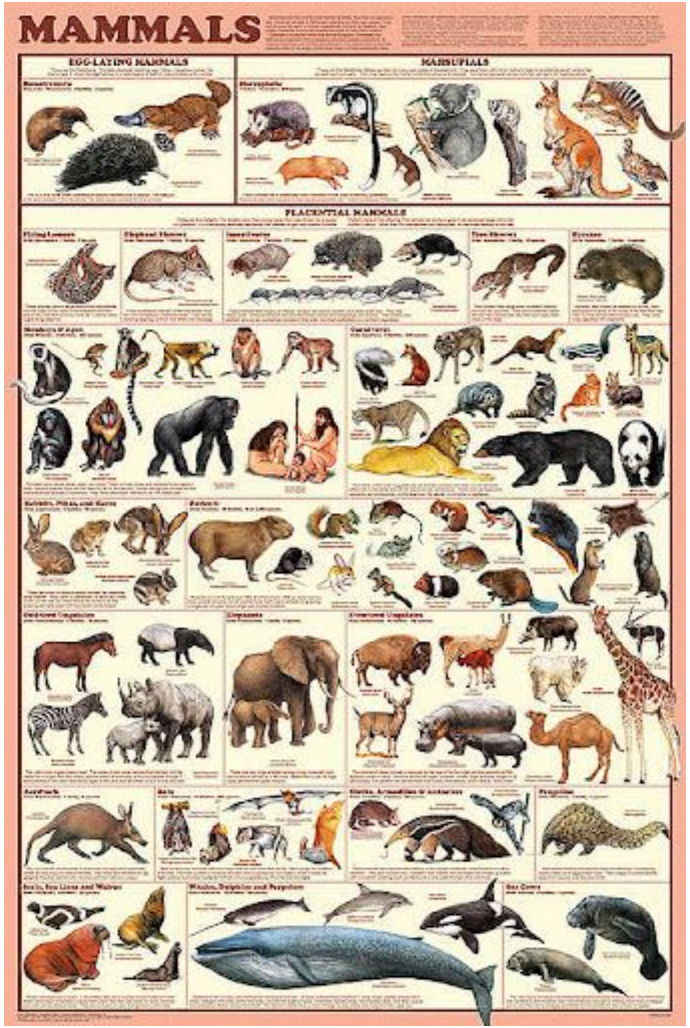
```
mammal(X) :- dog(X); cat(X).
```

(3) relation activating back the source concepts:

```
dog(X), angry(X) :- angrydog(X).
```

(4) non-deterministic relation activating alternative choices:

```
dog(X); cat(X) :- mammal(X).
```



Unary predicates

```
p(X) :- a(X), b(X).
p(X) :- a(X); b(X).
p(X), q(X) :- a(X).
p(X); q(X) :- a(X).
```

(1) relation defining new concepts:

```
angrydog(X) :- dog(X), angry(X).
```

(2) relation defining taxonomical relations:

```
mammal(X) :- dog(X); cat(X).
```

(3) relation activating back the source concepts:

```
dog(X), angry(X) :- angrydog(X).
```

(4) non-deterministic relation activating alternative choices:

```
dog(X); cat(X) :- mammal(X).
```

Unary predicates

```
p(X) :- a(X), b(X).
p(X) :- a(X); b(X).
p(X), q(X) :- a(X).
p(X); q(X) :- a(X).
```

(1) relation defining new concepts:

```
angrydog(X) :- dog(X), angry(X).
```

(2) relation defining taxonomical relations:

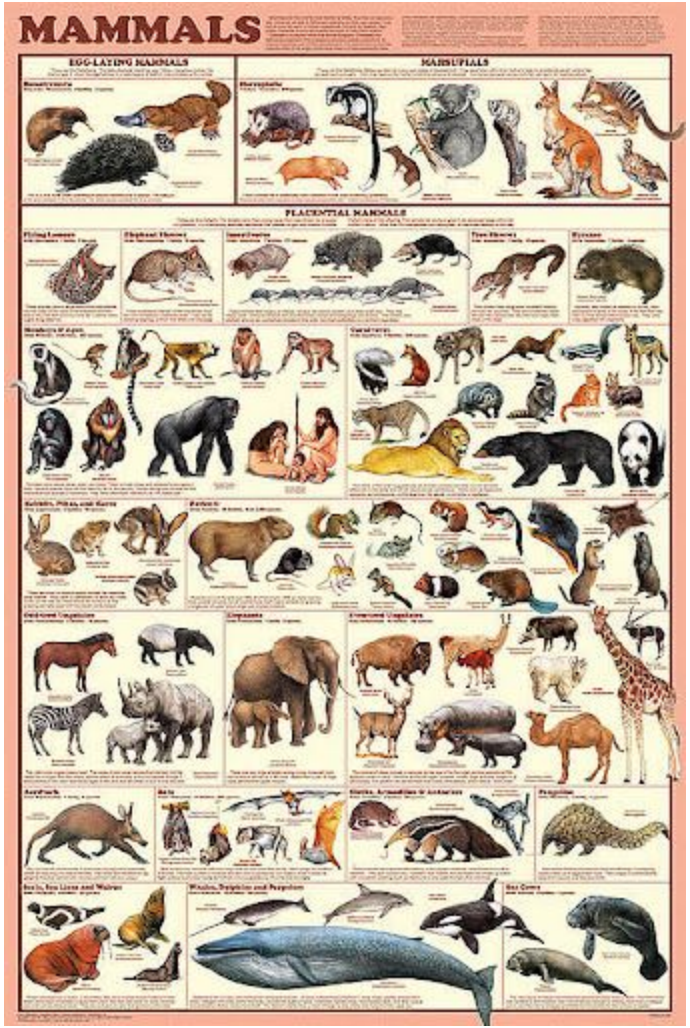
```
mammal(X) :- dog(X); cat(X).
```

(3) relation activating back the source concepts:

```
dog(X), angry(X) :- angrydog(X).
```

(4) non-deterministic relation activating alternative choices:

```
dog(X); cat(X) :- mammal(X).
```



Binary predicates

- Give an entity, we may bind it to other entities

$$dog(x) \wedge tail(y) \wedge has(x, y)$$

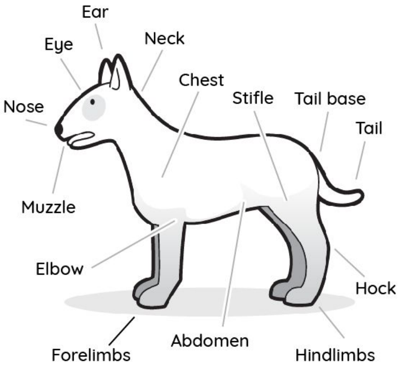
Binary predicates

- Give an entity, we may bind it to other entities

$$dog(x) \land tail(y) \land has(x, y)$$

- At rule level, this becomes an ***existential rule***, which is **not treatable** by standard logic programming derivation (nor by description logic reasoners)

$$\forall x : dog(x) \rightarrow \exists y : tail(y) \land has(x, y)$$



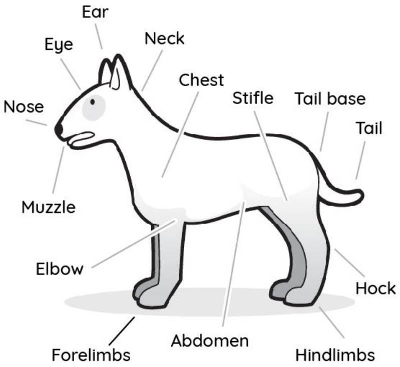
Binary predicates

- Give an entity, we may bind it to other entities

$$dog(x) \land tail(y) \land has(x, y)$$

- At rule level, this becomes an ***existential rule***, which is **not treatable** by standard logic programming derivation (nor by description logic reasoners)

$$\forall x : dog(x) \rightarrow \exists y : tail(y) \land has(x, y)$$



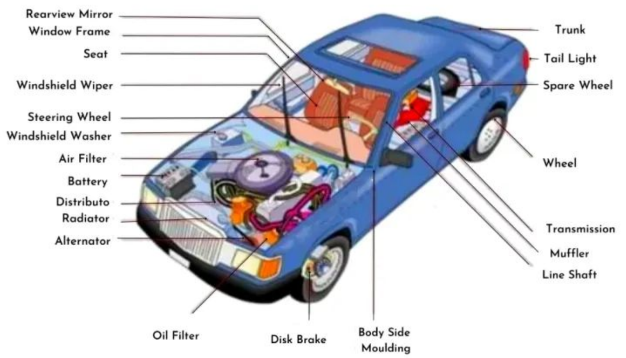
there is a similar problem with active rules (causal laws!)

Binary predicates (abusing the standard LP notation)

```
p(X) :- Y/ a(X, Y), Z/ b(X, Z).
p(X) :- Y/ a(X, Y); Z/ b(X, Z).
Y/ p(X, Y), Z/ q(X, Z) :- a(X).
Y/ p(X, Y); Z/ q(X, Z) :- a(X).
```

Binary predicates

```
p(X) :- Y/ a(X, Y), Z/ b(X, Z).
p(X) :- Y/ a(X, Y); Z/ b(X, Z).
Y/ p(X, Y), Z/ q(X, Z) :- a(X).
Y/ p(X, Y); Z/ q(X, Z) :- a(X).
```



(1) relation determining a concept by composition:

```
car(X) :- Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z).
```

(2) specifies an operation of conceptual generalization:

```
student(X) :- Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z).
```

(3) reifies mereonomical relations (similarly to a contructor in OOP):

```
Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z) :- car(X).
```

(4) activates the possible realizations of a concept:

```
Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z) :- student(X).
```

Binary predicates

```
p(X) :- Y/ a(X, Y), Z/ b(X, Z).
p(X) :- Y/ a(X, Y); Z/ b(X, Z).
Y/ p(X, Y), Z/ q(X, Z) :- a(X).
Y/ p(X, Y); Z/ q(X, Z) :- a(X).
```

(1) relation determining a concept by composition:

```
car(X) :- Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z).
```

(2) specifies an operation of conceptual generalization:

```
student(X) :- Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z).
```

(3) reifies mereonomical relations (similarly to a contructor in OOP):

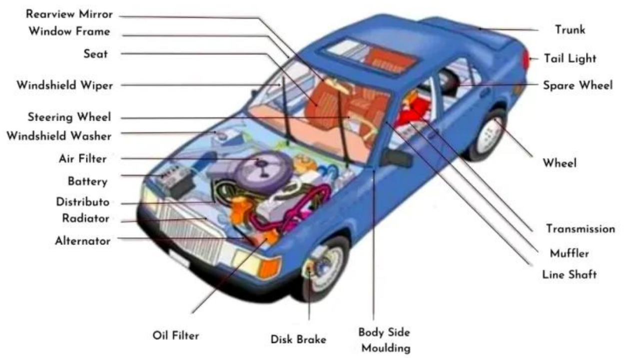
```
Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z) :- car(X).
```

(4) activates the possible realizations of a concept:

```
Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z) :- student(X).
```

Binary predicates

```
p(X) :- Y/ a(X, Y), Z/ b(X, Z).
p(X) :- Y/ a(X, Y); Z/ b(X, Z).
Y/ p(X, Y), Z/ q(X, Z) :- a(X).
Y/ p(X, Y); Z/ q(X, Z) :- a(X).
```



(1) relation determining a concept by composition:

```
car(X) :- Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z).
```

(2) specifies an operation of conceptual generalization:

```
student(X) :- Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z).
```

(3) reifies mereonomical relations (similarly to a ***constructor in Object Oriented Programming***):

```
Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z) :- car(X).
```

(4) activates the possible realizations of a concept:

```
Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z) :- student(X).
```

Binary predicates

```
p(X) :- Y/ a(X, Y), Z/ b(X, Z).
p(X) :- Y/ a(X, Y); Z/ b(X, Z).
Y/ p(X, Y), Z/ q(X, Z) :- a(X).
Y/ p(X, Y); Z/ q(X, Z) :- a(X).
```

(1) relation determining a concept by composition:

```
car(X) :- Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z).
```

(2) specifies an operation of conceptual generalization:

```
student(X) :- Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z).
```

(3) reifies mereonomical relations (similarly to a contructor in OOP):

```
Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z) :- car(X).
```

(4) activates the possible realizations of a concept:

```
Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z) :- student(X).
```

(1) relation defining new concepts:

```
angrydog(X) :- dog(X), angry(X).
```

(2) relation defining taxonomical relations:

```
mammal(X) :- dog(X); cat(X).
```

(3) relation activating back the source concepts:

```
dog(X), angry(X) :- angrydog(X).
```

(4) non-deterministic relation activating alternative choices:

```
dog(X); cat(X) :- mammal(X).
```

conceptual merge by morphism *(take the prototype dog and make it angrier).* in logic usually operationalized as intersection.

conceptual merge by aggregation *(construct a whole out of components).*

(1) relation determining a concept by composition:

```
car(X) :- Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z).
```

(2) specifies an operation of conceptual generalization:

```
student(X) :- Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z).
```

(3) reifies mereonomical relations (similarly to a contructor in OOP):

```
Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z) :- car(X).
```

(4) activates the possible realizations of a concept:

```
Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z) :- student(X).
```

(1) relation defining new concepts:

```
angrydog(X) :- dog(X), angry(X).
```

(2) relation defining taxonomical relations:

```
mammal(X) :- dog(X); cat(X).
```

(3) relation activating back the source concepts:

```
dog(X), angry(X) :- angrydog(X).
```

(4) non-deterministic relation activating alternative choices:

```
dog(X); cat(X) :- mammal(X).
```

conceptual merge **by morphism** *(take the prototype dog and make it angrier).* in logic usually operationalized as intersection.

COMPREHENSION

conceptual merge **by aggregation** *(construct a whole out of components).*

(1) relation determining a concept by composition:

```
car(X) :- Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z).
```

(2) specifies an operation of conceptual generalization:

```
student(X) :- Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z).
```

(3) reifies mereonomical relations (similarly to a contructor in OOP):

```
Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z) :- car(X).
```

(4) activates the possible realizations of a concept:

```
Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z) :- student(X).
```

(1) relation defining new concepts:

```
angrydog(X) :- dog(X), angry(X).
```

(2) relation defining taxonomical relations:

```
mammal(X) :- dog(X); cat(X).
```

(3) relation activating back the source concepts:

```
dog(X), angry(X) :- angrydog(X).
```

(4) non-deterministic relation activating alternative choices:

```
dog(X); cat(X) :- mammal(X).
```

GENERALIZATION

(1) relation determining a concept by composition:

```
car(X) :- Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z).
```

(2) specifies an operation of conceptual generalization:

```
student(X) :- Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z).
```

(3) reifies mereonomical relations (similarly to a contructor in OOP):

```
Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z) :- car(X).
```

(4) activates the possible realizations of a concept:

```
Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z) :- student(X).
```

(1) relation defining new concepts:

```
angrydog(X) :- dog(X), angry(X).
```

(2) relation defining taxonomical relations:

```
mammal(X) :- dog(X); cat(X).
```

(3) relation activating back the source concepts:

```
dog(X), angry(X) :- angrydog(X).
```

(4) non-deterministic relation activating alternative choices:

```
dog(X); cat(X) :- mammal(X).
```

fusion

GENERALIZATION

(1) relation determining a concept by composition:

```
car(X) :- Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z).
```

(2) specifies an operation of conceptual generalization:

```
student(X) :- Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z).
```

(3) reifies mereonomical relations (similarly to a contructor in OOP):

```
Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z) :- car(X).
```

(4) activates the possible realizations of a concept:

```
Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z) :- student(X).
```

(1) relation defining new concepts:

```
angrydog(X) :- dog(X), angry(X).
```

(2) relation defining taxonomical relations:

```
mammal(X) :- dog(X); cat(X).
```

(3) relation activating back the source concepts:

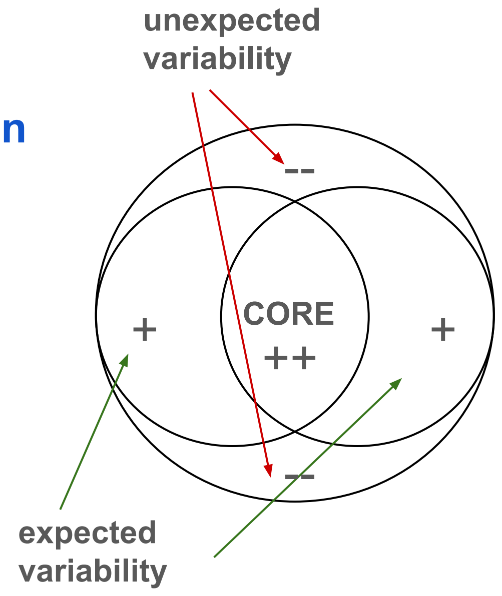
```
dog(X), angry(X) :- angrydog(X).
```

(4) non-deterministic relation activating alternative choices:

```
dog(X); cat(X) :- mammal(X).
```

GENERALIZATION

fusion



(1) relation determining a concept by composition:

```
car(X) :- Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z).
```

(2) specifies an operation of conceptual generalization:

```
student(X) :- Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z).
```

(3) reifies mereonomical relations (similarly to a contructor in OOP):

```
Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z) :- car(X).
```

(4) activates the possible realizations of a concept:

```
Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z) :- student(X).
```

(1) relation defining new concepts:

```
angrydog(X) :- dog(X), angry(
```

(2) relation defining taxonomical relations:

```
mammal(X) :- dog(X); cat(X).
```

(3) relation activating back the source concepts:

```
dog(X), angry(X) :- angrydog(X).
```

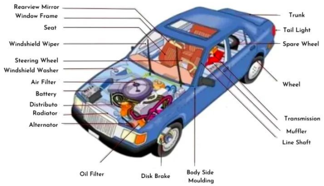
(4) non-deterministic relation activating alternative choices:

```
dog(X); cat(X) :- mammal(X).
```

DESCRIPTION

(1) relation determining a concept by composition:

```
car(X) :- Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X,
```



(2) specifies an operation of conceptual generalization:

```
student(X) :- Y/ humanities(Y), studies(X, Y); Z/ scien            , Z).
```

(3) reifies mereonomical relations (similarly to a contructor in OOP):

```
Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z) :- car(X).
```

(4) activates the possible realizations of a concept:

```
Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z) :- student(X).
```

(1) relation defining new concepts:

```
angrydog(X) :- dog(X), angry(
```

(2) relation defining taxonomical relations:

```
mammal(X) :- dog(X); cat(X).
```

(3) relation activating back the source concepts:

```
dog(X), angry(X) :- angrydog(X).
```

(4) non-deterministic relation activating alternative choices:

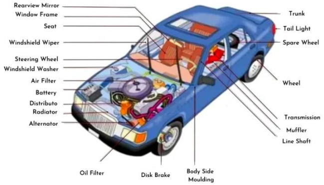
```
dog(X); cat(X) :- mammal(X).
```

telling at large?

DESCRIPTION

(1) relation determining a concept by composition:

```
car(X) :- Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X,
```



(2) specifies an operation of conceptual generalization:

```
student(X) :- Y/ humanities(Y), studies(X, Y); Z/ scien   , Z).
```

(3) reifies mereonomical relations (similarly to a contructor in OOP):

```
Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z) :- car(X).
```

(4) activates the possible realizations of a concept:

```
Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z) :- student(X).
```

(1) relation defining new concepts:

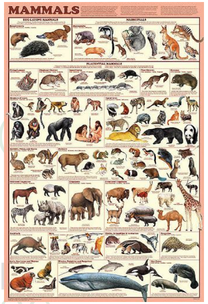
```
angrydog(X) :- dog(X), angry(
```

(2) relation defining taxonomical relations:

```
mammal(X) :- dog(X); cat(X).
```

(3) relation activating back the source conce

```
dog(X), angry(X) :- angrydog(X).
```



(4) non-deterministic relation activating alternative choices:

```
dog(X); cat(X) :- mammal(X).
```

(1) relation determining a concept by composition:

```
car(X) :- Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z).
```

SPECIFICATION

(2) specifies an operation of conceptual generalization:

```
student(X) :- Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z).
```

(3) reifies mereonomical relations (similarly to a contructor in OOP):

```
Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z) :- car(X).
```

(4) activates the possible realizations of a concept:

```
Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z) :- student(X).
```

(1) relation defining new concepts:

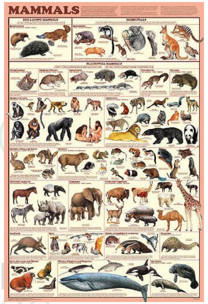
```
angrydog(X) :- dog(X), angry(
```

(2) relation defining taxonomical relations:

```
mammal(X) :- dog(X); cat(X).
```

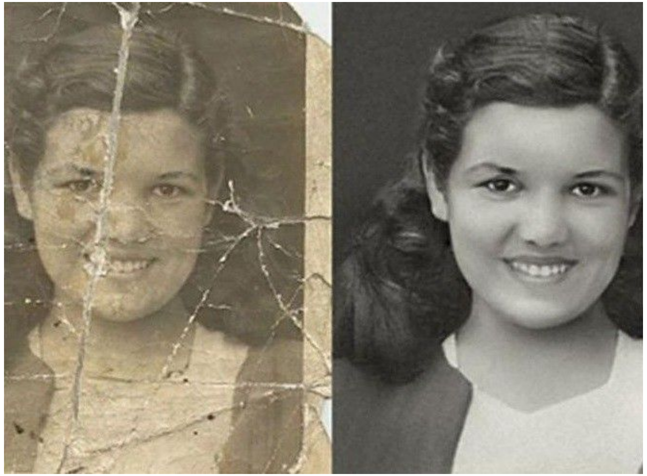
(3) relation activating back the source conce

```
dog(X), angry(X) :- angrydog(X).
```



(4) non-deterministic relation activating alternative choices:

```
dog(X); cat(X) :- mammal(X).
```



“filling the gaps” mechanism
(including text completion, as in LLM)

SPECIFICATION

(1) relation determining a concept by composition:

```
car(X) :- Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z).
```

(2) specifies an operation of conceptual generalization:

```
student(X) :- Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z).
```

(3) reifies mereonomical relations (similarly to a contructor in OOP):

```
Y/ engine(Y), Z/ wheels(Z), has(X, Y), has(X, Z) :- car(X).
```

(4) activates the possible realizations of a concept:

```
Y/ humanities(Y), studies(X, Y); Z/ sciences(Z), studies(X, Z) :- student(X).
```

A suite of mechanisms!

via ***contrast***

3. DESCRIPTION

un-packing

vs

packing

1. COMPREHENSION

via ***merge***

via ***detachment***

4. SPECIFICATION

zooming-in

vs

zooming-out

2. GENERALIZATION

via ***fusion***

A suite of mechanisms!

decompression

via ***contrast***	via ***detachment***
3. DESCRIPTION	**4. SPECIFICATION**
un-packing	**zooming-in**
vs	vs
packing	**zooming-out**
1. COMPREHENSION	**2. GENERALIZATION**
via ***merge***	via ***fusion***

compression

A suite of mechanisms!

decompression

via ***contrast***

3. DESCRIPTION

un-packing

vs

packing

1. COMPREHENSION

via ***merge***

via ***detachment***

4. SPECIFICATION

zooming-in

vs

zooming-out

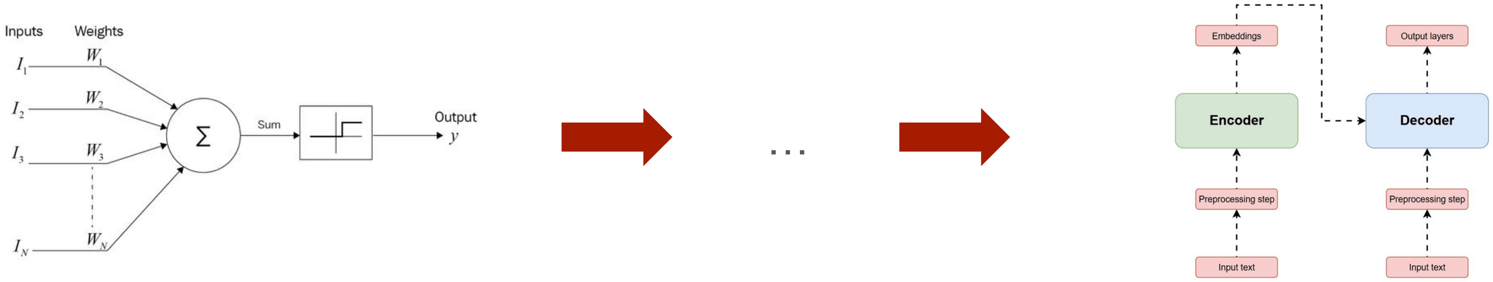
2. GENERALIZATION

via ***fusion***

re-encoding compression ***quantization***

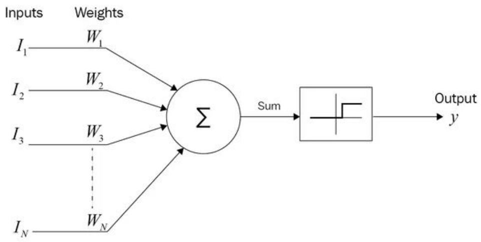
Conclusion

- The first mathematical model of a neuron was the McCulloch-Pitts (MCP) model in 1943, which paved the way to the Perceptron and then… to transformers (including LLMs).

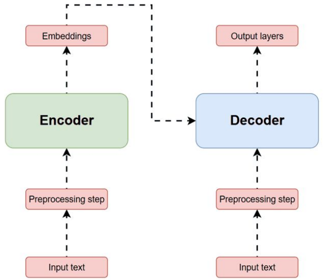


Conclusion

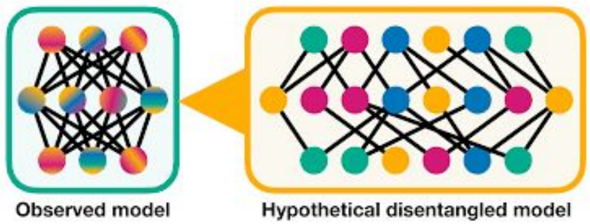
- The first mathematical model of a neuron was the McCulloch-Pitts (MCP) model in 1943, which paved the way to the Perceptron and then… to transformers (including LLMs).

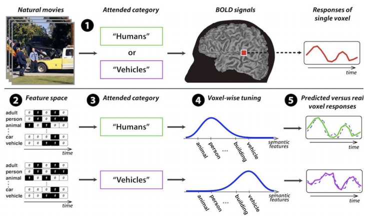


…



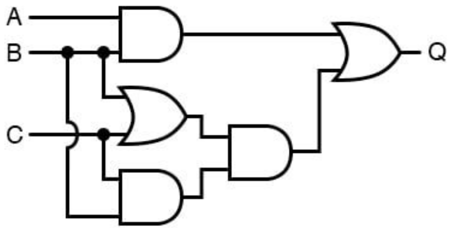
and now we're trying to understand what they do!





Conclusion

- The first mathematical model of a neuron was the McCulloch-Pitts (MCP) model in 1943, which paved the way to the Perceptron and then… to transformers (including LLMs).



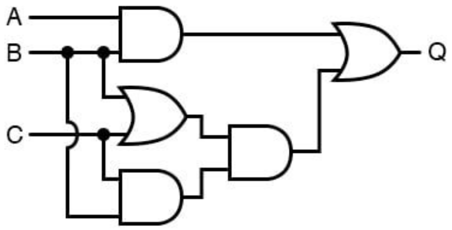
…

```
% neutrality(+Matrix,+Exprs,-Exprs): the function 𝒩(X)
neutrality(AttM, X, Y) :-
    mv_mult(AttM, X, Z),      % ℛ⁺(X)
    maplist(bnot, Z, Y).

% innocuousity(+Matrix,+Exprs,-Exprs): the function ℐ(X)
innocuousity(AttM, X, Y) :-
    transpose(AttM, AttM_t),  % transpose operation
    mv_mult(AttM_t, X, Z).    % ℛ⁻(X)
    maplist(bnot, Z, Y).

% defense(+Matrix,+Exprs,-Exprs): the function ℱ(X)
defense(AttM, X, Y) :-
    neutrality(AttM, X, Z),
    neutrality(AttM, Z, Y).
```

- In this paper, rather than trying to make sense of inferential circuits embedded in transformers, we looked at known inferential constructs by the lens of simple digital circuits…

Conclusion

- The first mathematical model of a neuron was the McCulloch-Pitts (MCP) model in 1943, which paved the way to the Perceptron and then… to transformers (including LLMs).



…

```
% neutrality(+Matrix,+Exprs,-Exprs): the function 𝒩(X)
neutrality(AttM, X, Y) :-
    mv_mult(AttM, X, Z),       % ℛ⁺(X)
    maplist(bnot, Z, Y).

% innocuousity(+Matrix,+Exprs,-Exprs): the function ℐ(X)
innocuousity(AttM, X, Y) :-
    transpose(AttM, AttM_t),  % transpose operation
    mv_mult(AttM_t, X, Z).    % ℛ⁻(X)
    maplist(bnot, Z, Y).

% defense(+Matrix,+Exprs,-Exprs): the function ℱ(X)
defense(AttM, X, Y) :-
    neutrality(AttM, X, Z),
    neutrality(AttM, Z, Y).
```

- In this paper, rather than trying to make sense of inferential circuits embedded in transformers, we looked at known inferential constructs by the lens of simple digital circuits… and we found that:

➡ **there exists a barebone common structure for inferential mechanisms!**

Further results!

- In the paper we have additional results:
  - an application of a probabilistic interpretation of logic programs predicting dependencies across the 4 inferential mechanisms,
  - an elaboration on how learning would work differently for generalization and comprehension,
  - an observation on the fact that LLMs mimic a description mechanism through a specification one, entailing that they may take contrast wrong.
  - and yet others considerations!!

Exploring structures of inferential mechanisms through simplistic digital circuits

25 October 2025, AIC workshop @ ECAI 2025

10th International Workshop on Artificial Intelligence and Cognition

Giovanni Sileno
g.sileno@uva.nl

University of Amsterdam

Jean-Louis Dessalles
jean-louis.dessalles@telecom-paris.fr

Télécom Paris