Constraint Acquisition

Nadjib Lazaar, LIRMM, CNRS, University of Montpellier

M2R, Montpellier, 09 nov. 2021

CP & ML

CONSTRAINT PROGRAMMING (CP)

(variables: X, domains: D,constraints: C)

Constraint Network

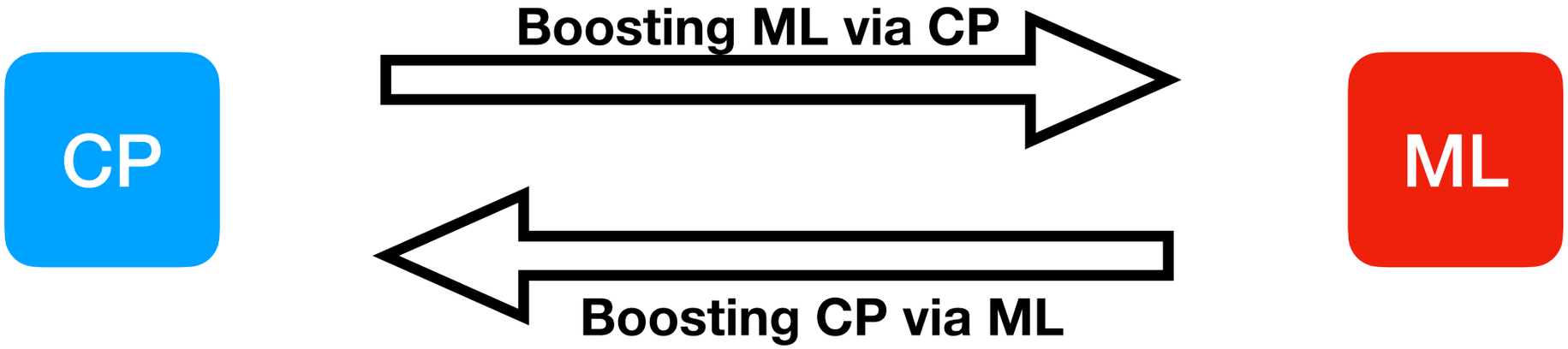
Solution

CP Solver

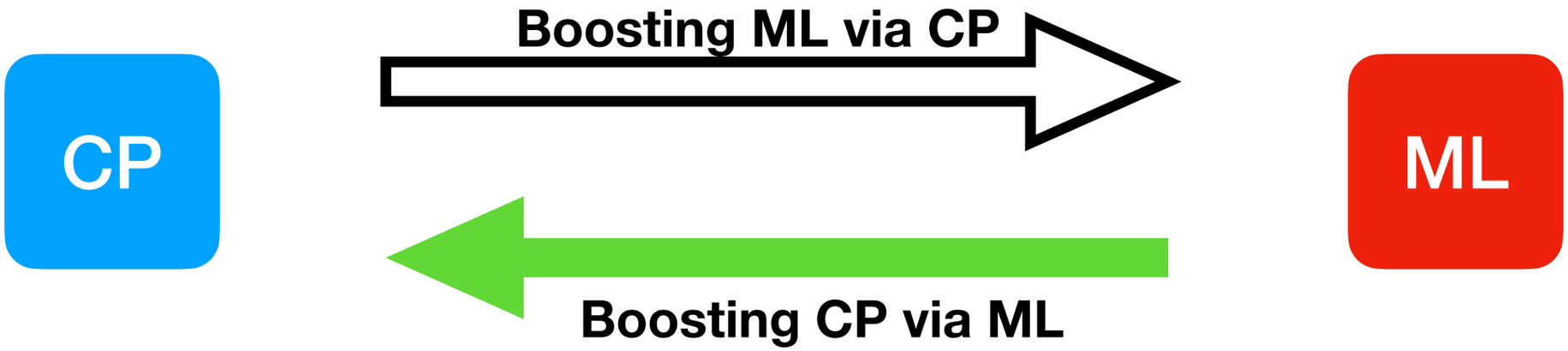
Propagation

Search

ML & CP



ML & CP

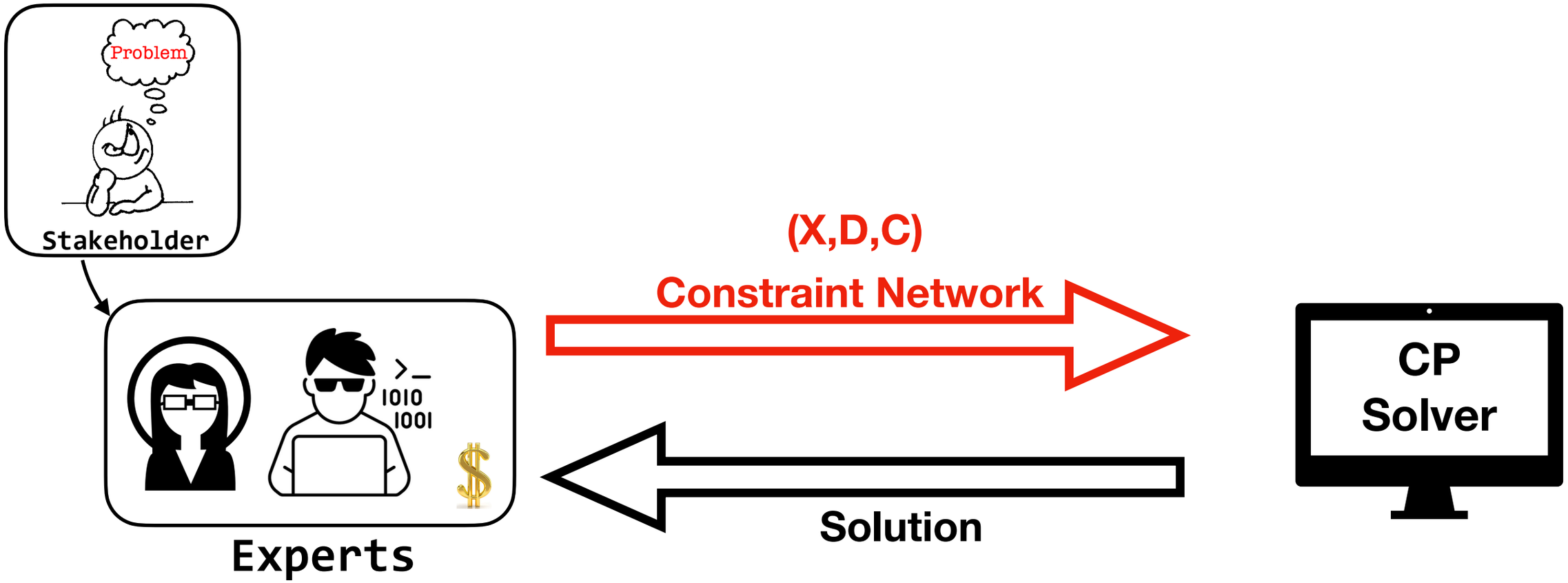


- **ML for Modelling:** From Data to declarative models
  - Constraint Acquisition

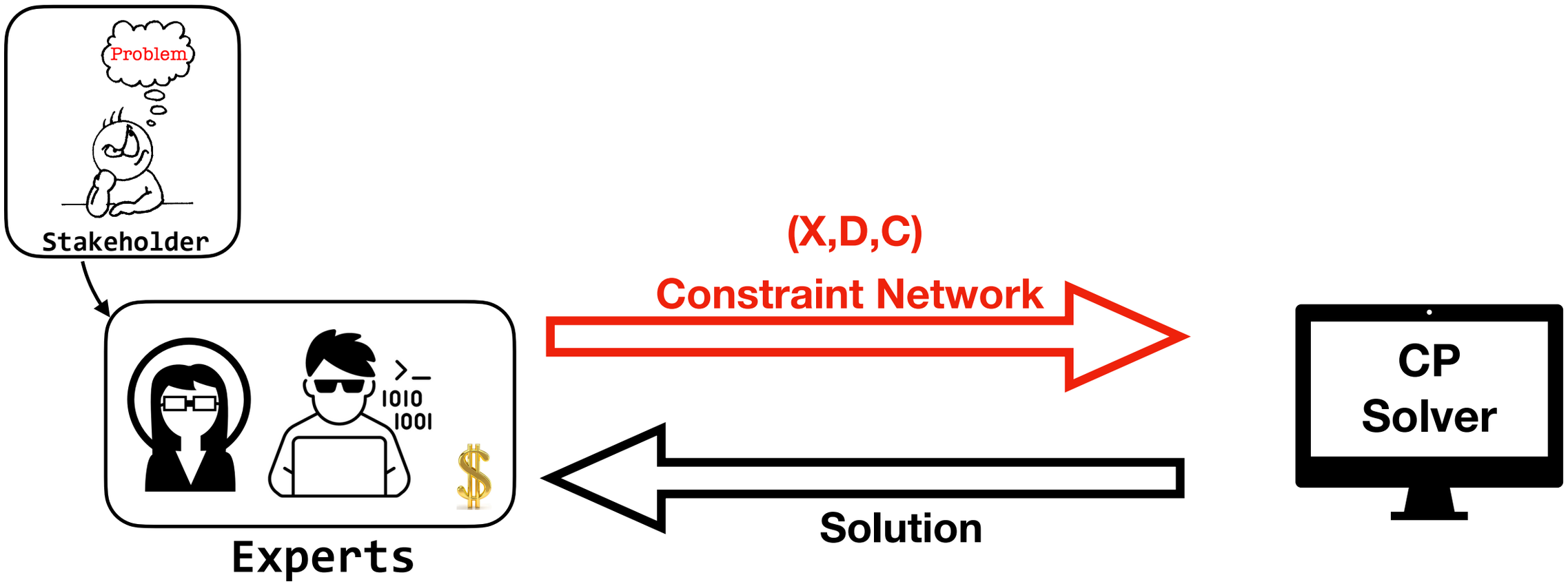
- **ML for Solving:** Use of Data&Models to boost the resolution
  - Adaptive Solver (consistency level, var and val selectors)
  - Clause learning

Constraint Acquisition (CA)

MOTIVATION


Problem
Stakeholder
(X,D,C)
Constraint Network
CP
Solver
>_
1010
1001
$
Experts
Solution

MOTIVATION



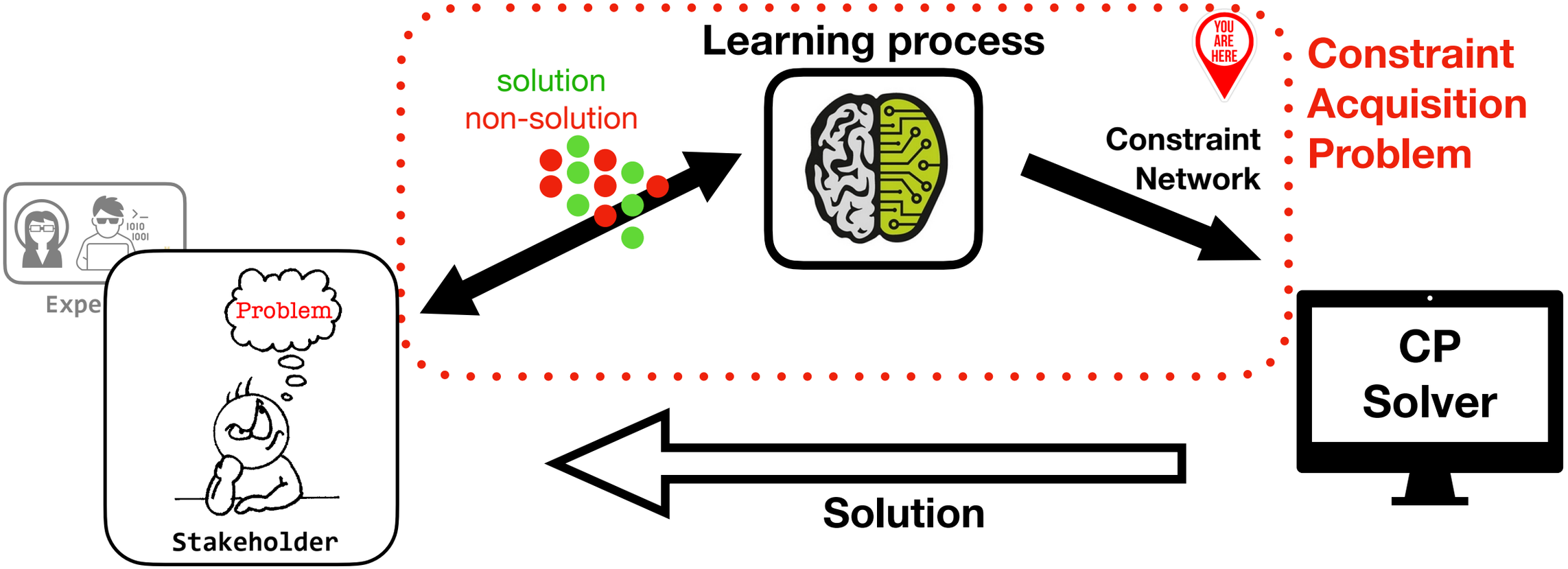
Question: How does the user write down the constraints of a problem?

Limitations:

- Modelling constraint networks require a fair expertise
- Conceptual and technological gap
- Error-prone
- Hard to take advantage of raw data

Need: Simple way to build constraint network ➔ Modeller-assistant

MOTIVATION



Question: How does the user write down the constraints of a problem?

Limitations:

- Modelling constraint networks require a fair expertise
- Conceptual and technological gap
- Error-prone
- Hard to take advantage of raw data

Need: Simple way to build constraint network ➔ Modeller-assistant

ACQUISITION USING STANDARD ML

ACQUISITION USING STANDARD ML

- Empirical Model Learning [Lombardi and Milano, AIJ17]
  - Extracting an Empirical Model using Neural Networks and Decision Trees
  - Empirical Model in terms of variables/constraints

ACQUISITION USING STANDARD ML

- Empirical Model Learning [Lombardi and Milano, AIJ17]
  - Extracting an Empirical Model using Neural Networks and Decision Trees
  - Empirical Model in terms of variables/constraints

- Boundary estimation for constraint optimization problem [Spieker and Gotlieb, 18]
  - Learning boundaries for objective variables
  - Based on supervised learning (data curation, regression models)

ACQUISITION USING STANDARD ML

- Empirical Model Learning [Lombardi and Milano, AIJ17]
  - Extracting an Empirical Model using Neural Networks and Decision Trees
  - Empirical Model in terms of variables/constraints

- Boundary estimation for constraint optimization problem [Spieker and Gotlieb, 18]
  - Learning boundaries for objective variables
  - Based on supervised learning (data curation, regression models)

- Classifier-based CA [Prestwich et al, 21]
  - Training a classifier to discriminate between solutions non-solutions
  - Naive Bayes Classifier for a new passive CA

VERSION SPACE LEARNING (OVERVIEW)

[MITCHELL82]

VERSION SPACE LEARNING (OVERVIEW)

[MITCHELL82]

- Let $X = \{x_1, \ldots, x_n\}$ a set of attributes of domains $D = \{D_1, \ldots, D_n\}$

VERSION SPACE LEARNING (OVERVIEW)

[MITCHELL82]

- Let $X = \{x_1, \ldots, x_n\}$ a set of attributes of domains $D = \{D_1, \ldots, D_n\}$

- A concept is a Boolean function $f : X \rightarrow \{0,1\}$
  - $f(a) = 0 \Rightarrow a \in D^X$ is a negative instance
  - $f(b) = 1 \Rightarrow b \in D^X$ is a positive instance
  - A function of a fixed set of predicates $P_i(X)$, which can be evaluated over instances (ex: $f(X) \equiv P_1(X) \wedge \neg P_2(X) \vee P_3(X)$)

VERSION SPACE LEARNING (OVERVIEW)

[MITCHELL82]

- Let $X = \{x_1, \ldots, x_n\}$ a set of attributes of domains $D = \{D_1, \ldots, D_n\}$

- A concept is a Boolean function $f : X \rightarrow \{0,1\}$
  - $f(a) = 0 \Rightarrow a \in D^X$ is a negative instance
  - $f(b) = 1 \Rightarrow b \in D^X$ is a positive instance
  - A function of a fixed set of predicates $P_i(X)$, which can be evaluated over instances (ex: $f(X) \equiv P_1(X) \wedge \neg P_2(X) \vee P_3(X)$)
  - In particular, restrict attention to pure conjunctive concepts

VERSION SPACE LEARNING (OVERVIEW)

[MITCHELL82]

- Let $X = \{x_1, \ldots, x_n\}$ a set of attributes of domains $D = \{D_1, \ldots, D_n\}$

- A concept is a Boolean function $f : X \to \{0,1\}$
  - $f(a) = 0 \Rightarrow a \in D^X$ is a negative instance
  - $f(b) = 1 \Rightarrow b \in D^X$ is a positive instance
  - A function of a fixed set of predicates $P_i(X)$, which can be evaluated over instances (ex: $f(X) \equiv P_1(X) \wedge \neg P_2(X) \vee P_3(X)$)
    - In particular, restrict attention to pure conjunctive concepts

- The space of all possible $f(X)$ is called the hypothesis space H

VERSION SPACE LEARNING (OVERVIEW)

[MITCHELL82]

- Let $X = \{x_1, \ldots, x_n\}$ a set of attributes of domains $D = \{D_1, \ldots, D_n\}$

- A concept is a Boolean function $f : X \to \{0,1\}$
  - $f(a) = 0 \Rightarrow a \in D^X$ is a negative instance
  - $f(b) = 1 \Rightarrow b \in D^X$ is a positive instance
  - A function of a fixed set of predicates $P_i(X)$, which can be evaluated over instances (ex: $f(X) \equiv P_1(X) \wedge \neg P_2(X) \vee P_3(X)$)
    - In particular, restrict attention to pure conjunctive concepts

- The space of all possible $f(X)$ is called the hypothesis space H
- Version space is the subspace of H consistent with the training set

VERSION SPACE LEARNING (OVERVIEW)

[MITCHELL82]

- Let $X = \{x_1, \ldots, x_n\}$ a set of attributes of domains $D = \{D_1, \ldots, D_n\}$

- A concept is a Boolean function $f : X \to \{0,1\}$
  - $f(a) = 0 \Rightarrow a \in D^X$ is a negative instance
  - $f(b) = 1 \Rightarrow b \in D^X$ is a positive instance
  - A function of a fixed set of predicates $P_i(X)$, which can be evaluated over instances (ex: $f(X) \equiv P_1(X) \wedge \neg P_2(X) \vee P_3(X)$)
    - In particular, restrict attention to pure conjunctive concepts

- The space of all possible $f(X)$ is called the hypothesis space H
- Version space is the subspace of H consistent with the training set

VERSION SPACE LEARNING (OVERVIEW)

[MITCHELL82]

- Let $X = \{x_1, \ldots, x_n\}$ a set of attributes of domains $D = \{D_1, \ldots, D_n\}$

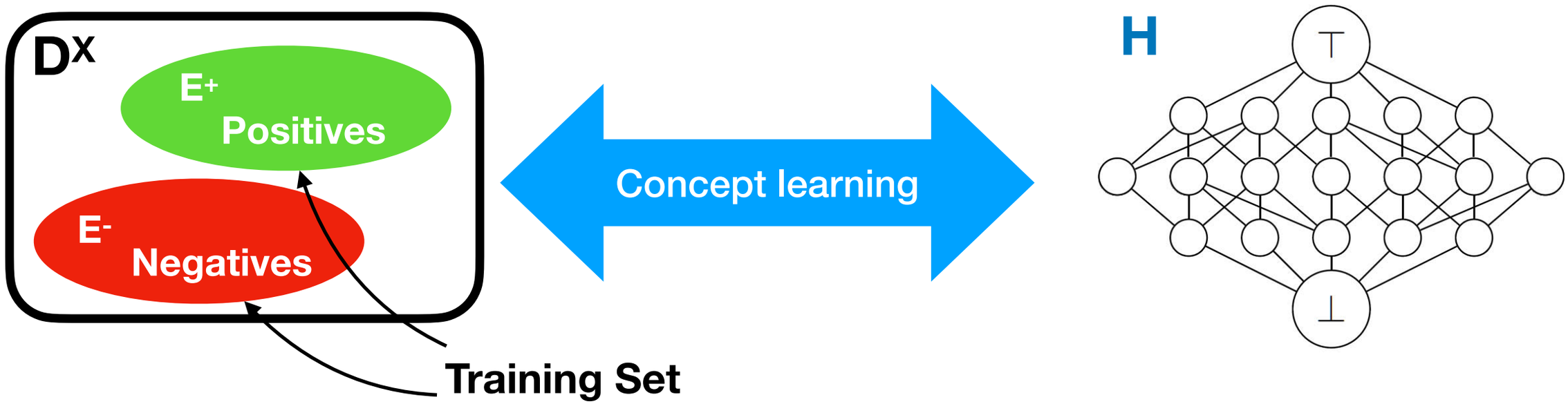
- A concept is a Boolean function $f : X \to \{0,1\}$
  - $f(a) = 0 \Rightarrow a \in D^X$ is a negative instance
  - $f(b) = 1 \Rightarrow b \in D^X$ is a positive instance
  - A function of a fixed set of predicates $P_i(X)$, which can be evaluated over instances (ex: $f(X) \equiv P_1(X) \wedge \neg P_2(X) \vee P_3(X)$)
    - In particular, restrict attention to pure conjunctive concepts

- The space of all possible $f(X)$ is called the hypothesis space H
- Version space is the subspace of H consistent with the training set

- Version space learning has the task to search for a $f(X)$ that correctly classifies a given set of positives and negatives.

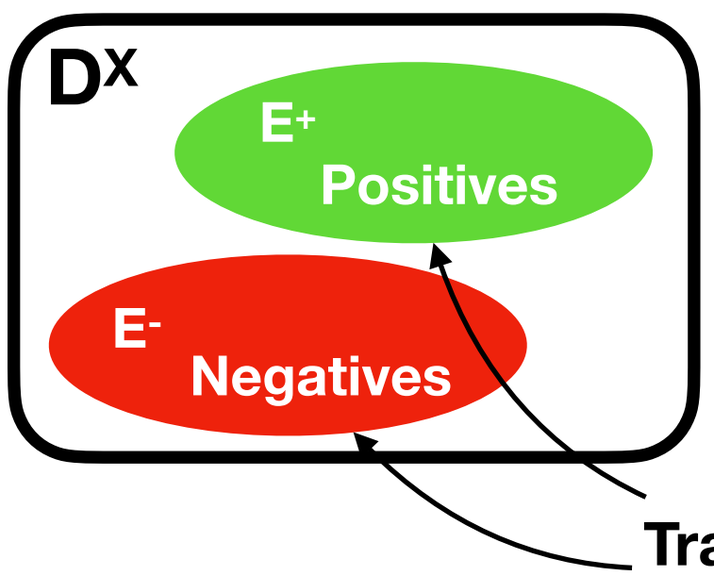
VERSION SPACE LEARNING (OVERVIEW)

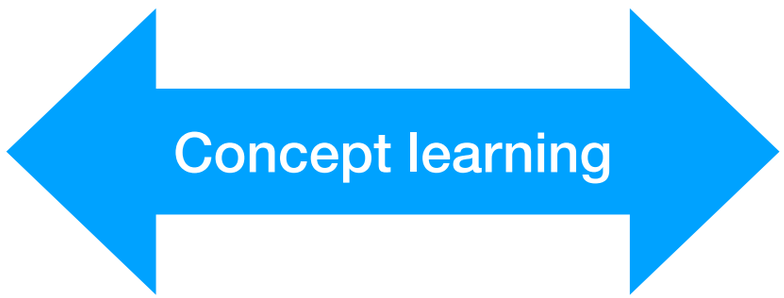
[MITCHELL82]

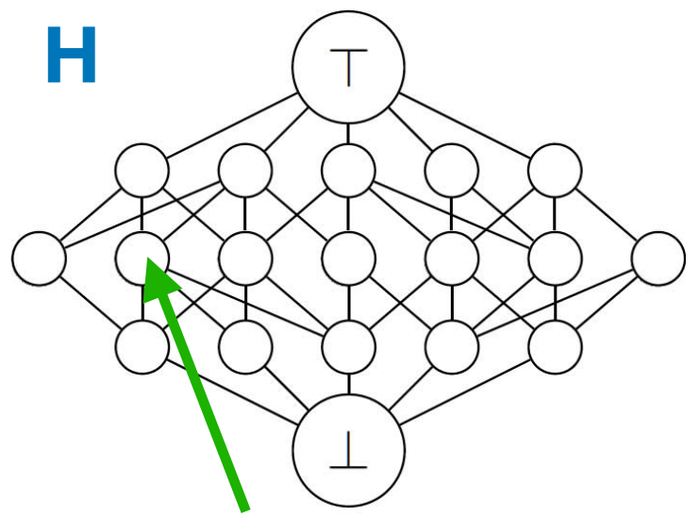


VERSION SPACE LEARNING (OVERVIEW)

[MITCHELL82]





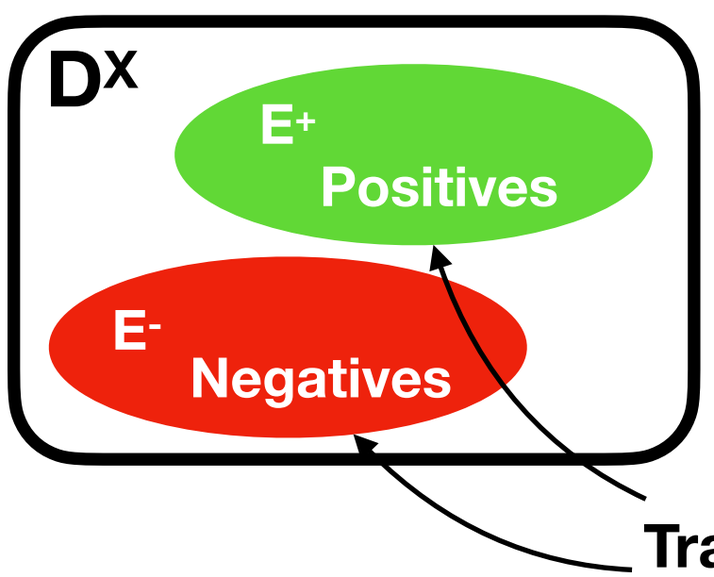


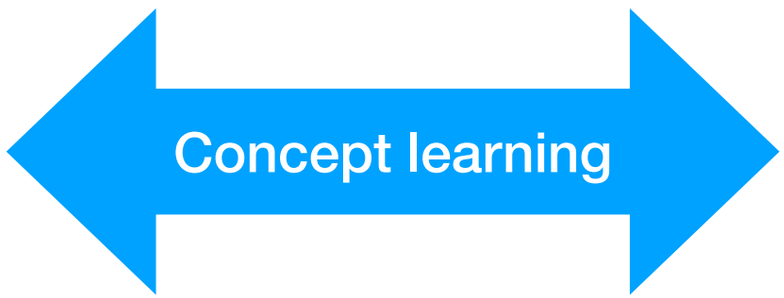
$f \equiv P_1 \wedge P_2 \wedge P_3$

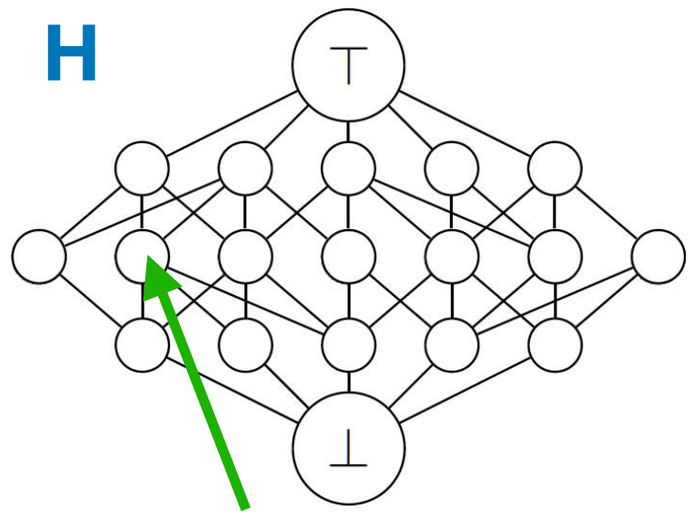
Version space

VERSION SPACE LEARNING (OVERVIEW)

[MITCHELL82]





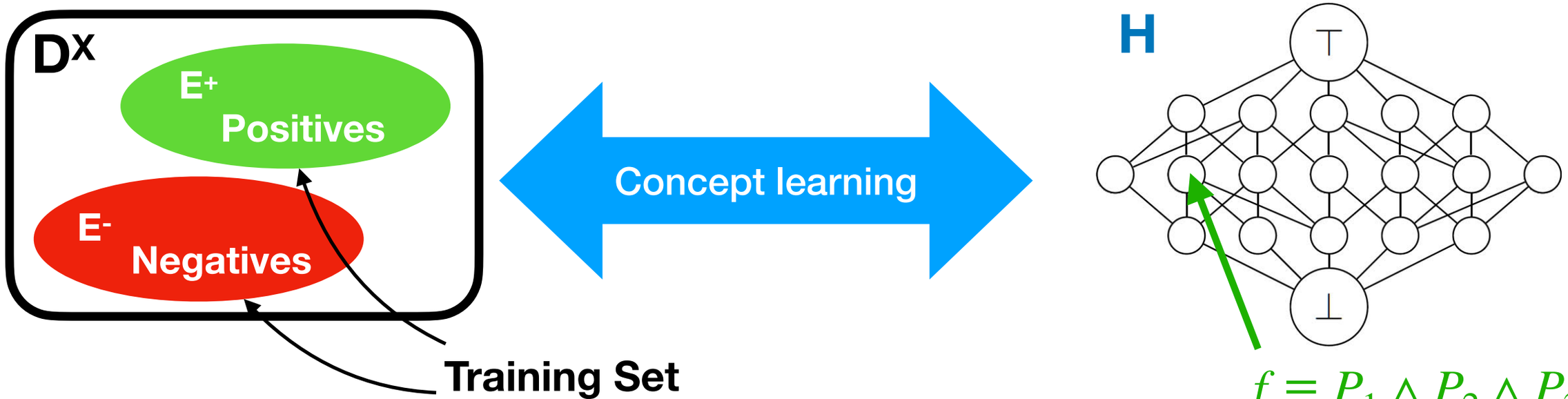


$f \equiv P_1 \wedge P_2 \wedge P_3$

Version space

VERSION SPACE LEARNING (OVERVIEW)

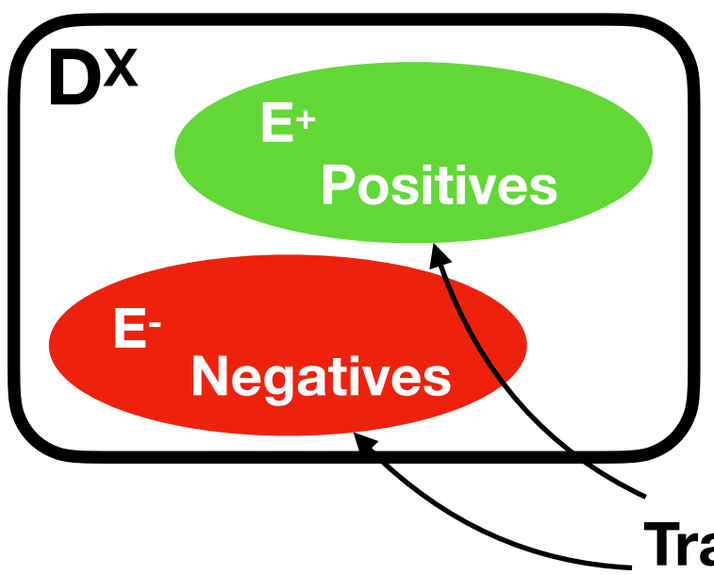
[MITCHELL82]

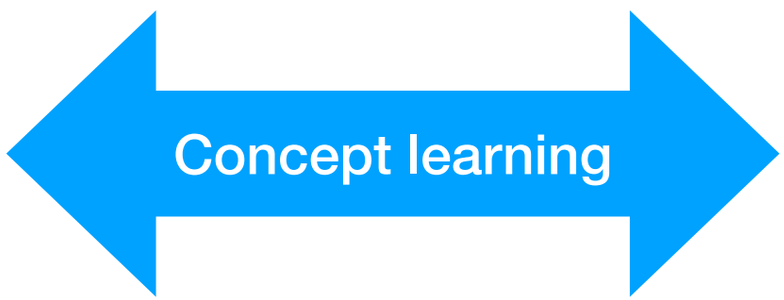


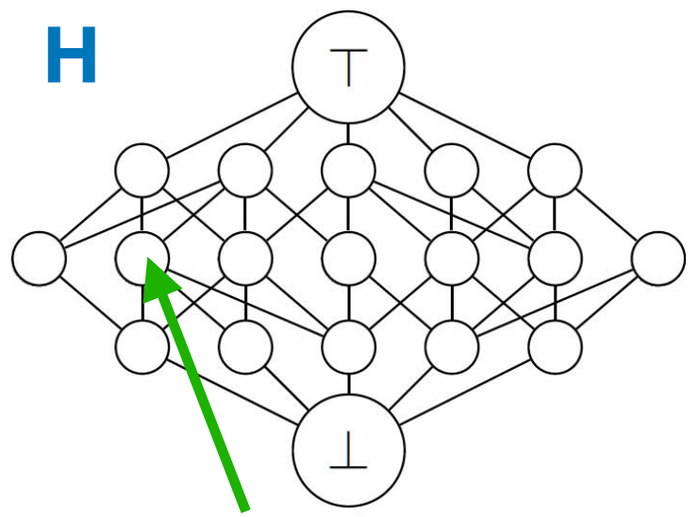
Constraint Programming:

VERSION SPACE LEARNING (OVERVIEW)

[MITCHELL82]



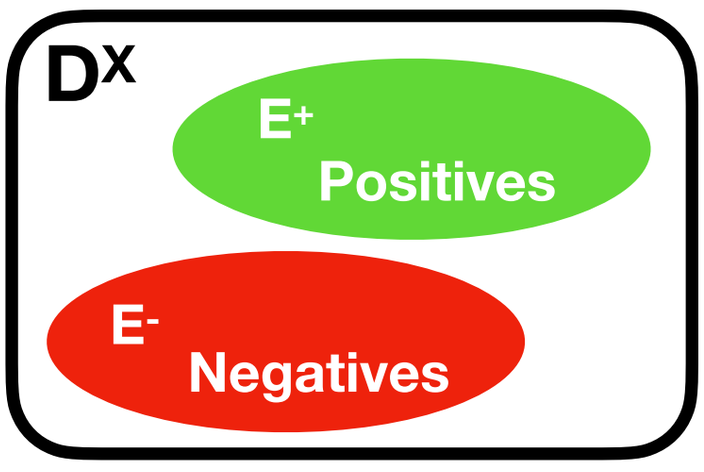




$f \equiv P_1 \wedge P_2 \wedge P_3$

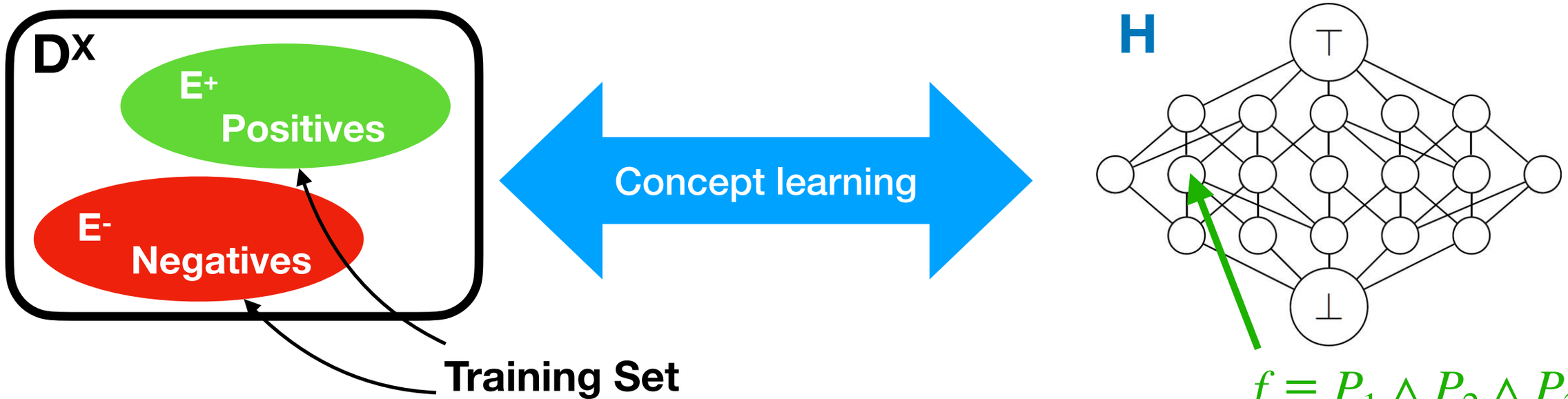
Version space

Constraint Programming:



VERSION SPACE LEARNING (OVERVIEW)

[MITCHELL82]



Constraint Programming:

VERSION SPACE LEARNING (EXAMPLE)

VERSION SPACE LEARNING (EXAMPLE)

- Attributes (variables):
  - **Size**: small, medium, large
  - **Shape:** triangle, square, circle

VERSION SPACE LEARNING (EXAMPLE)

- Attributes (variables):
  - **Size**: small, medium, large
  - **Shape:** triangle, square, circle

- Possible instances?
  - There are only **9** possible instances

VERSION SPACE LEARNING (EXAMPLE)

- Attributes (variables):
  - **Size**: small, medium, large
  - **Shape:** triangle, square, circle

- Possible instances?
  - There are only **9** possible instances

- Max number of concepts?
  - We can have $\mathbf{2^9}$ different concepts
    - Size= small
      - (3 instances)
    - Shape != square
      - (6 instances)
    - (Size=large) and ((Shape=square) or (Shape=circle))
      - (2 instances)

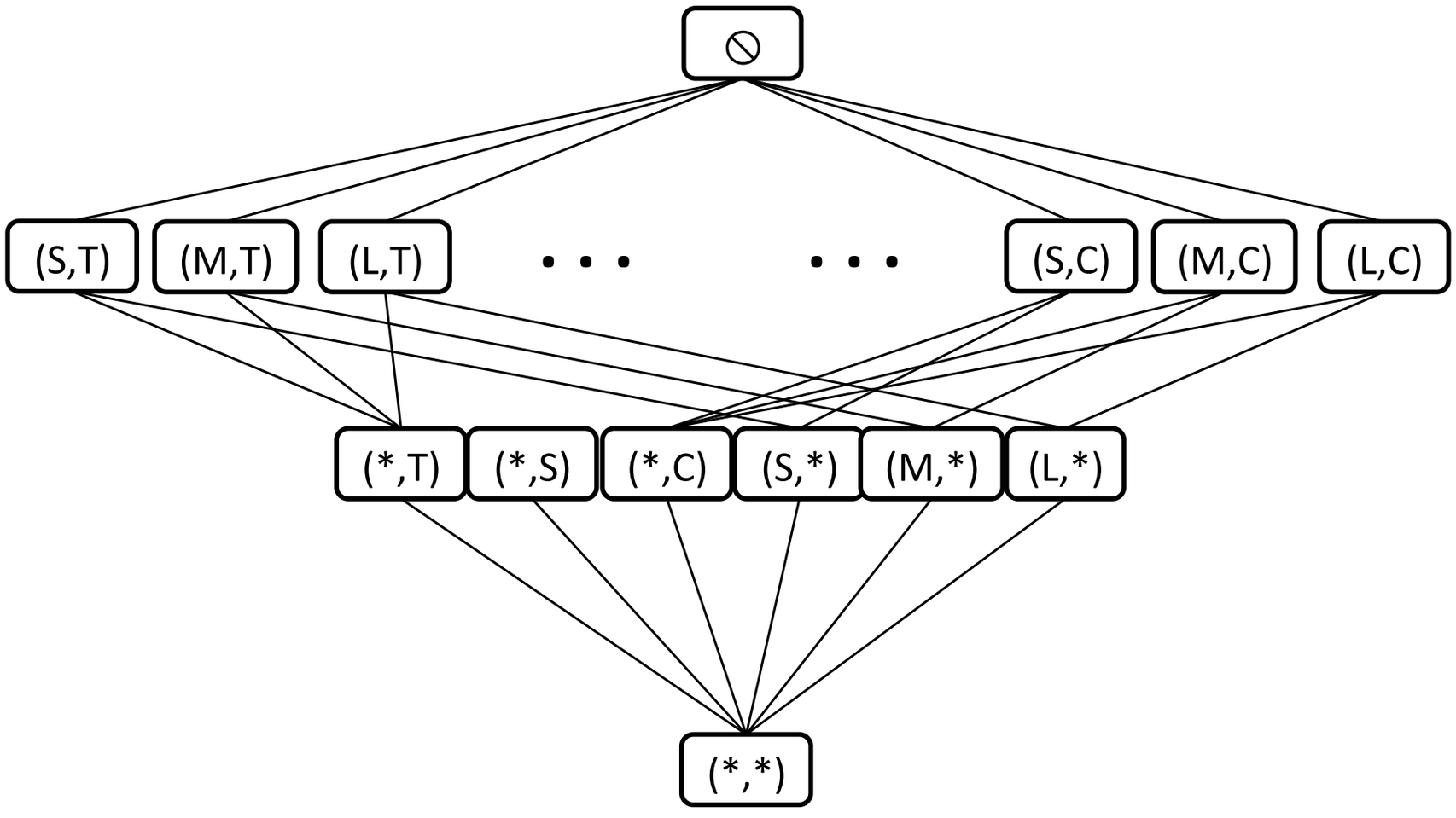
VERSION SPACE LEARNING (EXAMPLE)

VERSION SPACE LEARNING (EXAMPLE)

(Size,Shape)= ([S,M,L],[T,S,C])

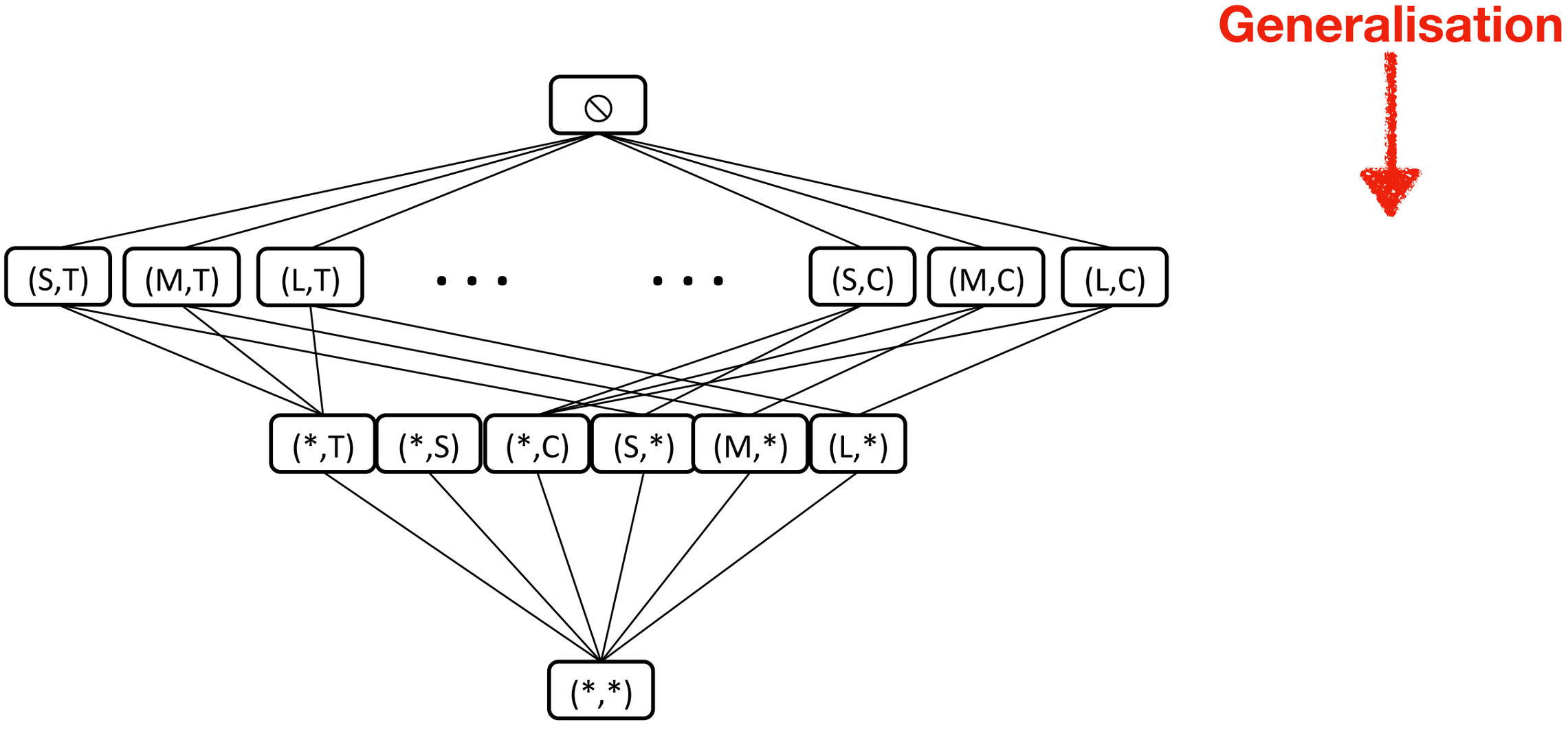
VERSION SPACE LEARNING (EXAMPLE)

(Size,Shape)= ([S,M,L],[T,S,C])



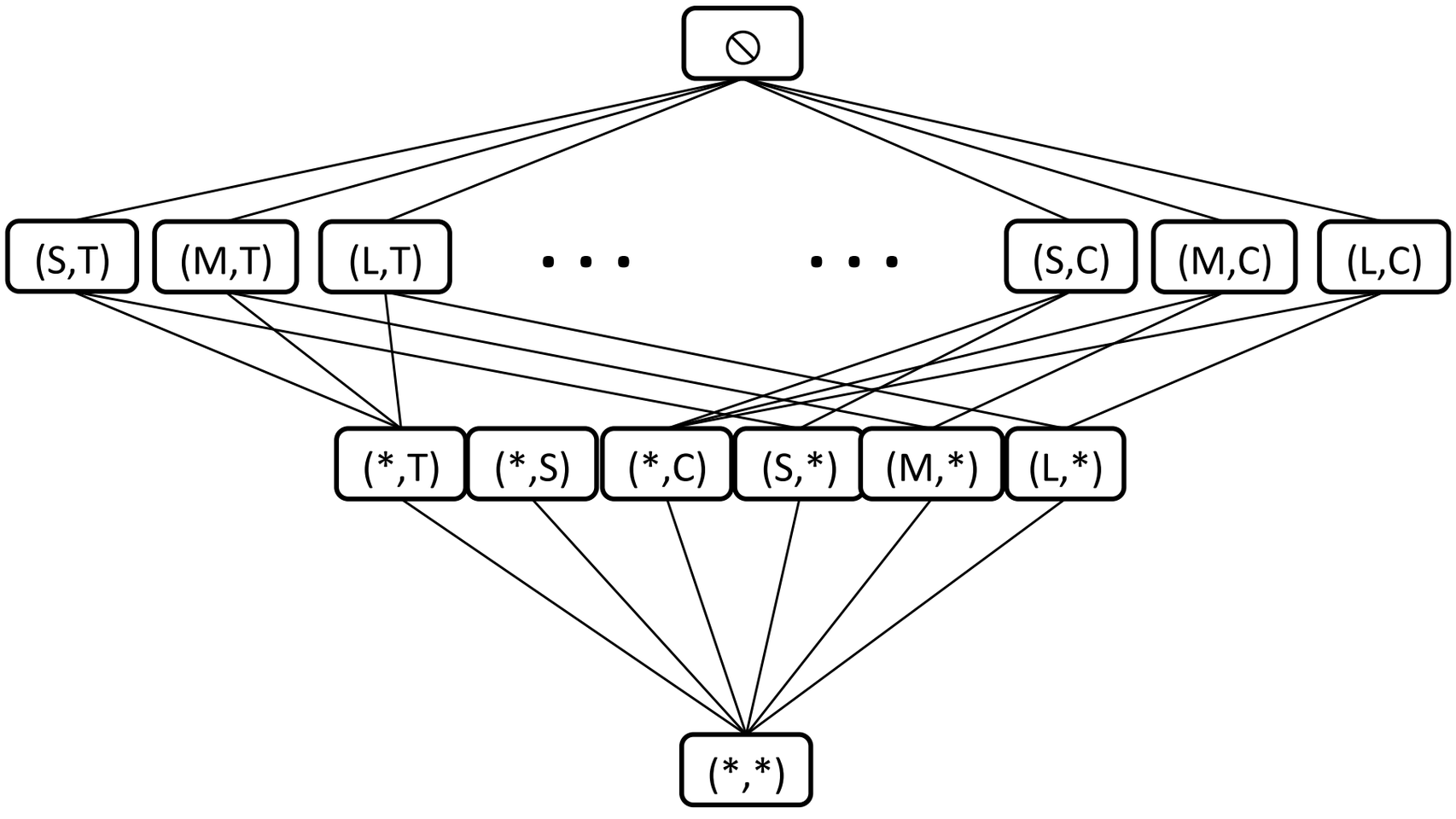
VERSION SPACE LEARNING (EXAMPLE)

(Size,Shape)= ([S,M,L],[T,S,C])



VERSION SPACE LEARNING (EXAMPLE)

(Size,Shape)= ([S,M,L],[T,S,C])



Generalisation

Specialisation

RESTRICTING THE HYPOTHESIS SPACE

RESTRICTING THE HYPOTHESIS SPACE

- Have lattice structure for the entire space of all possible concepts over this instance space (= the 512 possible subsets of the previous example)

RESTRICTING THE HYPOTHESIS SPACE

- Have lattice structure for the entire space of all possible concepts over this instance space (= the 512 possible subsets of the previous example)

- Let's assume that our hypothesis space is something smaller

RESTRICTING THE HYPOTHESIS SPACE

- Have lattice structure for the entire space of all possible concepts over this instance space (= the 512 possible subsets of the previous example)

- Let's assume that our hypothesis space is something smaller

- In particular, restrict attention to pure conjunctive concepts

RESTRICTING THE HYPOTHESIS SPACE

- Have lattice structure for the entire space of all possible concepts over this instance space (= the 512 possible subsets of the previous example)

- Let's assume that our hypothesis space is something smaller

- In particular, restrict attention to pure conjunctive concepts

- A pure conjunctive concept is one that has a pure conjunctive concept description

RESTRICTING THE HYPOTHESIS SPACE

- Have lattice structure for the entire space of all possible concepts over this instance space (= the 512 possible subsets of the previous example)

- Let's assume that our hypothesis space is something smaller

- In particular, restrict attention to pure conjunctive concepts

- A pure conjunctive concept is one that has a pure conjunctive concept description

- A pure conjunctive concept description is one whose only logical operators are conjunctions (ANDs)

RESTRICTING THE HYPOTHESIS SPACE

- Have lattice structure for the entire space of all possible concepts over this instance space (= the 512 possible subsets of the previous example)

- Let's assume that our hypothesis space is something smaller

- In particular, restrict attention to pure conjunctive concepts

- A pure conjunctive concept is one that has a pure conjunctive concept description

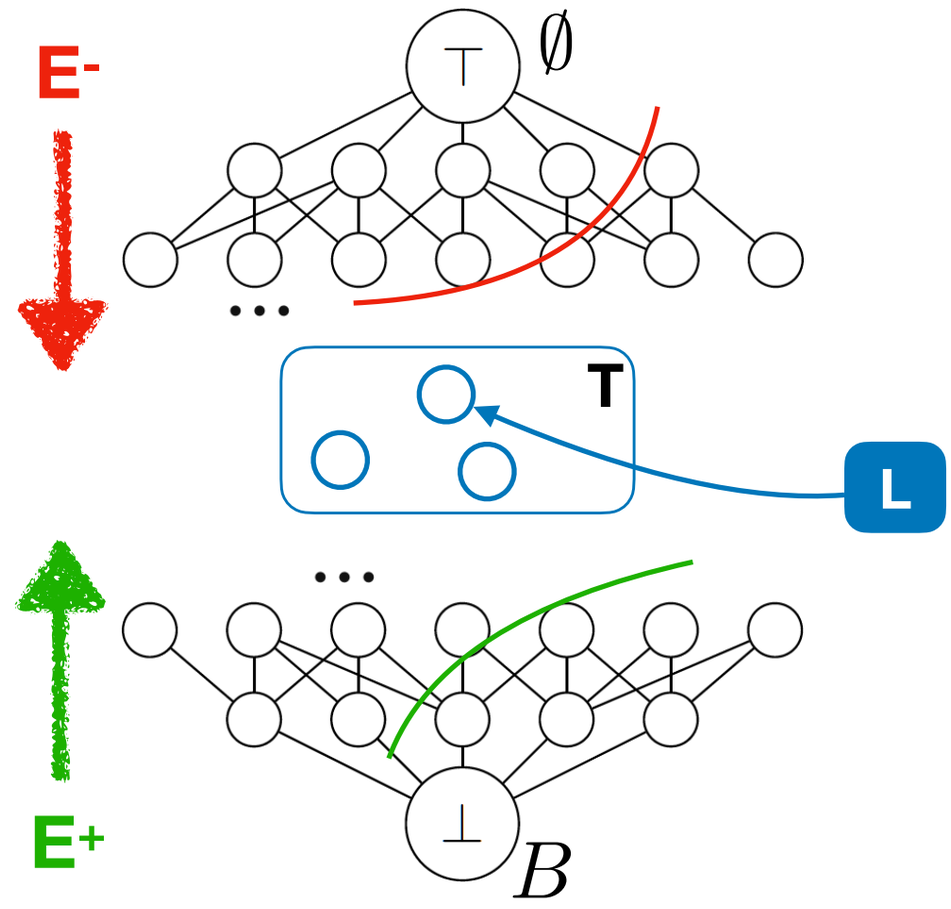
- A pure conjunctive concept description is one whose only logical operators are conjunctions (ANDs)

- No disjunctions (ORs) or negations (NOTs) are allowed

CONSTRAINT ACQUISITION PROBLEM

- Inputs:
  - (X,D): Vocabulary
  - Γ: Constraint language
    - B: Basis (constraints/hypothesis)
  - T: Target Network (concept to learn)
  - E=(E⁺,E⁻): training data

- Output:
- L: Learned network



CONSTRAINT ACQUISITION PROBLEM

- Inputs:
  - (X,D): Vocabulary
  - Γ: Constraint language
    - B: Basis (constraints/hypothesis)
  - T: Target Network (concept to learn)
  - E=(E⁺,E⁻): training data

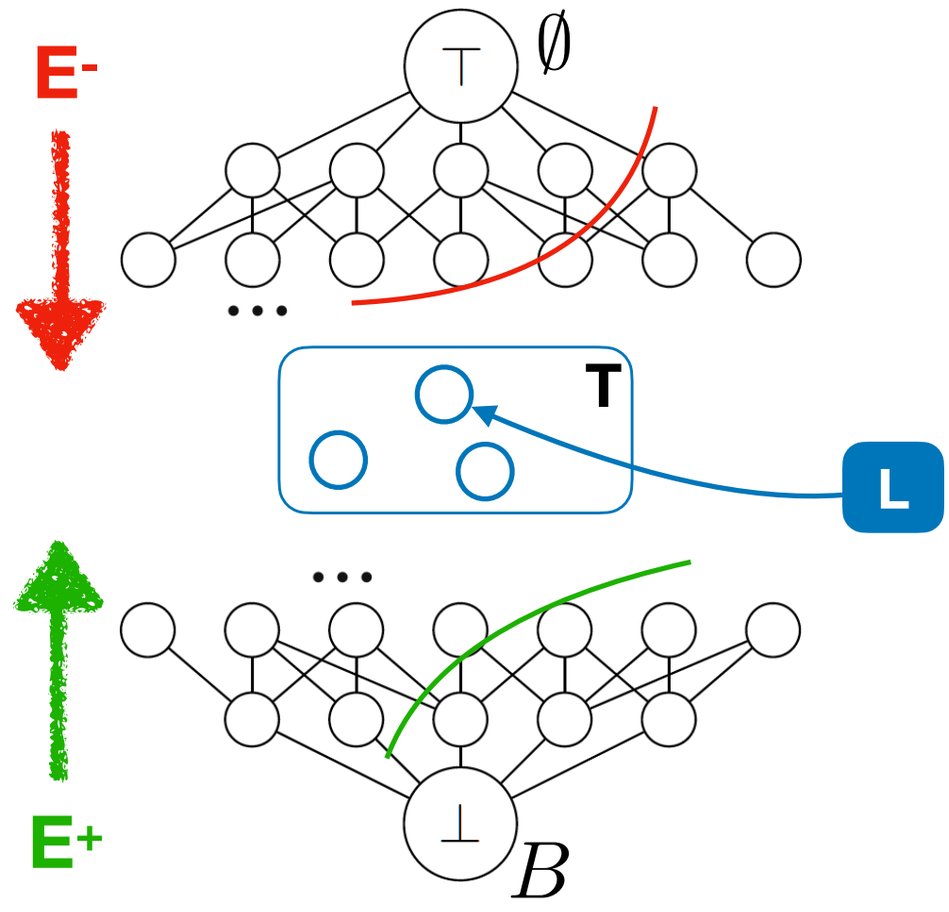
- Output:
- L: Learned network

Convergence Problem:

- L agrees with E
- For any other network $C' \subseteq B$

agreeing with E, we have:

$$sol(C') = sol(L)$$



CONSTRAINT ACQUISITION PROBLEM

- Inputs:
  - (X,D): Vocabulary
  - Γ: Constraint language
    - B: Basis (constraints/hypothesis)
  - T: Target Network (concept to learn)
  - E=(E⁺,E⁻): training data

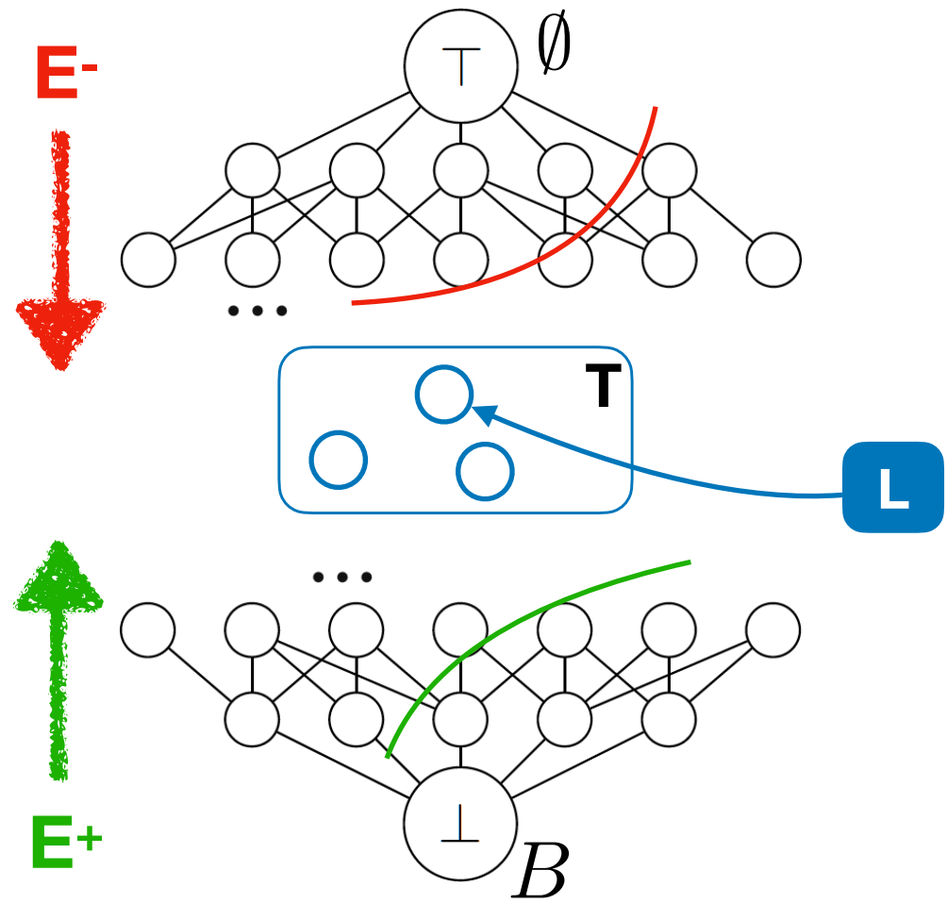
- Output:
- L: Learned network

Convergence Problem:

- L agrees with E
- For any other network $C' \subseteq B$

agreeing with E, we have:

$$sol(C') = sol(L)$$



coNP-Complete

[Bessiere, Koriche, Lazaar, O'Sullivan, AIJ17]

EXAMPLE

- **INPUT:**
  - $X = \{x_1, x_2, x_3\}$
  - $D(x_i) = \{1,2,3\}$
  - $\Gamma = \{ <, = \}$
  - $B = \{x_i < x_j, x_i = x_j, \forall i, j\}$
  - $T \equiv \{(1,2,1), (2,3,2)\}$

EXAMPLE

- **INPUT:**
  - $X = \{x_1, x_2, x_3\}$
  - $D(x_i) = \{1,2,3\}$
  - $\Gamma = \{ <, = \}$
  - $B = \{x_i < x_j, x_i = x_j, \forall i, j\}$
  - $T \equiv \{(1,2,1), (2,3,2)\}$

- **OUTPUT:**
  - $L_1 = \{x_1 = x_3, x_1 < x_2\}$

OR

  - $L_2 = \{x_1 = x_3, x_3 < x_2\}$

Equivalence class

EXAMPLE

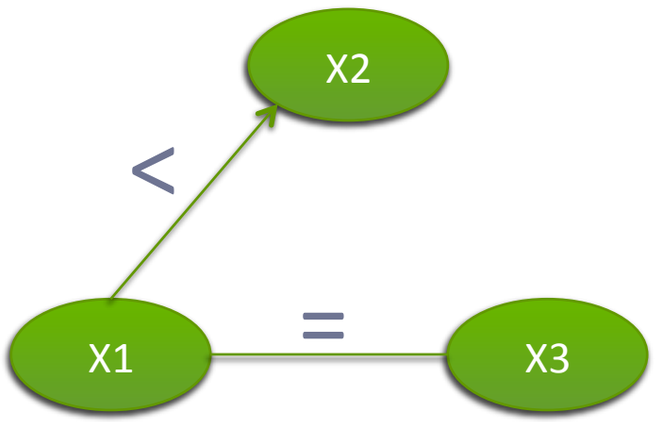
- **INPUT:**
  - $X = \{x_1, x_2, x_3\}$
  - $D(x_i) = \{1,2,3\}$
  - $\Gamma = \{ <, = \}$
  - $B = \{x_i < x_j, x_i = x_j, \forall i, j\}$
  - $T \equiv \{(1,2,1), (2,3,2)\}$

- **OUTPUT:**
  - $L_1 = \{x_1 = x_3, x_1 < x_2\}$

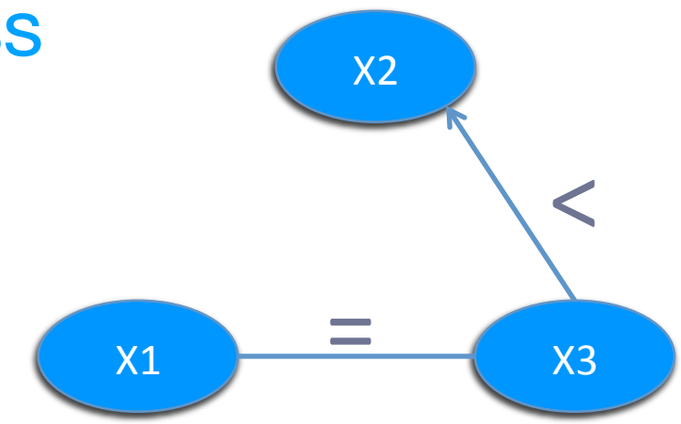
OR

  - $L_2 = \{x_1 = x_3, x_3 < x_2\}$

Equivalence class

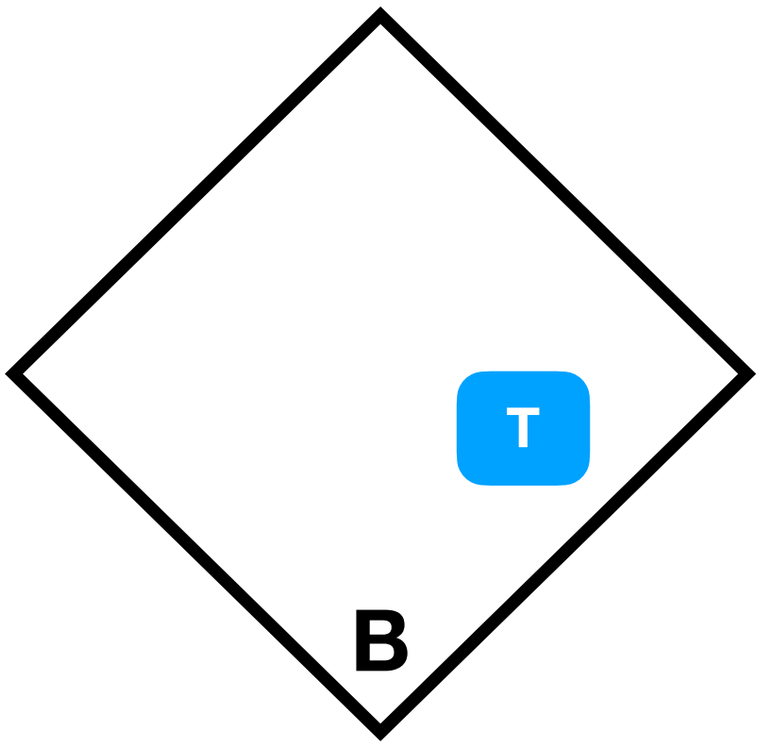


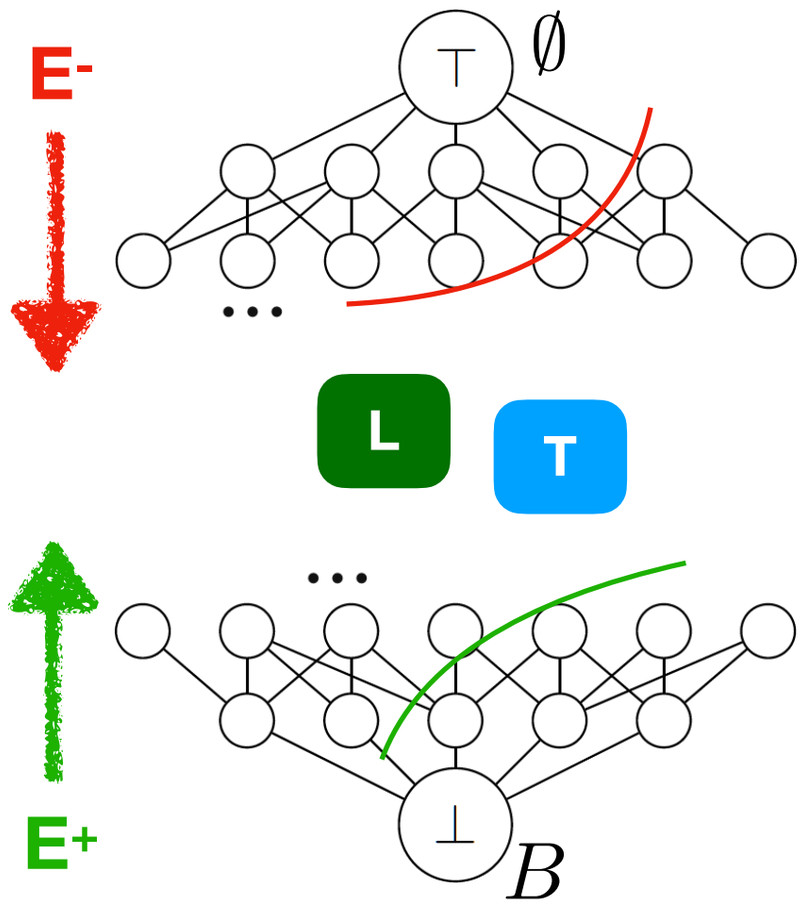
L_1



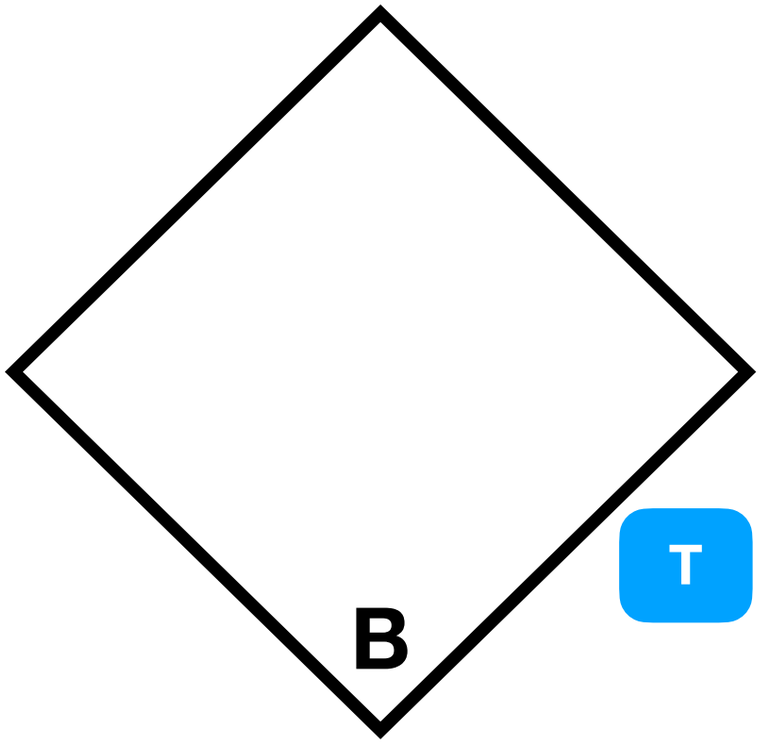
L_2

CONSTRAINT ACQUISITION PROBLEM

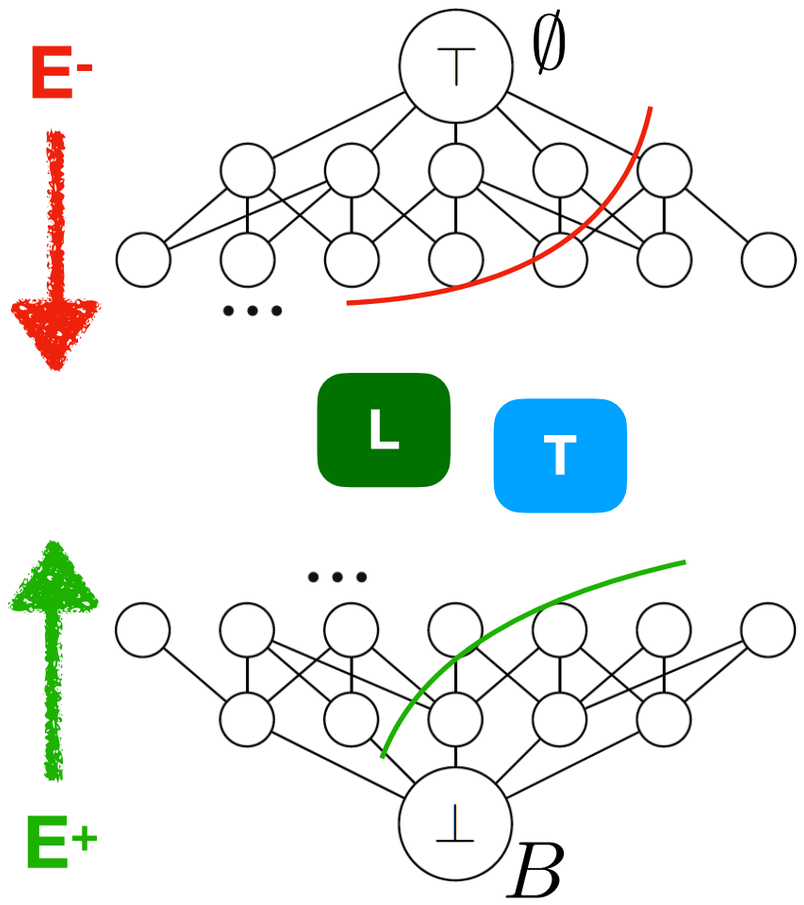




CONSTRAINT ACQUISITION PROBLEM



Collapse!

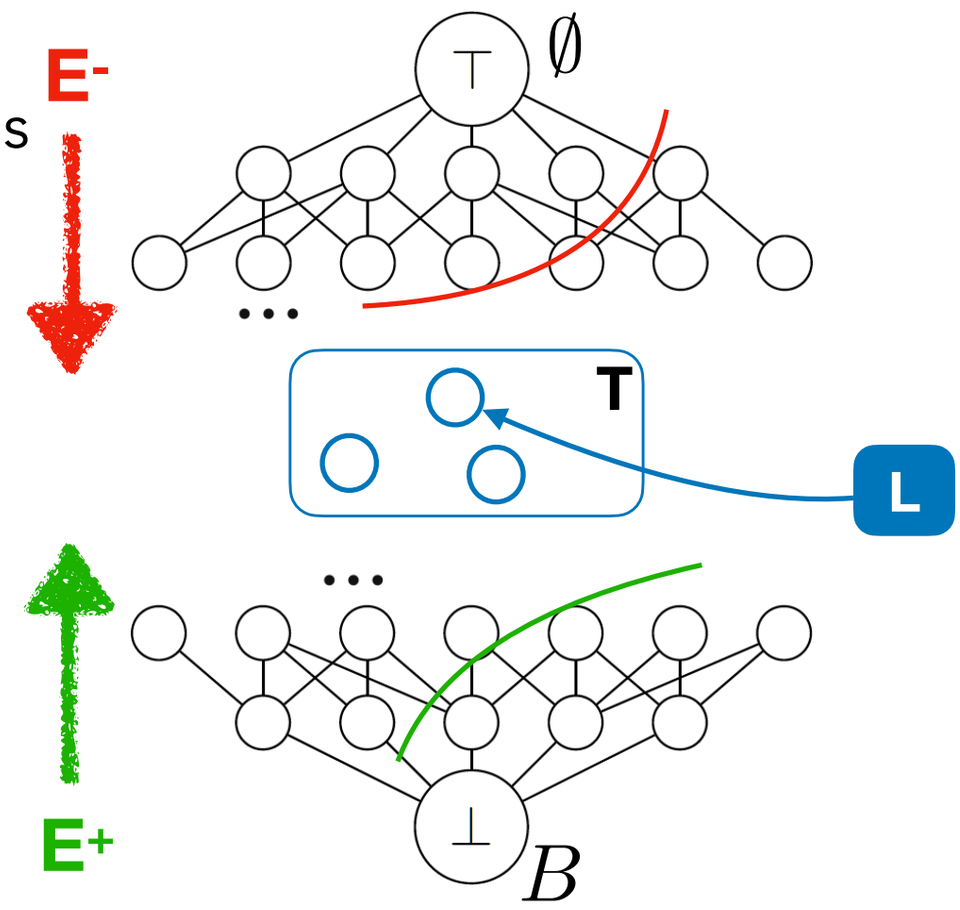


CONACQ

ACQUISITION USING MEMBERSHIP QUERIES

CONACQ [COCONUT These 2006; Bessiere, Koriche, Lazaar, O'Sullivan, AIJ17]

- SAT-Based constraint acquisition
- Bidirectional inference using Membership queries
- Conacq1.0 (passive learning)
- Conacq2.0 (active learning)
- Connexion between literals l_i and constraints C_i
  - $l_i = 1 \Rightarrow C_i \in L$



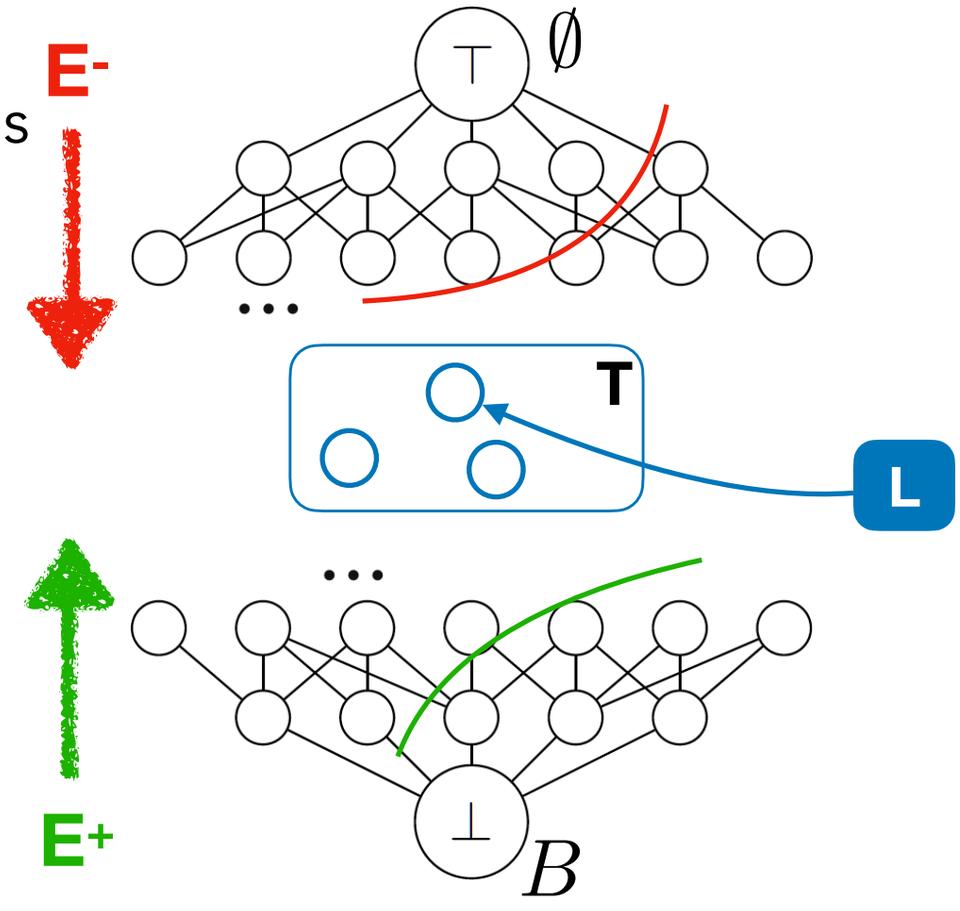
ACQUISITION USING MEMBERSHIP QUERIES

CONACQ [COCONUT These 2006; Bessiere, Koriche, Lazaar, O'Sullivan, AIJ17]

- SAT-Based constraint acquisition
- Bidirectional inference using Membership queries
- Conacq1.0 (passive learning)
- Conacq2.0 (active learning)
- Connexion between literals l_i and constraints C_i
  - $l_i = 1 \Rightarrow C_i \in L$

$$\Omega = (\neg x_2 \wedge \neg x_4 \wedge \neg x_7)$$

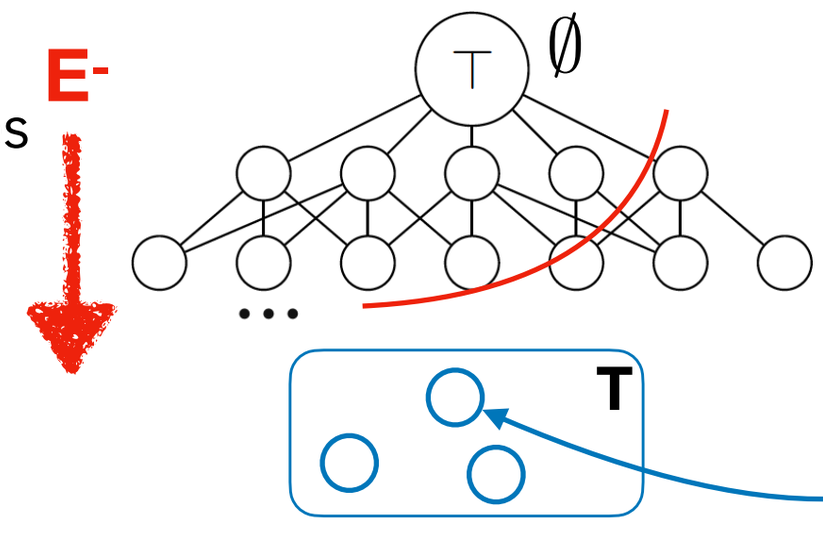
Positive example
Bottom-up inference



ACQUISITION USING MEMBERSHIP QUERIES

CONACQ [COCONUT These 2006; Bessiere, Koriche, Lazaar, O'Sullivan, AIJ17]

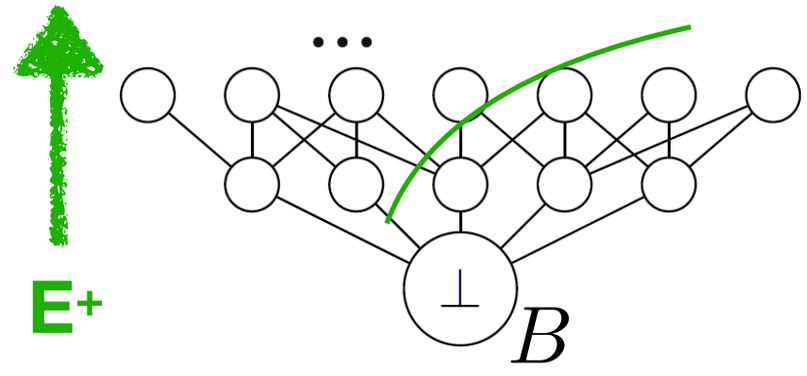
- SAT-Based constraint acquisition
- Bidirectional inference using Membership queries
- Conacq1.0 (passive learning)
- Conacq2.0 (active learning)
- Connexion between literals l_i and constraints C_i
  - $l_i = 1 \Rightarrow C_i \in L$



$$\Omega = \underbrace{(\neg x_2 \wedge \neg x_4 \wedge \neg x_7)}_{\text{Positive example}} \bigwedge \underbrace{(x_1 \vee x_3 \vee x_5 \vee x_8)}_{\text{Negative example}} \ldots$$

Positive example
Bottom-up inference

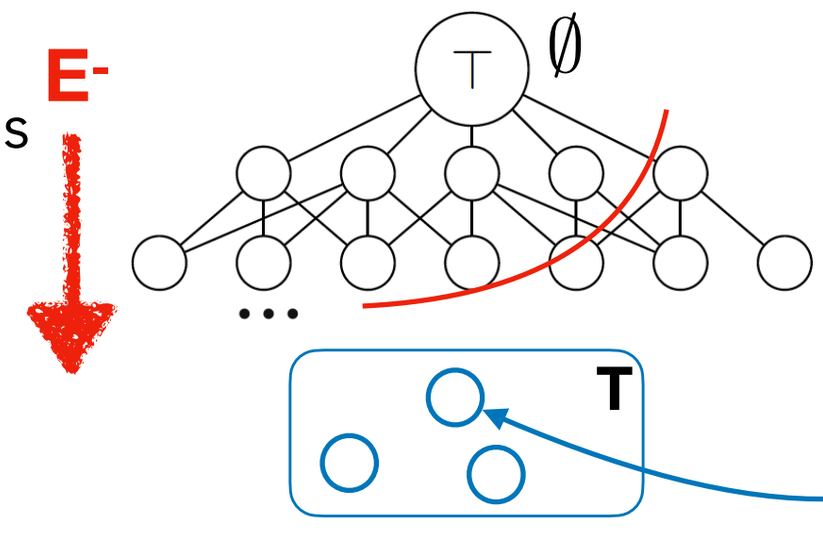
Negative example
Top-down inference



ACQUISITION USING MEMBERSHIP QUERIES

CONACQ [COCONUT These 2006; Bessiere, Koriche, Lazaar, O'Sullivan, AIJ17]

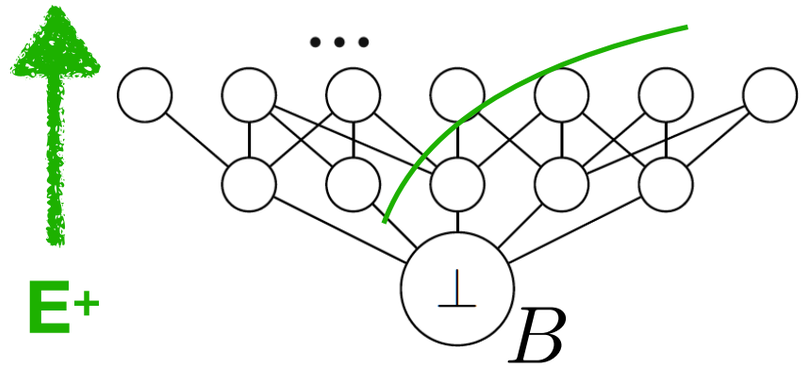
- SAT-Based constraint acquisition
- Bidirectional inference using Membership queries
- Conacq1.0 (passive learning)
- Conacq2.0 (active learning)
- Connexion between literals l_i and constraints C_i
  - $l_i = 1 \Rightarrow C_i \in L$



$$\Omega = \underbrace{(\neg x_2 \wedge \neg x_4 \wedge \neg x_7)}_{\text{Positive example}} \bigwedge \underbrace{(x_1 \vee x_3 \vee x_5 \vee x_8)}_{\text{Negative example}} \ldots$$

Positive example
Bottom-up inference

Negative example
Top-down inference



Non-learnability using Membership queries [AIJ17]

CONACQ.1 (PASSIVE LEARNING)

```
Foreach example e in TrainingSet:

   Identify constraints in B rejecting e

   If e is negative
      Top-down-inference()

   Else
      Bottom-up–inference()
```

CONACQ.1 (PASSIVE LEARNING)

Algorithm 1: The CONACQ.1 Algorithm

Input: a basis $\mathbf{B}$ and a training set E
Output: a clausal theory Ω encoding $\mathbf{C_B}(E)$, a Boolean value v saying if convergence is reached

1 $\Omega \leftarrow \varnothing$
2 **foreach** *example* $e \in E$ **do**
3 $\quad \kappa(e) \leftarrow \{c \in \mathbf{B} \mid \boldsymbol{x}(e)$ violates $c\}$
4 $\quad$ **if** $y(e) = 0$ **then** $\Omega \leftarrow \Omega \wedge \left(\bigvee_{c \in \kappa(e)} a(c)\right)$
5 $\quad$ **if** $y(e) = 1$ **then** $T \leftarrow \Omega \wedge \bigwedge_{c \in \kappa(e)} \neg a(c)$
6 $\quad$ **if** Ω *is unsatisfiable* **then return** *"collapse"*
7 $v \leftarrow$ CONVERGENCE(Ω)
8 **return** (Ω, v)

CONACQ.1 (PASSIVE LEARNING)

Algorithm 1: The CONACQ.1 Algorithm

Input: a basis $\mathbf{B}$ and a training set E
Output: a clausal theory Ω encoding $\mathbf{C_B}(E)$, a Boolean value v saying if convergence is reached

1 $\Omega \leftarrow \varnothing$
2 **foreach** *example* $e \in E$ **do**
3 $\quad \kappa(e) \leftarrow \{c \in \mathbf{B} \mid \boldsymbol{x}(e)$ violates $c\}$
4 $\quad$ **if** $y(e) = 0$ **then** $\Omega \leftarrow \Omega \wedge \left(\bigvee_{c \in \kappa(e)} a(c)\right)$
5 $\quad$ **if** $y(e) = 1$ **then** $T \leftarrow \Omega \wedge \bigwedge_{c \in \kappa(e)} \neg a(c)$
6 $\quad$ **if** Ω *is unsatisfiable* **then return** *"collapse"*
7 $v \leftarrow$ CONVERGENCE(Ω)
8 **return** (Ω, v)

Top-down inference

CONACQ.1 (PASSIVE LEARNING)

Algorithm 1: The CONACQ.1 Algorithm

Input: a basis $\mathbf{B}$ and a training set E
Output: a clausal theory Ω encoding $\mathbf{C_B}(E)$, a Boolean value v saying if convergence is reached

1 $\Omega \leftarrow \varnothing$
2 **foreach** *example* $e \in E$ **do**
3 $\quad \kappa(e) \leftarrow \{c \in \mathbf{B} \mid \boldsymbol{x}(e) \text{ violates } c\}$
4 $\quad$ **if** $y(e) = 0$ **then** $\Omega \leftarrow \Omega \wedge \left(\bigvee_{c \in \kappa(e)} a(c)\right)$
5 $\quad$ **if** $y(e) = 1$ **then** $T \leftarrow \Omega \wedge \bigwedge_{c \in \kappa(e)} \neg a(c)$
6 $\quad$ **if** Ω *is unsatisfiable* **then return** *"collapse"*
7 $v \leftarrow$ CONVERGENCE(Ω)
8 **return** (Ω, v)

Top-down inference

Bottom-up inference

CONACQ.2 (ACTIVE LEARNING)

```
while not converged do

   Generate an ‘informative’ query

   If no-query then ‘converge!’

   If query is negative
      Top-down-inference()

   Else
      Bottom-up–inference()
```

CONACQ.2 (ACTIVE LEARNING)

Algorithm 2 : The CONACQ.2 Algorithm

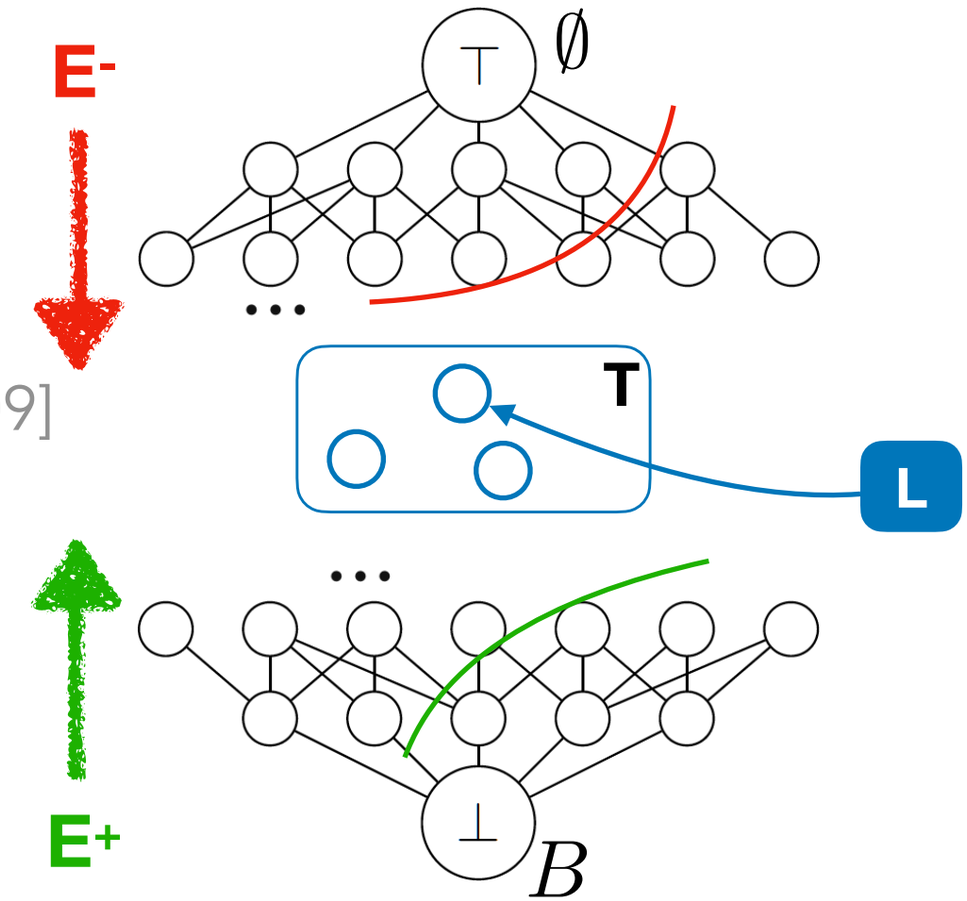
Input: a basis **B**, a background knowledge K , a strategy *Strategy*
Output: a clausal theory Ω encoding the target network

1 $\Omega \leftarrow \varnothing$; $converged \leftarrow false$; $N \leftarrow \varnothing$
2 **while** $\neg converged$ **do**
3 $\quad q \leftarrow \text{QUERYGENERATION}(\mathbf{B}, \Omega, K, N, Strategy)$
4 $\quad$ **if** $q = nil$ **then** $converged \leftarrow true$
5 $\quad$ **else**
6 $\quad\quad$ **if** $\text{ASK}(q) = no$ **then** $\Omega \leftarrow \Omega \wedge \left(\bigvee_{c \in \kappa(q)} a(c)\right)$
7 $\quad\quad$ **else** $\Omega \leftarrow \Omega \wedge \bigwedge_{c \in \kappa(q)} \neg a(c)$
8 **return** Ω

More Complexe Systems

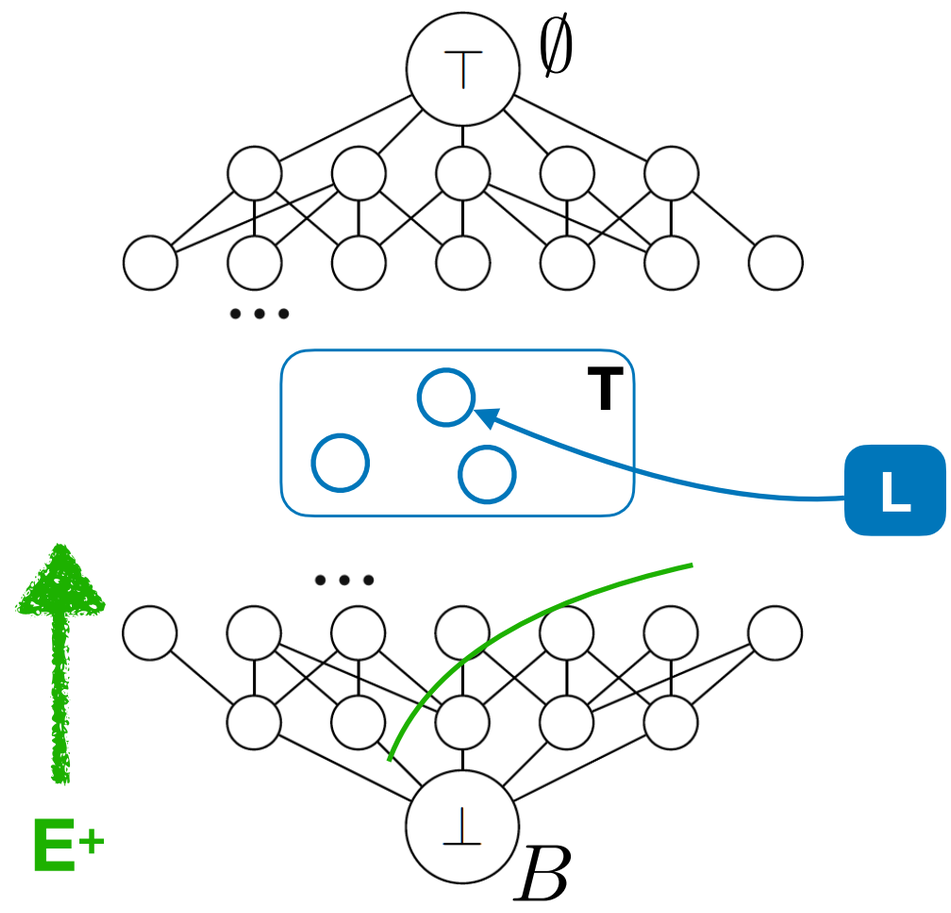
ACQUISITION USING COMPLEX QUERIES

- Matchmaker agents [Freuder and Wallace wAAAI97]
- Argument-Based CONACQ [Friedrich et al. ICDM'09]
- ILP-Based Acquisition [Lallouet et al. CP'10]



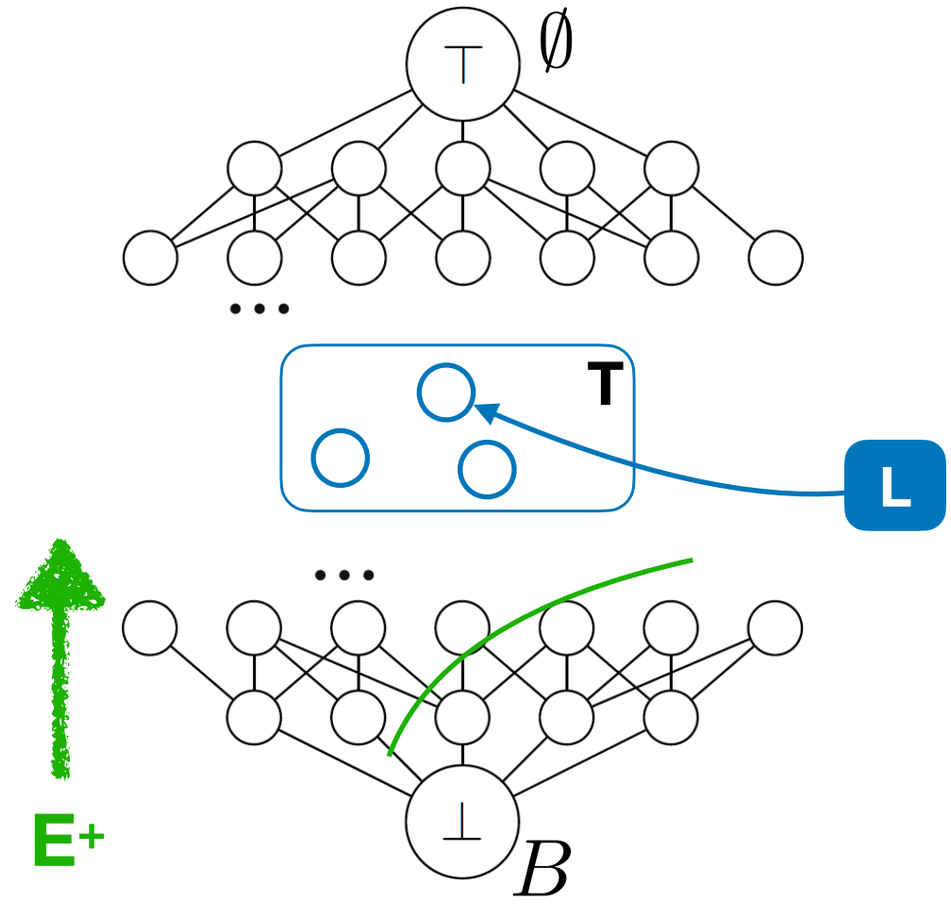
STRUCTURED PROBLEM ACQUISITION

- ModelSeeker [Beldiceanu and Simonis, CP'11'12]
  - A passive learning
  - Based on global constraint catalogue (≈1000)
  - Buttom-up inference
  - ModelSeeker learns constraints underlying the scheduling of the Bundesliga (the German Football Liga) from just **one** example.



STRUCTURED PROBLEM ACQUISITION

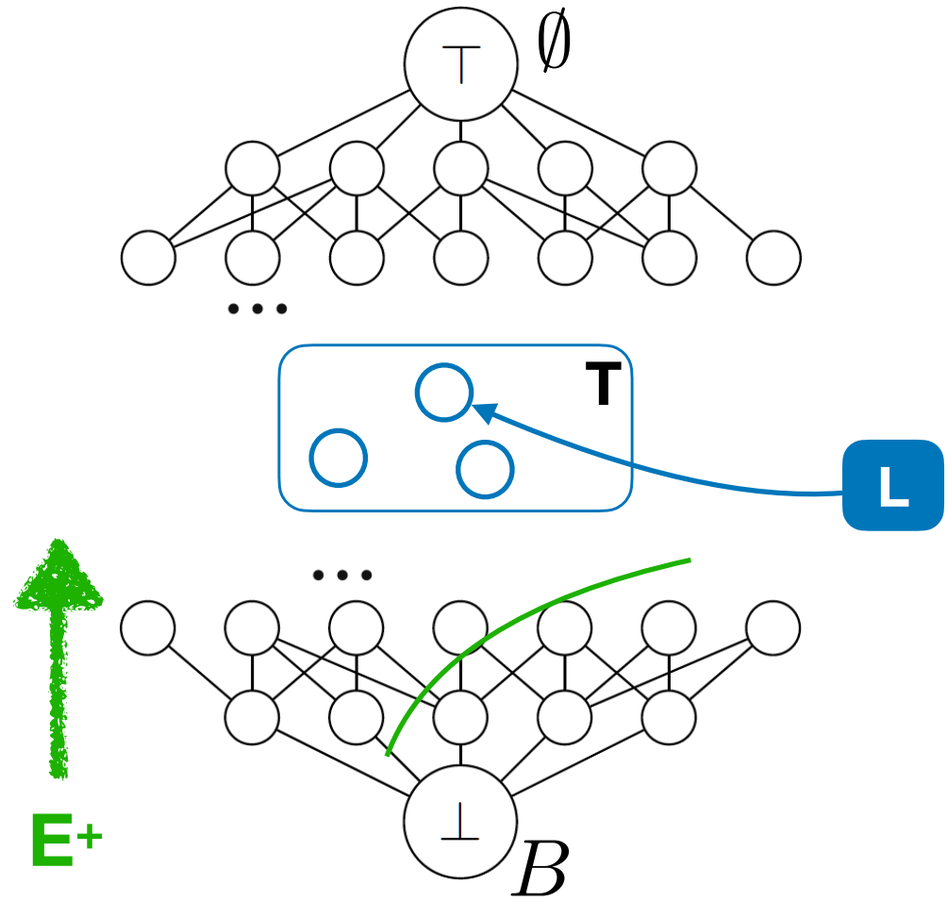
- ModelSeeker [Beldiceanu and Simonis, CP'11'12]
  - A passive learning
  - Based on global constraint catalogue (≈1000)
  - Buttom-up inference
  - ModelSeeker learns constraints underlying the scheduling of the Bundesliga (the German Football Liga) from just **one** example.



1						5	2	
				7	8			
						6		
	9			4				
			5			1		
	7							
		6	2					
	4						7	8
								3

STRUCTURED PROBLEM ACQUISITION

- ModelSeeker [Beldiceanu and Simonis, CP'11'12]
  - A passive learning
  - Based on global constraint catalogue (≈1000)
  - Buttom-up inference
  - ModelSeeker learns constraints underlying the scheduling of the Bundesliga (the German Football Liga) from just **one** example.



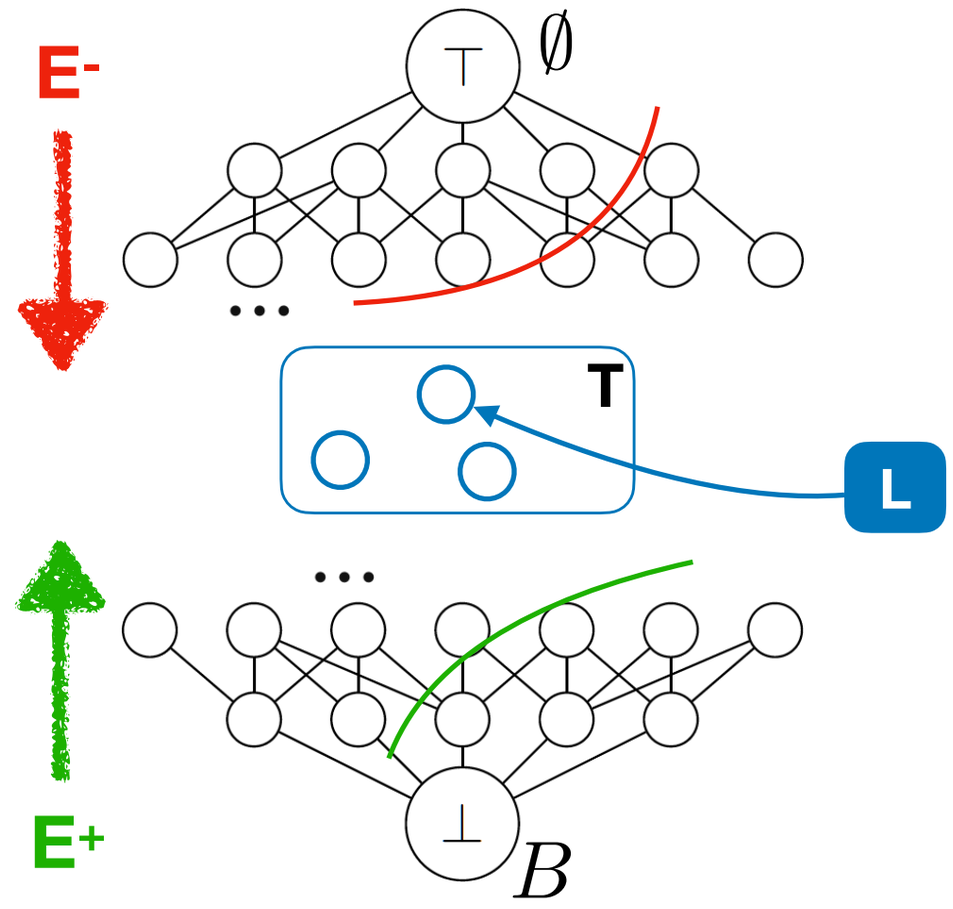
1						5	2	
				7	8			
						6		
	9			4				
			5			1		
	7							
		6	2					
	4						7	8
								3

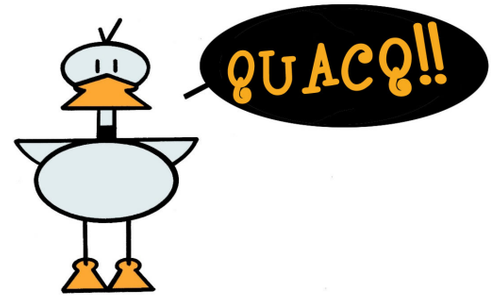
2	4	9			3			
	8			1	2	4		
						9		
						7	2	4
	1						3	
3	9	4						
		8						
		6	4	5			9	
			1			8	6	5

QUACQ

QUACQ: QUICK ACQUISITION

- QUACQ [Bessiere et al. 2013, 2020]
  - Active learning approach
  - Bidirectional inference
    - Top-down only if no positive example
  - Based on **partial** queries to elucidate the scope of the constraint to learn
  - Asymptotically optimal number of queries





MEMBERSHIP QUERIES

ask(

5	2	8	7	3	4	9	6	1
1	6	9	1	2	5	3	7	8
3	7	4	8	6	9	4	2	5
1	4	2	6	9	8	7	5	3
7	9	6	3	5	2	8	1	4
8	5	3	4	7	1	2	9	6
6	8	5	2	4	7	1	3	9
9	1	7	5	8	3	6	4	2
2	3	4	9	1	6	5	8	7

)=

MEMBERSHIP QUERIES

ask(

5	2	8	7	3	4	9	6	1
4	6	9	1	2	5	3	7	8
3	7	1	8	6	9	4	2	5
1	4	2	6	9	8	7	5	3
7	9	6	3	5	2	8	1	4
8	5	3	4	7	1	2	9	6
6	8	5	2	4	7	1	3	9
9	1	7	5	8	3	6	4	2
2	3	4	9	1	6	5	8	7

)=

PARTIAL QUERIES

ask(

5								
1								
3								
1								
7								
8								
6								
9								
2								

)=

PARTIAL QUERIES

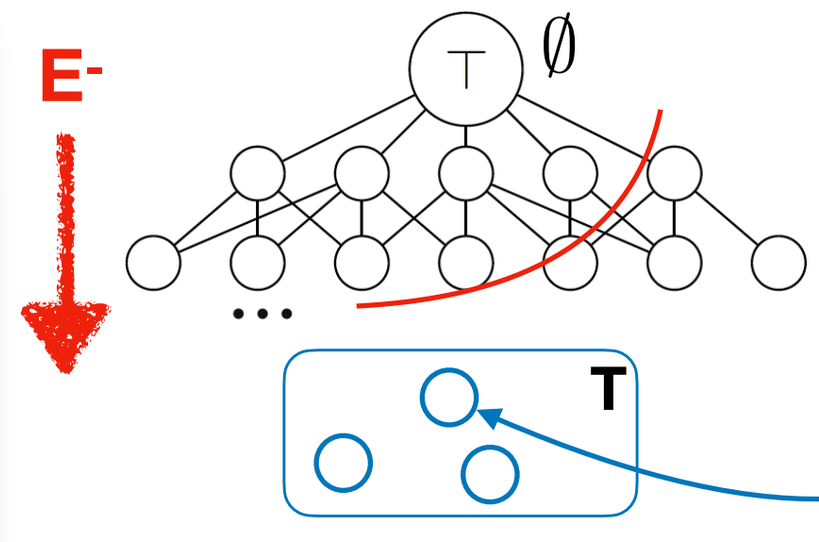
ask(

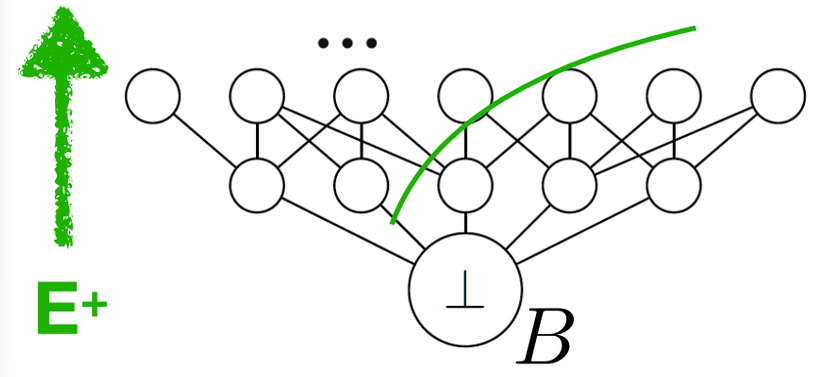
1								
7								
8								
6								
9								
2								

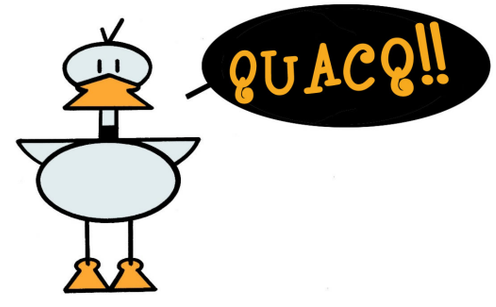
)=

ACTIVE LEARNING UNDER QUACQ

```
QUACQ(B):

while not converged do

   Generate an ‘informative’ query on (B,L)

   If no-query then ‘converge!’

   If query is negative
      //Top-down-inference
      FindScope
      FindC

   Else
      //Bottom-up–inference
      reduce_B
```







FINDSCOPE

findScope

$x_{1,2}$

$x_{1,4}$

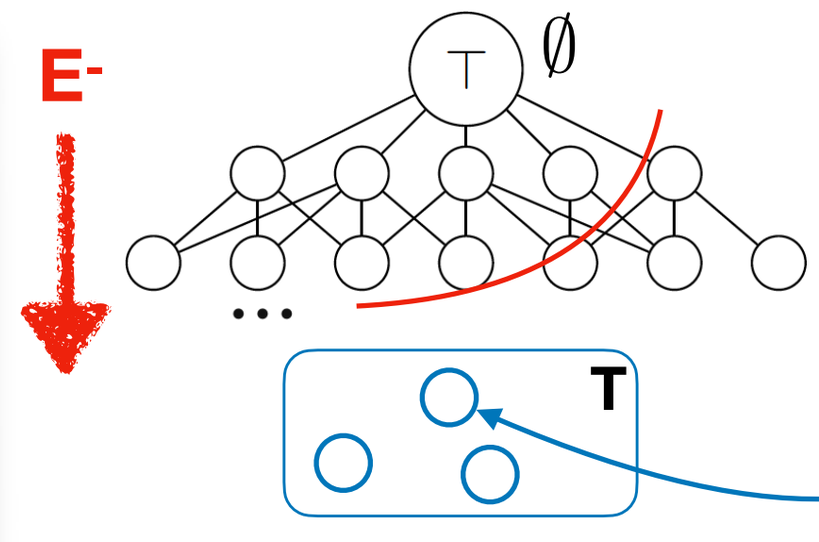
ask(

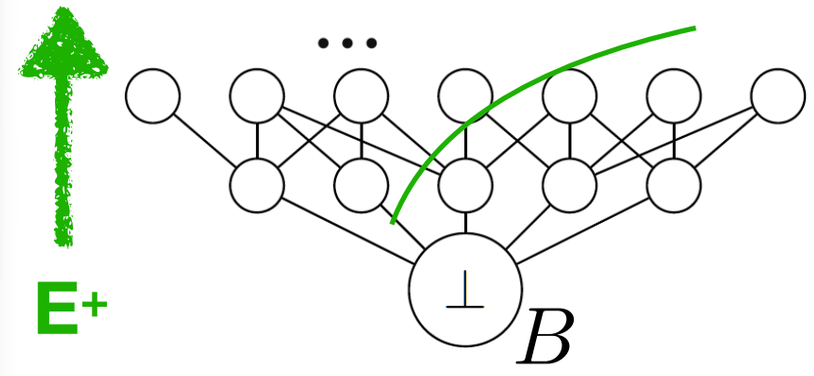
5	2	8	7	3	4	9	6	1
1	6	9	1	2	5	3	7	8
3	7	4	8	6	9	4	2	5
1	4	2	6	9	8	7	5	3
7	9	6	3	5	2	8	1	4
8	5	3	4	7	1	2	9	6
6	8	5	2	4	7	1	3	9
9	1	7	5	8	3	6	4	2
2	3	4	9	1	6	5	8	7

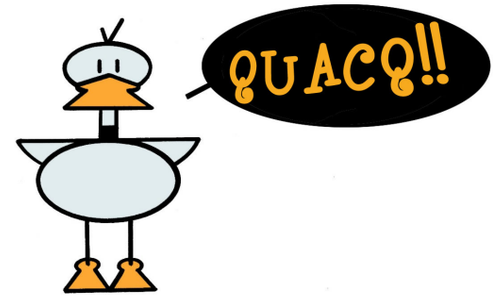
)=

ACTIVE LEARNING UNDER QUACQ

```
QUACQ(B):

while not converged do

   Generate an ‘informative’ query on (B,L)

   If no-query then ‘converge!’

   If query is negative
      //Top-down-inference
      FindScope
      FindC

   Else
      //Bottom-up–inference
      reduce_B
```







FINDC

findScope

$x_{1,2}$ ←

$x_{1,4}$ ←

ask(

5	2	8	7	3	4	9	6	1
1	6	9	1	2	5	3	7	8
3	7	4	8	6	9	4	2	5
1	4	2	6	9	8	7	5	3
7	9	6	3	5	2	8	1	4
8	5	3	4	7	1	2	9	6
6	8	5	2	4	7	1	3	9
9	1	7	5	8	3	6	4	2
2	3	4	9	1	6	5	8	7

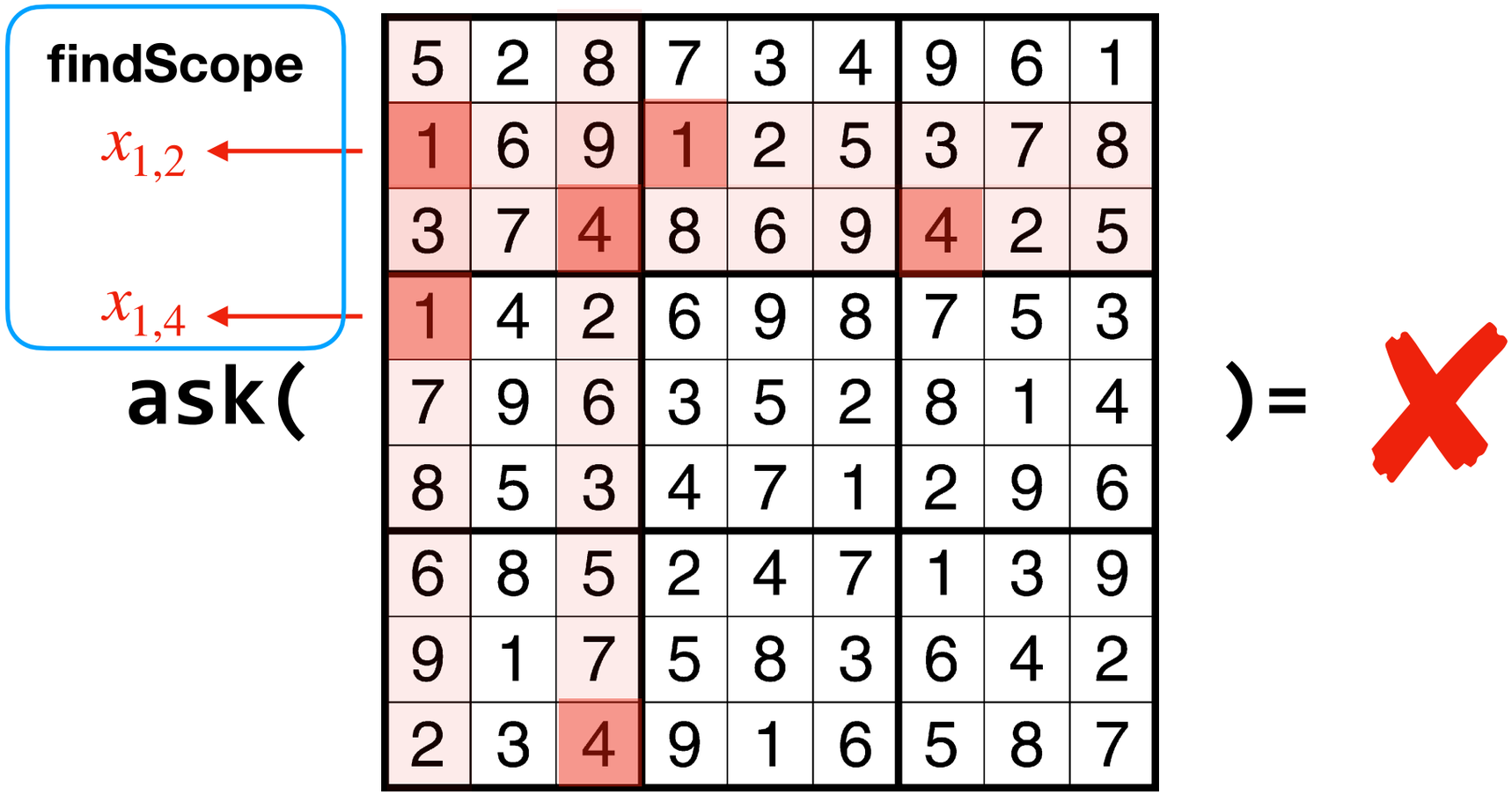
)= ✗

findC

$x_{1,2} \neq x_{1,4}$

findScope

FINDSCOPE


findScope
$x_{1,2}$
$x_{1,4}$
ask(
)=
5 2 8 7 3 4 9 6 1
1 6 9 1 2 5 3 7 8
3 7 4 8 6 9 4 2 5
1 4 2 6 9 8 7 5 3
7 9 6 3 5 2 8 1 4
8 5 3 4 7 1 2 9 6
6 8 5 2 4 7 1 3 9
9 1 7 5 8 3 6 4 2
2 3 4 9 1 6 5 8 7

QUACQ: FINDSCOPE (SKETCH)

```
findScope(e,Y,R):

    if ask(R)=No return emptyset

    if Y singleton return Y

    split(Y)

    S1 gets findScope(e, Y1, Y2 union R)

    S2 gets findScope(e, Y2, S1 union R)

    return S1 union S2
```

QUACQ: FINDSCOPE (EXAMPLE)

Negative example e on 12345 variables
Expected scopes ={25, 234}

```
findScope(e,Y,R):
    if ask(R)=No return emptyset
    if Y singleton return Y
    split(Y)
    S1 gets findScope(e, Y1, Y2 union R)
    S2 gets findScope(e, Y2, S1 union R)
    return S1 union S2
```

trace:

findScope(e, 12345, ∅)

nb_queries=00

QUACQ: FINDSCOPE (EXAMPLE)

Negative example e on 12345 variables
Expected scopes ={25, 234}

```
findScope(e,Y,R):
    if ask(R)=No return emptyset
    if Y singleton return Y
    split(Y)
    S1 gets findScope(e, Y1, Y2 union R)
    S2 gets findScope(e, Y2, S1 union R)
    return S1 union S2
```

trace:

findScope(e, 12345, ∅)
└── findScope(e, 12, 345) ✓

nb_queries=01

QUACQ: FINDSCOPE (EXAMPLE)

Negative example e on 12345 variables
Expected scopes ={25, 234}

```
findScope(e,Y,R):
    if ask(R)=No return emptyset
    if Y singleton return Y
    split(Y)
    S1 gets findScope(e, Y1, Y2 union R)
    S2 gets findScope(e, Y2, S1 union R)
    return S1 union S2
```

trace:

findScope(e, 12345, ∅)
└── findScope(e, 12, 345)= 2
└── findScope(e, 1, 2345)= ∅ ✘

nb_queries=02

QUACQ: FINDSCOPE (EXAMPLE)

Negative example e on 12345 variables
Expected scopes ={25, 234}

```
findScope(e,Y,R):
    if ask(R)=No return emptyset
    if Y singleton return Y
    split(Y)
    S1 gets findScope(e, Y1, Y2 union R)
    S2 gets findScope(e, Y2, S1 union R)
    return S1 union S2
```

trace:

```
findScope(e, 12345, ∅)
    └── findScope(e, 12, 345)= 2
            └── findScope(e, 1, 2345)= ∅
            └── findScope(e, 2, 345)= 2
```

//already asked query
nb_queries=02

QUACQ: FINDSCOPE (EXAMPLE)

Negative example e on 12345 variables
Expected scopes ={25, 234}

```
findScope(e,Y,R):

    if ask(R)=No return emptyset

    if Y singleton return Y

    split(Y)

    S1 gets findScope(e, Y1, Y2 union R)

    S2 gets findScope(e, Y2, S1 union R)

    return S1 union S2
```

trace:

```
findScope(e, 12345, ∅)
    └──findScope(e, 12, 345)= 2
```

nb_queries=02

QUACQ: FINDSCOPE (EXAMPLE)

Negative example e on 12345 variables
Expected scopes ={25, 234}

```
findScope(e,Y,R):

    if ask(R)=No return emptyset

    if Y singleton return Y

    split(Y)

    S1 gets findScope(e, Y1, Y2 union R)

    S2 gets findScope(e, Y2, S1 union R)

    return S1 union S2
```

trace:

```
findScope(e, 12345, ∅)
    └──findScope(e, 12, 345)= 2
    └──findScope(e, 345, 2)
```

nb_queries=03

QUACQ: FINDSCOPE (EXAMPLE)

Negative example e on 12345 variables
Expected scopes ={25, 234}

```
findScope(e,Y,R):
    if ask(R)=No return emptyset
    if Y singleton return Y
    split(Y)
    S1 gets findScope(e, Y1, Y2 union R)
    S2 gets findScope(e, Y2, S1 union R)
    return S1 union S2
```

trace:

```
findScope(e, 12345, ∅)
    └── findScope(e, 12, 345)= 2
    └── findScope(e, 345, 2)
            └── findScope(e, 3, 452)= ∅   ✘
```

nb_queries=04

QUACQ: FINDSCOPE (EXAMPLE)

Negative example e on 12345 variables
Expected scopes ={25, 234}

```
findScope(e,Y,R):
    if ask(R)=No return emptyset
    if Y singleton return Y
    split(Y)
    S1 gets findScope(e, Y1, Y2 union R)
    S2 gets findScope(e, Y2, S1 union R)
    return S1 union S2
```

trace:

```
findScope(e, 12345, ∅)
    └── findScope(e, 12, 345)= 2
    └── findScope(e, 345, 2)= 34
            └── findScope(e, 3, 452)= ∅
            └── findScope(e, 45, 2)
```

//already asked query

nb_queries=04

QUACQ: FINDSCOPE (EXAMPLE)

Negative example e on 12345 variables
Expected scopes ={25, 234}

```
findScope(e,Y,R):
    if ask(R)=No return emptyset
    if Y singleton return Y
    split(Y)
    S1 gets findScope(e, Y1, Y2 union R)
    S2 gets findScope(e, Y2, S1 union R)
    return S1 union S2
```

trace:

```
findScope(e, 12345, ∅)
    └── findScope(e, 12, 345)= 2
    └── findScope(e, 345, 2)= 34
            └── findScope(e, 3, 452)= ∅
            └── findScope(e, 45, 2)
                    └── findScope(e, 4, 25)= ∅   ✘
```

nb_queries=05

QUACQ: FINDSCOPE (EXAMPLE)

Negative example e on 12345 variables
Expected scopes ={25, 234}

```
findScope(e,Y,R):
    if ask(R)=No return emptyset
    if Y singleton return Y
    split(Y)
    S1 gets findScope(e, Y1, Y2 union R)
    S2 gets findScope(e, Y2, S1 union R)
    return S1 union S2
```

trace:

```
findScope(e, 12345, ∅)
    └── findScope(e, 12, 345)= 2
    └── findScope(e, 345, 2)= 34
            └── findScope(e, 3, 452)= ∅
            └── findScope(e, 45, 2)= 5
                    └── findScope(e, 4, 25)= ∅
                    └── findScope(e, 5, 2)= 5   ✔   //already asked query
```

nb_queries=05

QUACQ: FINDSCOPE (EXAMPLE)

Negative example e on 12345 variables
Expected scopes ={25, 234}

```
findScope(e,Y,R):
    if ask(R)=No return emptyset
    if Y singleton return Y
    split(Y)
    S1 gets findScope(e, Y1, Y2 union R)
    S2 gets findScope(e, Y2, S1 union R)
    return S1 union S2
```

trace:

```
findScope(e, 12345, ∅)
    └── findScope(e, 12, 345)= 2
    └── findScope(e, 345, 2)= 34
            └── findScope(e, 3, 452)= ∅
            └── findScope(e, 45, 2)= 5
```

nb_queries=05

QUACQ: FINDSCOPE (EXAMPLE)

Negative example e on 12345 variables
Expected scopes ={25, 234}

```
findScope(e,Y,R):
    if ask(R)=No return emptyset
    if Y singleton return Y
    split(Y)
    S1 gets findScope(e, Y1, Y2 union R)
    S2 gets findScope(e, Y2, S1 union R)
    return S1 union S2
```

trace:

```
findScope(e, 12345, ∅)
    └── findScope(e, 12, 345)= 2
    └── findScope(e, 345, 2)= 5
```

nb_queries=05

QUACQ: FINDSCOPE (EXAMPLE)

Negative example e on 12345 variables
Expected scopes ={25, 234}

```
findScope(e,Y,R):
    if ask(R)=No return emptyset
    if Y singleton return Y
    split(Y)
    S1 gets findScope(e, Y1, Y2 union R)
    S2 gets findScope(e, Y2, S1 union R)
    return S1 union S2
```

trace:

findScope(e, 12345, ∅) = 25

nb_queries=05

findC

QUACQ: FINDC

```
findC(Y):

   add to Delta all constraints in B on Y

   while true

      Generate a query e that satisfies some and violates some of Delta

      if no-query then return Delta

      if ask(e)= Yes
      Remove from Delta all constraints rejecting e

      else
      Keep in Delta only constraints rejecting e
```

QUACQ: FINDC (EXAMPLE)

```
findC(x1x2):  //(concept on x1x2 says that x1 diff x2)

    1. Delta = {=, =<, >=, <, >, !=}
    2. e1 = (1,1)           //negative example
    3. Delta= {<,>,!=}      //keep in Delta constraints rejecting e
    4. e2=(1,2)             //positive example
    5. Delta ={<,!=}        //remove from Delta constraints rejecting e
    6. e3=(2,1)             //positive example
    7. Delta ={!=}          //remove from Delta constraints rejecting e
    8. Return (x1 != x2)
```

COMPLEXITY OF QUACQ

The number of queries required to find the target concept is in:

$$O(|T| . (log|X| + |\Gamma|)$$

E-

The number of queries required to converge is in:

$$O(|B|)$$

E+

Discussion

QUACQ LIMITATIONS

1. Large number of queries (e.g., more than 9,000 to learn sudoku)
2. Query generation can be time-consuming (e.g., 20min for sudoku)
3. Still limited practical applications

QUACQ LIMITATIONS

1. **Large number of queries (e.g., more than 9,000 to learn sudoku)**
2. Query generation can be time-consuming (e.g., 20min for sudoku)
3. Still limited practical applications

- **Solutions to (1):**
  - Complexe queries (generalisation [Bessiere et al., ECAI'14, ICTAI'15], recommandation [Daoudi et al., IJCAI'16], …)
  - More elicitation (MultiAcq [Arcangioli et al., IJCAI'16], MQuacq [Tsouros, CP'18],…)
  - Parallel Constraint Acquisition [Lazaar, AAAI'21]
  - Non-Human Users (programs, expert systems, robots…)

QUACQ LIMITATIONS

1. Large number of queries (e.g., more than 9,000 to learn sudoku)
2. **Query generation can be time-consuming (e.g., 20min for sudoku)**
3. Still limited practical applications

- **Solutions to (2):**
  - Adaptive and time-bounded query generator (T-QUACQ [Ait Addi et al, CPAIOR'18])
  - Smart query generation [Bessiere et al, CoRR'20]

QUACQ LIMITATIONS

1. **Large number of queries (e.g., more than 9,000 to learn sudoku)**
2. **Query generation can be time-consuming (e.g., 20min for sudoku)**
3. Still limited practical applications

- **Solutions to (1)&(2):**
  - Deep Constraint Acquisition (future work)

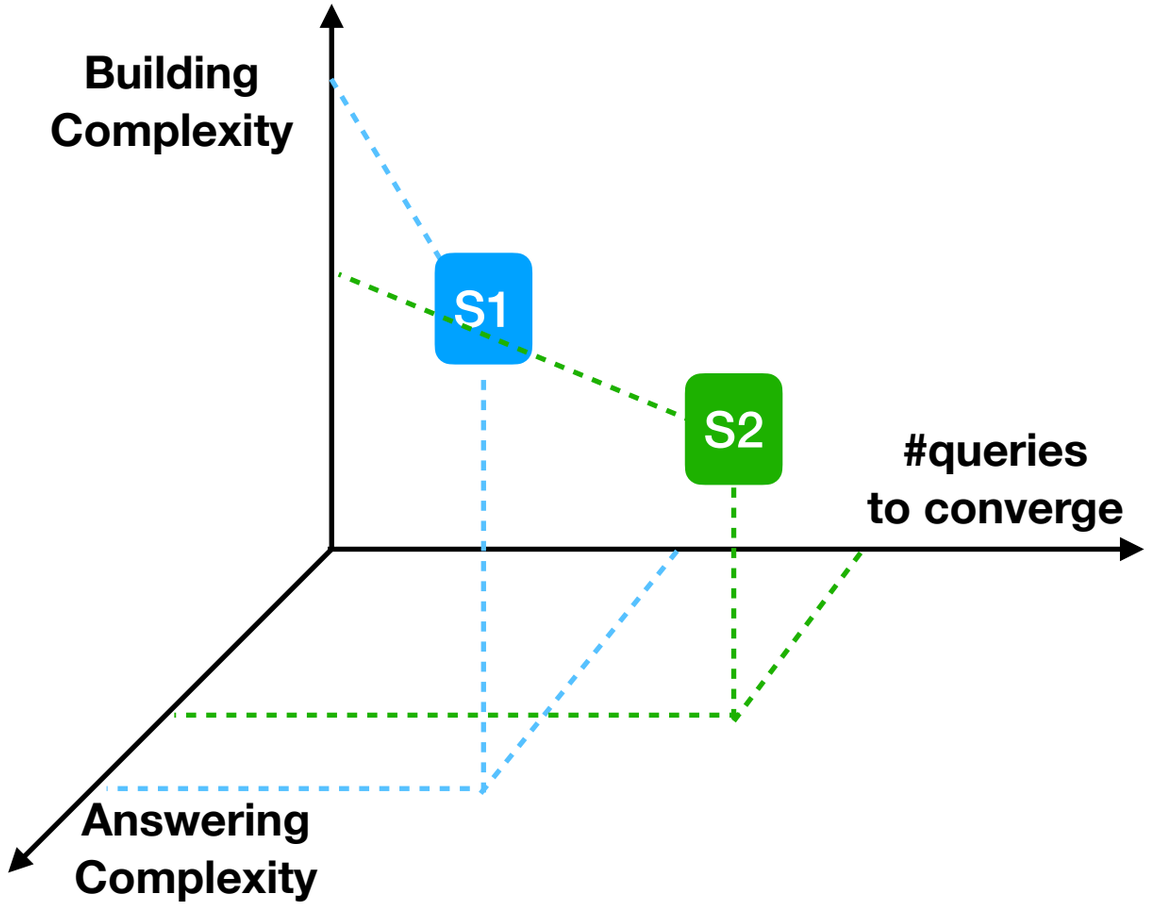
QUACQ LIMITATIONS

1. Large number of queries (e.g., more than 9,000 to learn sudoku)
2. Query generation can be time-consuming (e.g., 20min for sudoku)
3. **Still limited practical applications**

- **Ongoing works (3):**
  - Dedicated Acquisition for Scheduling problems
  - Timetabling Acquisition
  - Software verification&validation
  - Constraint Acquisition in Industrial Robots
  - …

CONSTRAINT ACQUISITION TOOLBOX

- Acquisition toolbox with a Query types taxonomy
- Robust toolbox under false positives/false negatives



SOME THOUGHTS

- Constraint Acquisition has attracted considerable attention in recent years
- Constraint Acquisition is the key to enable the spread of CP technology in industrial field (e.g., continuous development process)
- QUACQ-like solutions are explainable
- Constraint Acquisition is a good problem to show the power of a hybrid learning
- COCONUT Team is a good place to learn more about acquisition ;-)

Exercises

EX01: CONACQ

Given a vocabulary (X, D)= ([x_1, x_2, x_3], [1,3])

Given a language Γ= {=, ≠}

1. Compute the basis B
2. How many version space do we have?
3. How many constraint networks do we have?

Given a concept to learn: different values for x_i

- Passive Learning:

Given a training set: e_1=(1,1,2), e_2=(1,2,2), e_3=(2,2,3), e_4=(1,3,1)

4. Use Conacq.1 to learn the given concept
5. What happen if we add e_5=(1,2,3)?

- Passive Learning:

6. Use Conacq.2 to learn the given concept

EX02: QUACQ

Given a vocabulary (X, D)= ([x_1, x_2, x_3], [1,3])

Given a language Γ= {=, ≠}

Given a concept to learn: different values for x_i

1. Use QUACQ to learn the given concept

EX03: FINDSCOPE

- Given X= [1,2,3,4,5,6,7,8,9]
- Given a negative example e violating constraints on: [2,9], [3,4,8], [2,5,7]
    1. Do a shuffle on X
    2. Use FindScope function to return a violated constraint scope
    3. How many queries are asked to return a scope?

```
findScope(e,Y,R):

    if ask(R)=No return emptyset

    if Y singleton return Y

    split(Y)

    S1 gets findScope(e, Y1, Y2 union R)

    S2 gets findScope(e, Y2, S1 union R)

    return S1 union S2
```

EX04: FINDC

- Γ= {=, ≠, <, >, ≤,≥}
- Scope: (x_3,x_5)
- Target: $x_3 \neq x_5$
    1. Use FindC to learn the target constraint

```
findC(Y):

   add to Delta all constraints in B on Y

   while true

      Generate a query e that satisfies some and violates some of Delta

      if no-query then return Delta

      if ask(e)= Yes
      Remove from Delta all constraints rejecting e

      else
      Keep in Delta only constraints rejecting e
```

Demo

https://github.com/lirmm/ConstraintAcquisition


Thank
You!