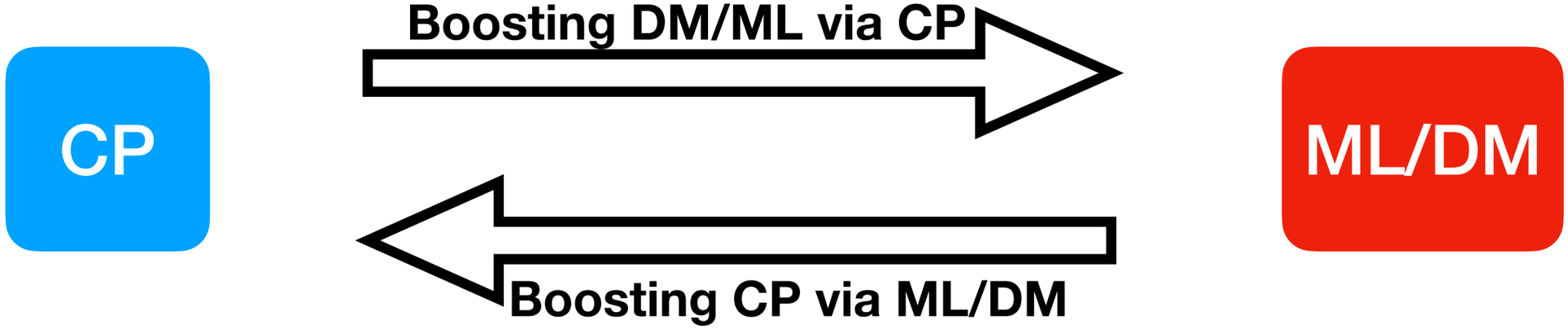
Constraint Acquisition

Nadjib Lazaar, COCONUT Team
LIRMM, University of Montpellier, CNRS

ACP'2020 GdR IA, RO, and ANITI Autumn School
26 nov. 2020

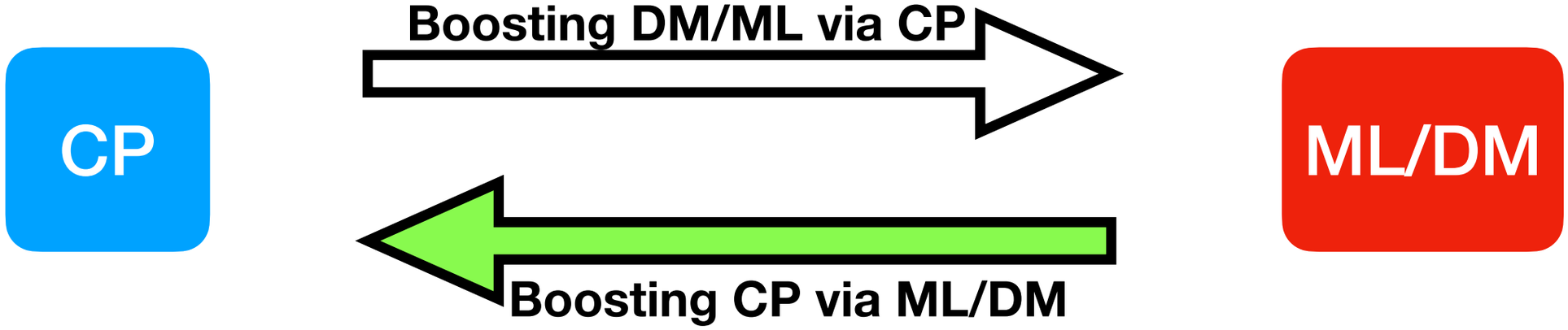
CONSTRAINT PROGRAMMING & MACHINE LEARNING/DATA MINING



- **Caviar Working Group - GdR IA**
  - www.lirmm.fr/~lazaar/caviar.html

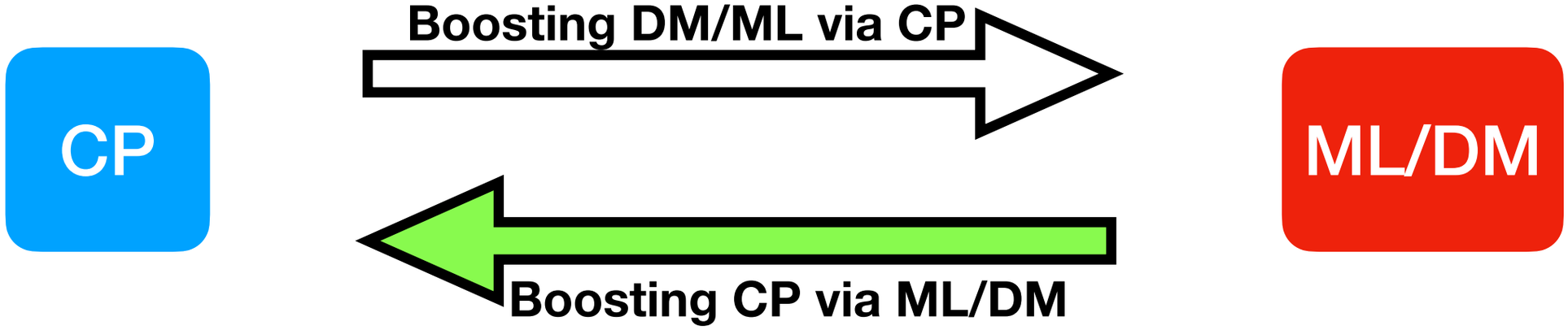
- **Declarative Data Mining - Samir Loudni**
- **Constraint Acquisition - Nadjib Lazaar**

CONSTRAINT PROGRAMMING & MACHINE LEARNING/DATA MINING



- **ML/DM for Solving:** Use of Data&Models to boost the resolution
  - Adaptive Solver (adaptive consistency, adaptive var/val heuristic)
  - Algorithm Selection
  - Clause learning in modern SAT solvers
  - Finding good patterns in the search tree
  - …

CONSTRAINT PROGRAMMING & MACHINE LEARNING/DATA MINING

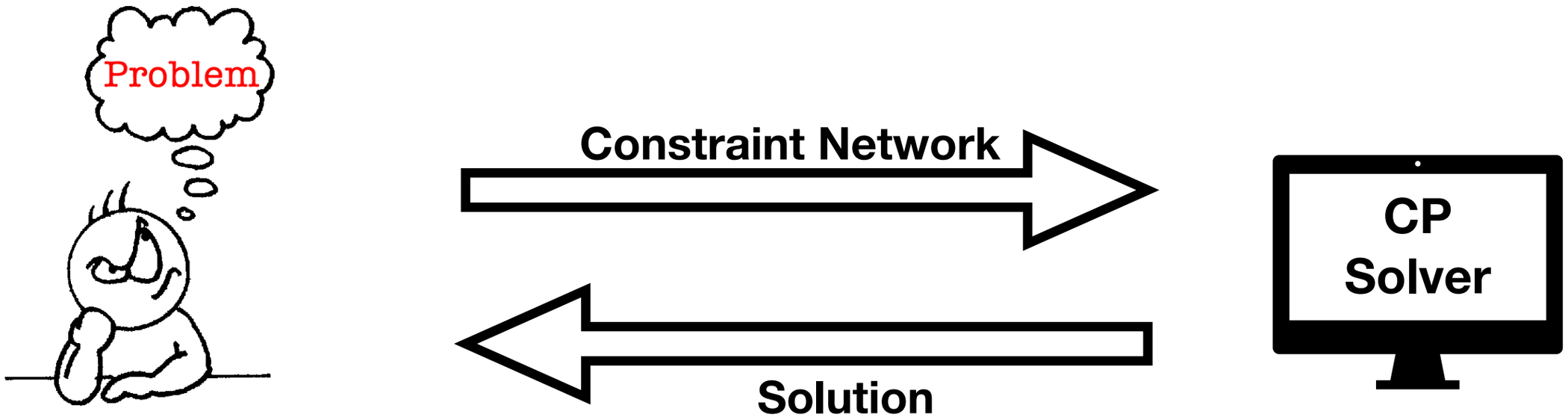


- **ML/DM for Solving:** Use of Data&Models to boost the resolution
  - Adaptive Solver (adaptive consistency, adaptive var/val heuristic)
  - Algorithm Selection
  - Clause learning in modern SAT solvers
  - Finding good patterns in the search tree
  - …
- **ML/DM for Modelling:** From Data to declarative models
  - Empirical model learning using Neural Networks, Decision Trees,
  - Learning boundaries for objective variables
  - **Constraint Acquisition**
  - …

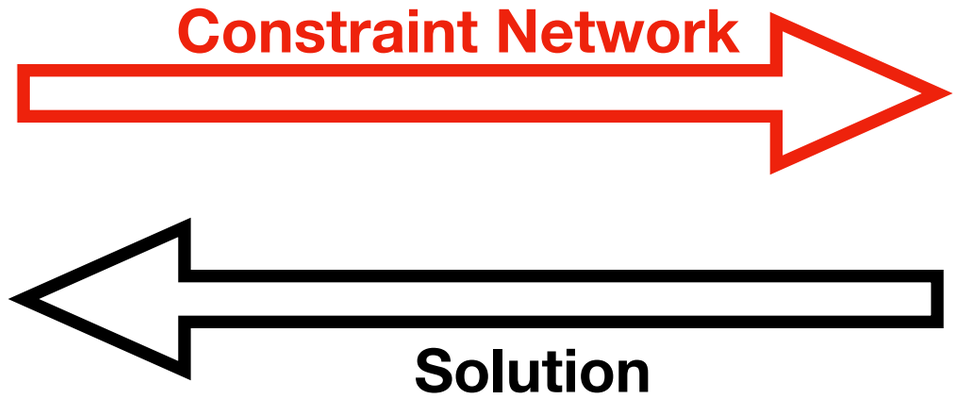
OUTLINE

- Version Space Learning (Overview)
- Constraint Acquisition Problem
- Constraint Acquisition Algorithms
- QUACQ: Quick Acquisition
- Discussion
- Exercises
- Demo

CONSTRAINT ACQUISITION – MOTIVATION



CONSTRAINT ACQUISITION – MOTIVATION



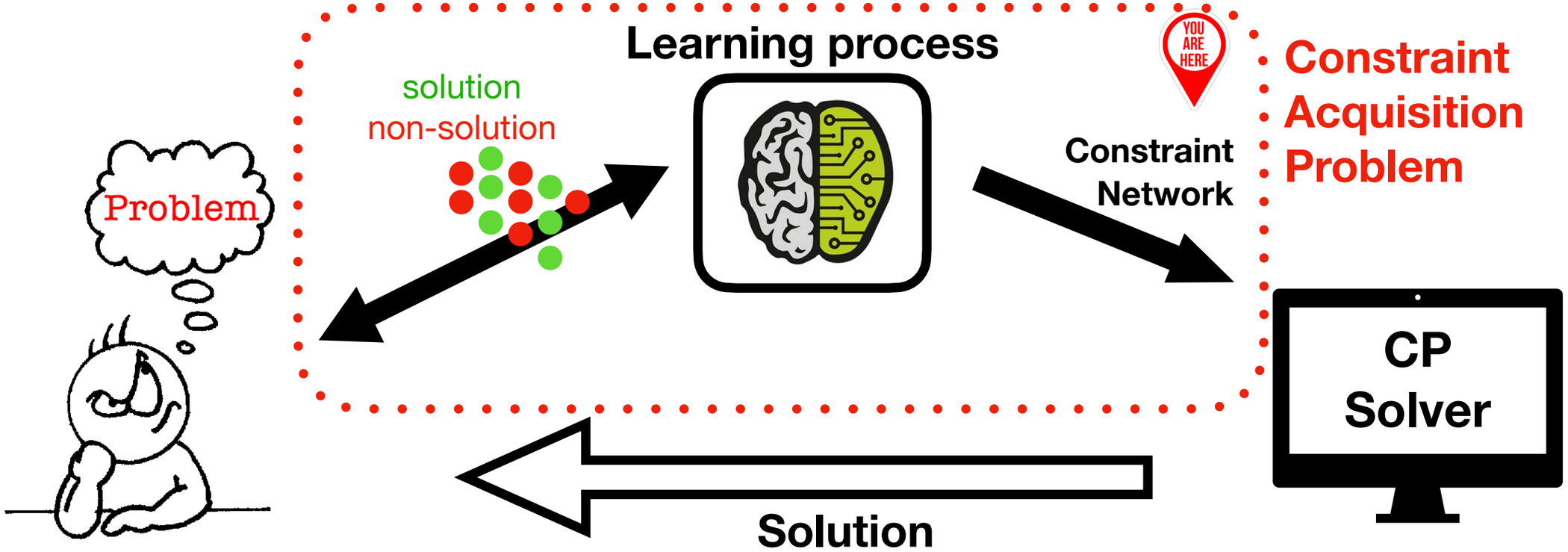
Question: How does the user write down the constraints of a problem?

Limitations:

- Modelling constraint networks require a fair expertise [Freuder99, Frisch et al.05, Smith06]
- Conceptual and technological gap
- Error-prone
- Hard to take advantage of raw data

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

CONSTRAINT ACQUISITION – MOTIVATION



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

Limitations:

- Modelling constraint networks require a fair expertise [Freuder99, Frisch et al.05, Smith06]
- Conceptual and technological gap
- Error-prone
- Hard to take advantage of raw data

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

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

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

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 $D = \{D_1, \ldots, D_n\}$

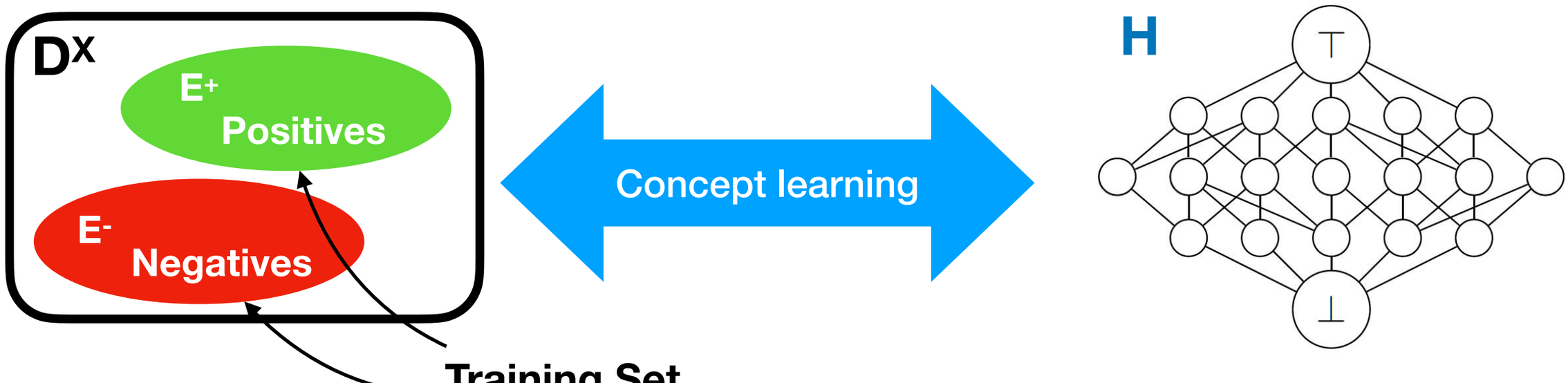
- A concept is a Boolean function $f : X \rightarrow \{0,1\}$
  - $f(a) = 0 \Rightarrow a \in D^X$ is a negative instance
  - $f(b) = 1 \Rightarrow b \in D^X$ is a positive instance
  - A function of a fixed set of predicates $P_i(X)$, which can be evaluated over instances (ex: $f(X) \equiv P_1(X) \wedge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

- 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.

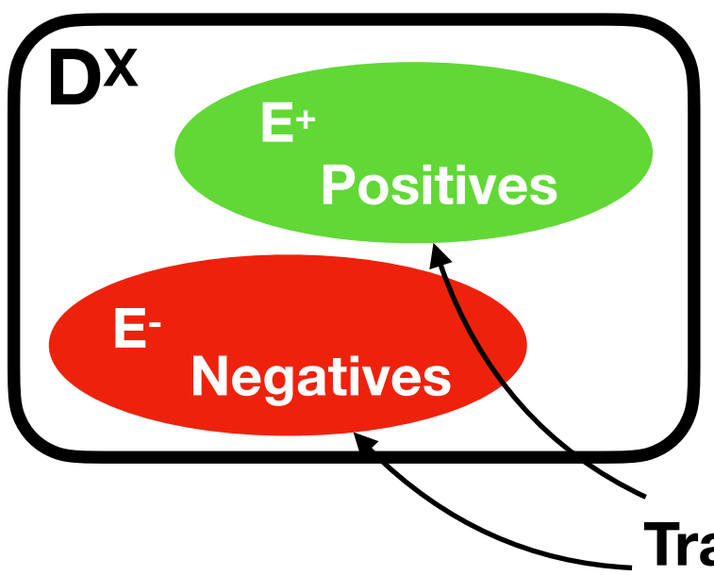
VERSION SPACE LEARNING (OVERVIEW)

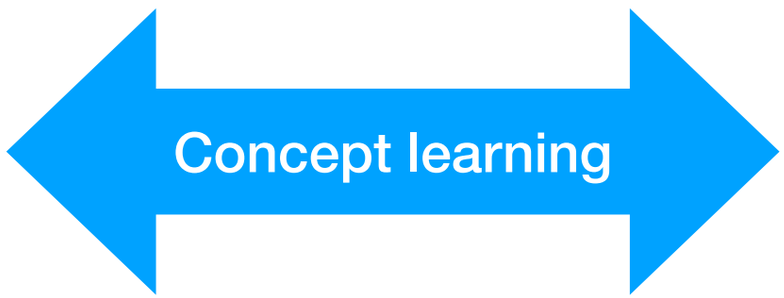
[MITCHELL82]

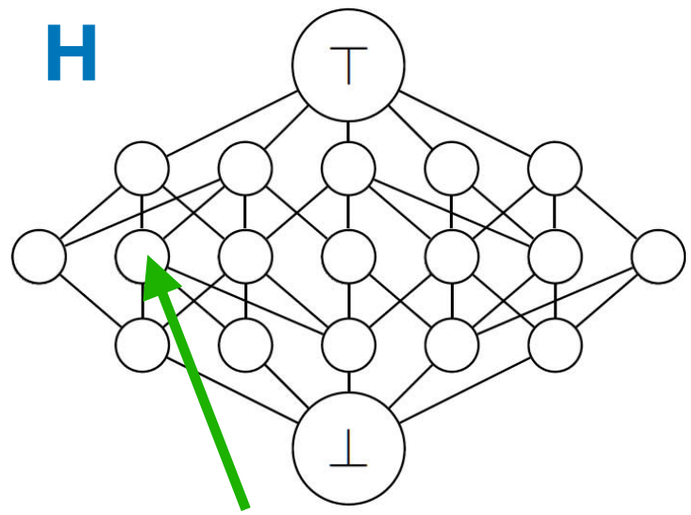


VERSION SPACE LEARNING (OVERVIEW)

[MITCHELL82]





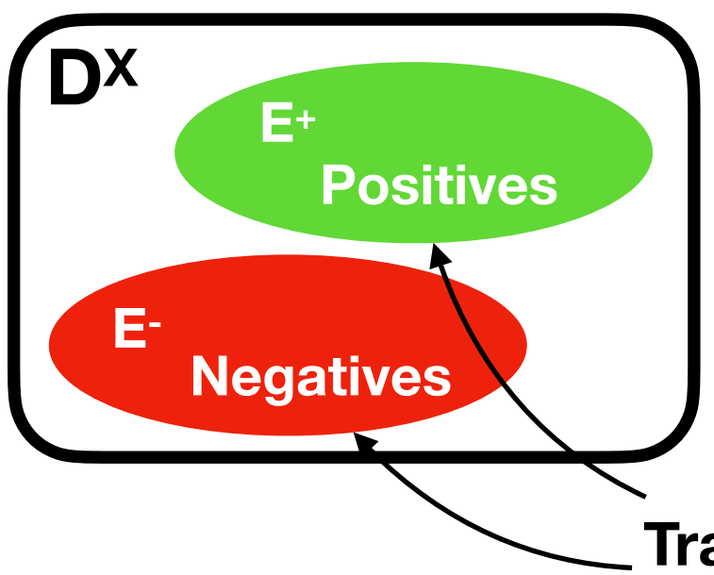


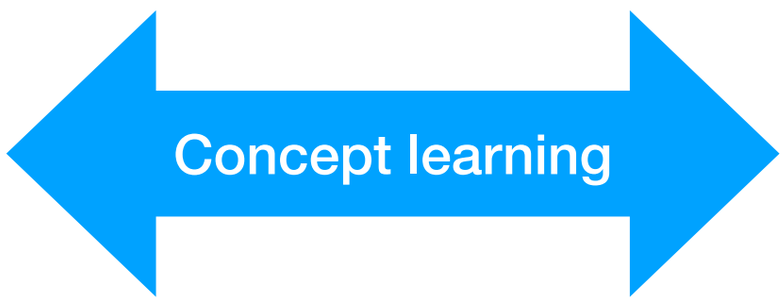
$f \equiv P_1 \wedge P_2 \wedge P_3$

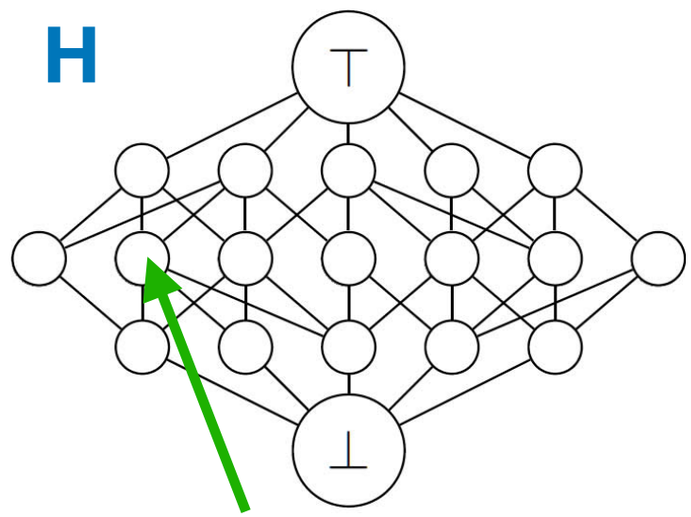
Version space

VERSION SPACE LEARNING (OVERVIEW)

[MITCHELL82]



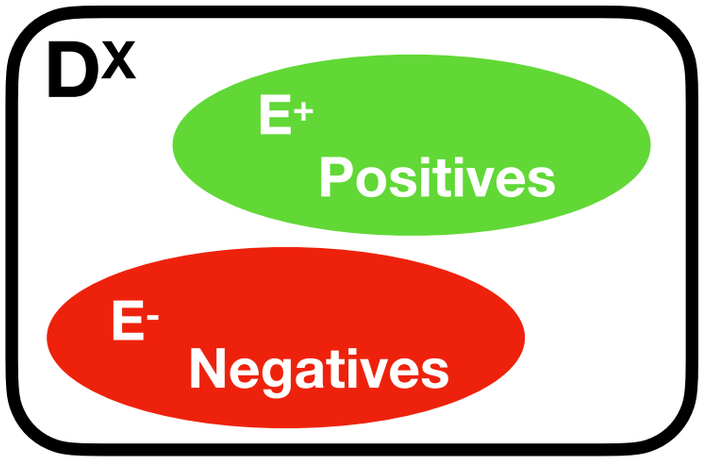




$f \equiv P_1 \wedge P_2 \wedge P_3$

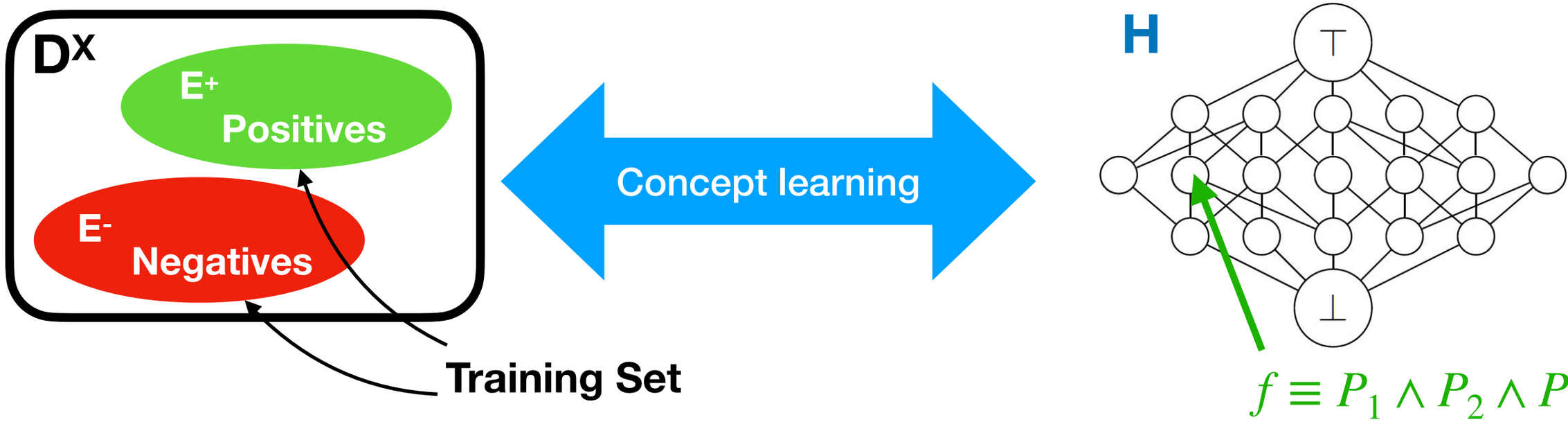
Version space

Constraint Programming:



VERSION SPACE LEARNING (OVERVIEW)

[MITCHELL82]

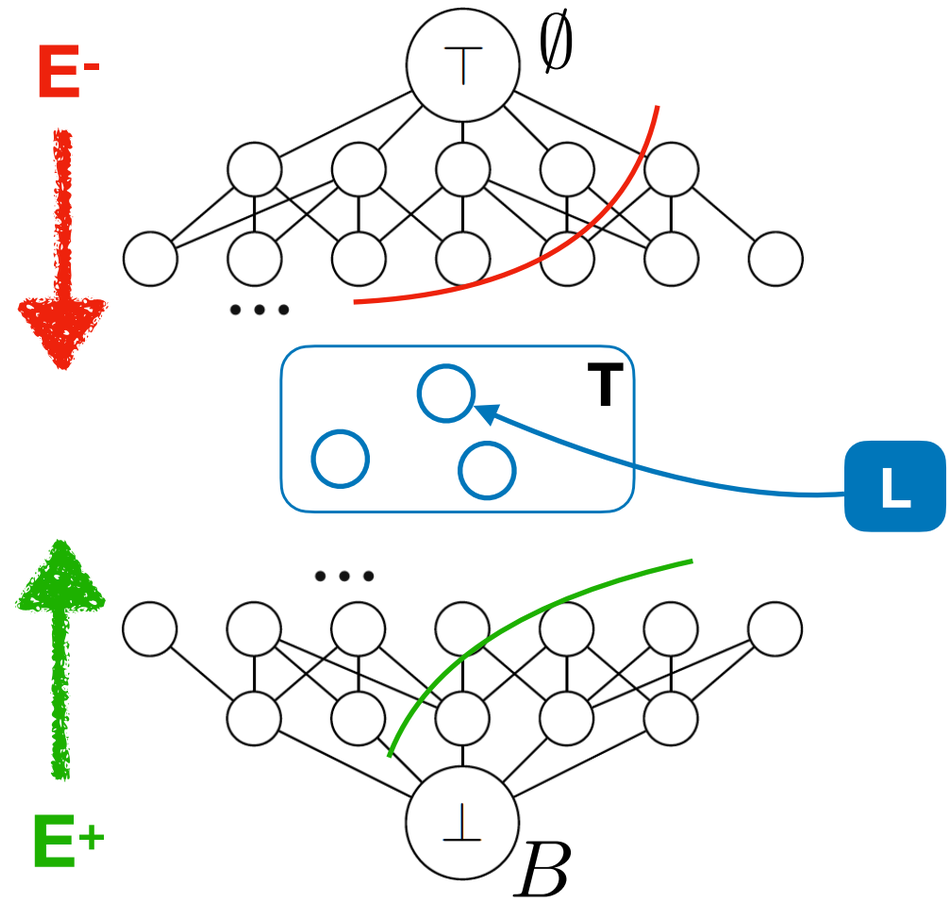


Constraint Programming:

CONSTRAINT ACQUISITION PROBLEM

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

- Output:
- L: Learned network such that:



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

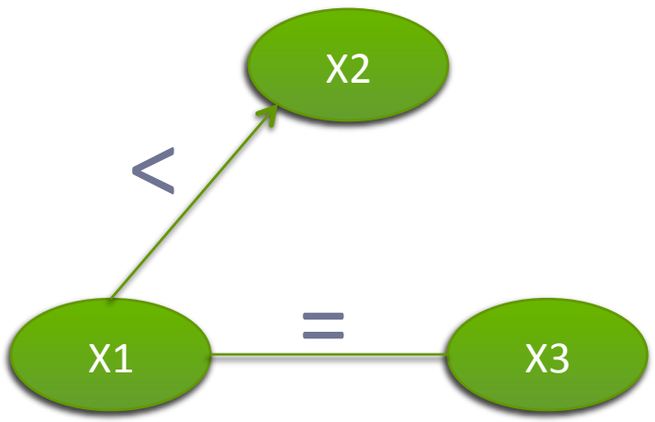
- **INPUT:**
  - $X = \{x_1, x_2, x_3\}$
  - $D(x_i) = \{1,2,3\}$
  - $\Gamma = \{ <, = \}$
  - $B = \{x_i < x_j, x_i = x_j, \forall i, j\}$
  - $T \equiv \{(1,2,1), (2,3,2)\}$

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

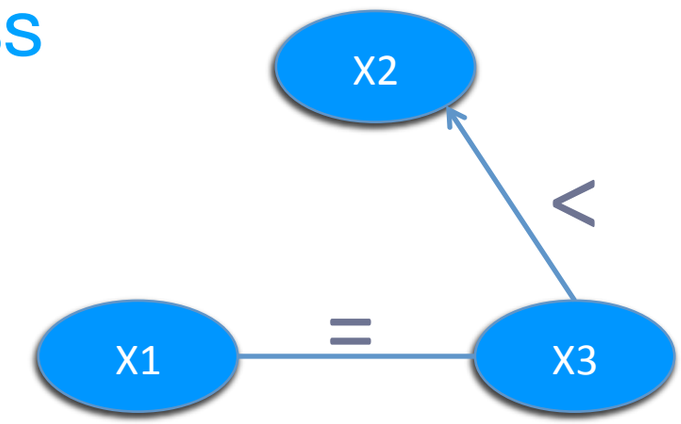
OR

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

Equivalence class



L_1



L_2

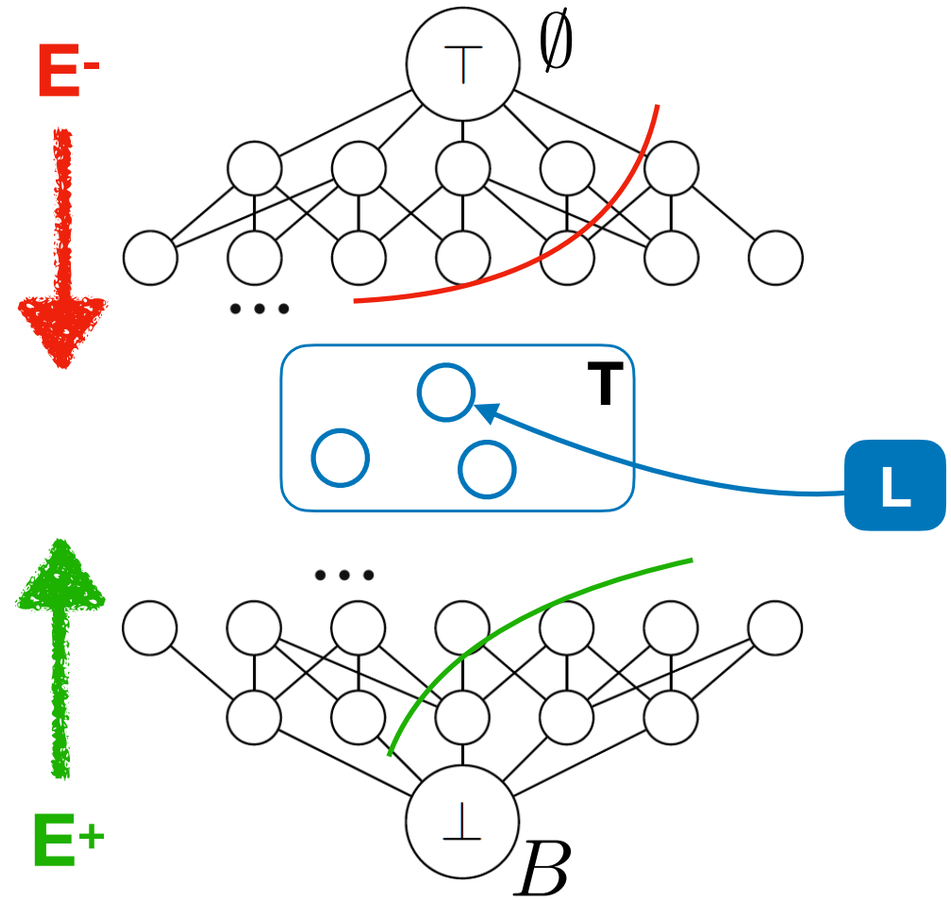
CONSTRAINT ACQUISITION PROBLEM

Convergence Problem:

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

agreeing with E, we have:

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



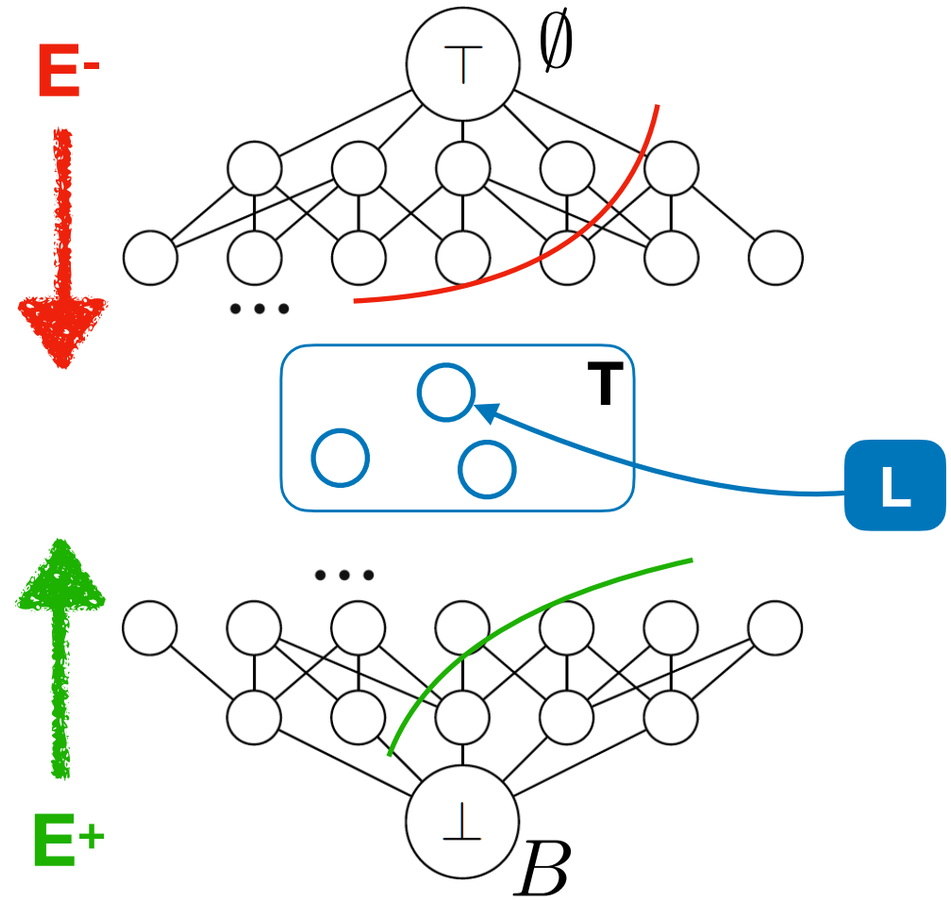
CONSTRAINT ACQUISITION PROBLEM

Convergence Problem:

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

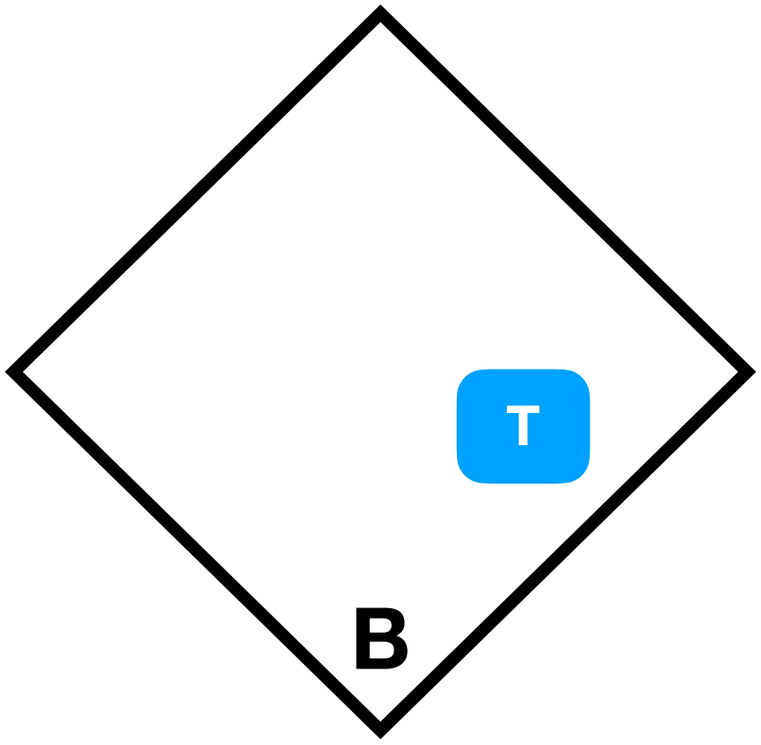
agreeing with E, we have:

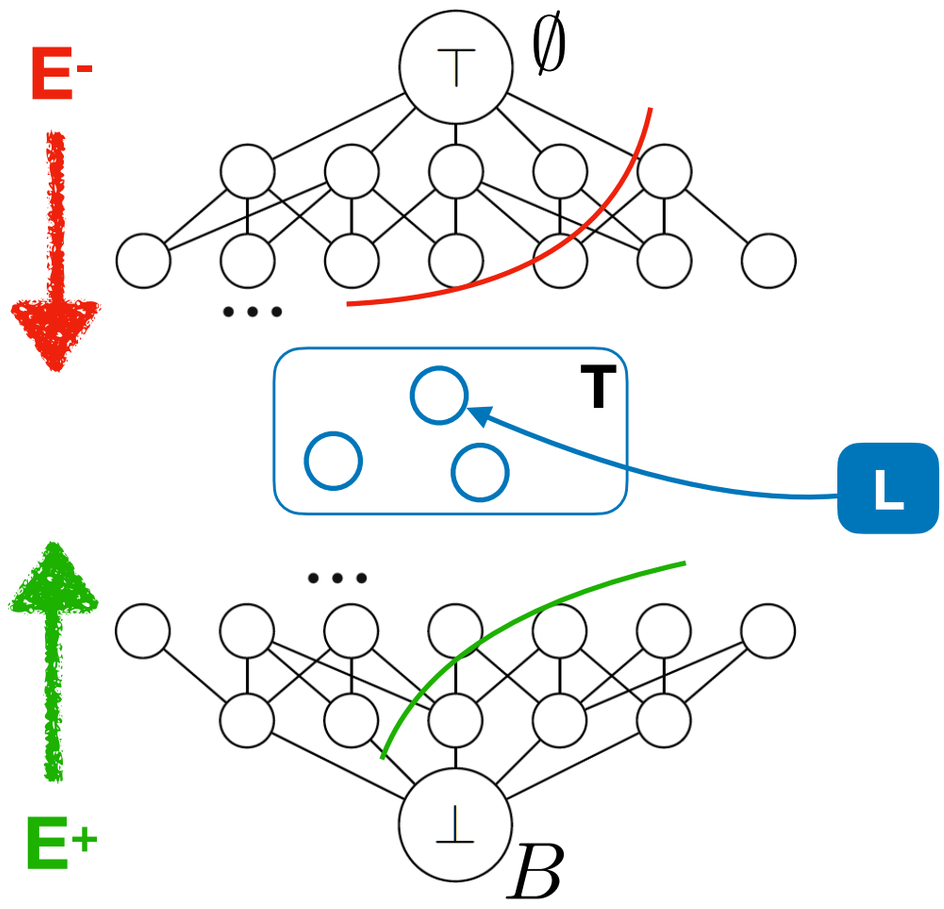
$$sol(C') = sol(L)$$



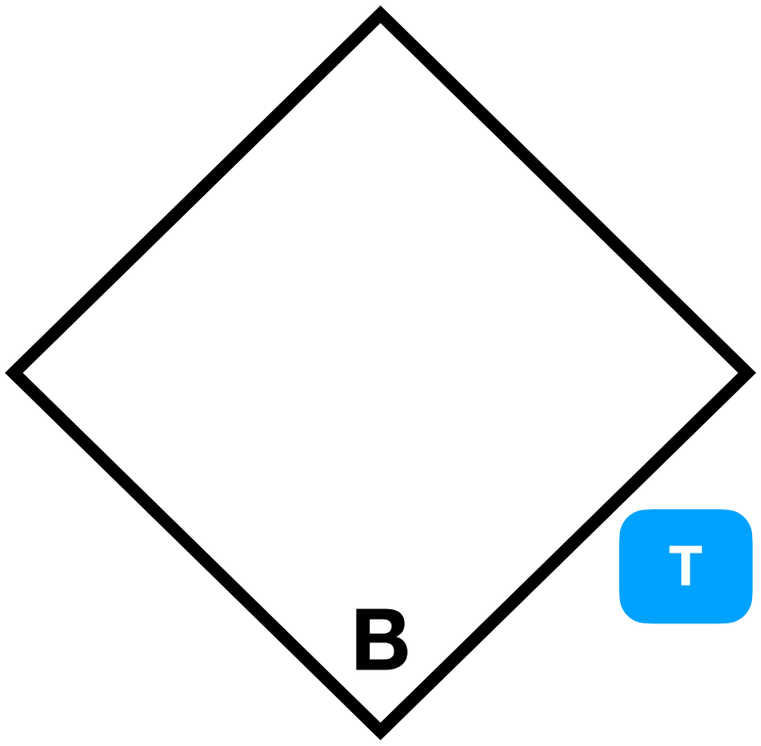
coNP-Complete [Bessiere, Koriche, Lazaar, O'Sullivan, AIJ17]

CONSTRAINT ACQUISITION PROBLEM

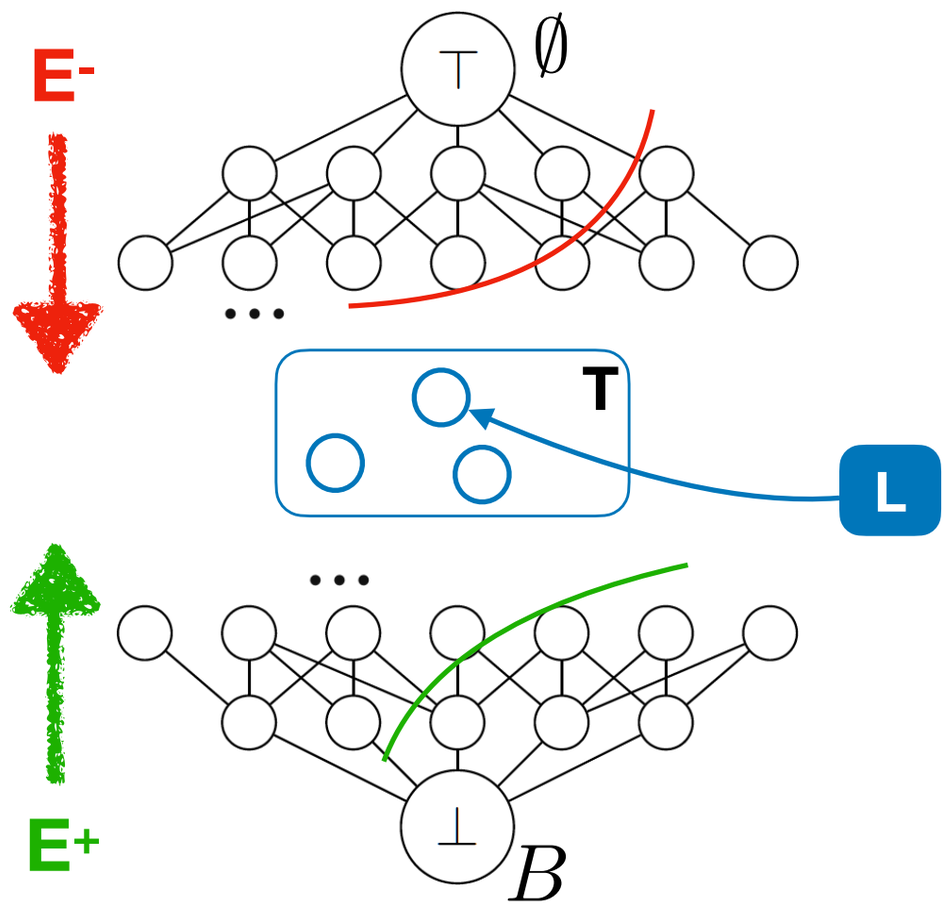




CONSTRAINT ACQUISITION PROBLEM



Collapse!

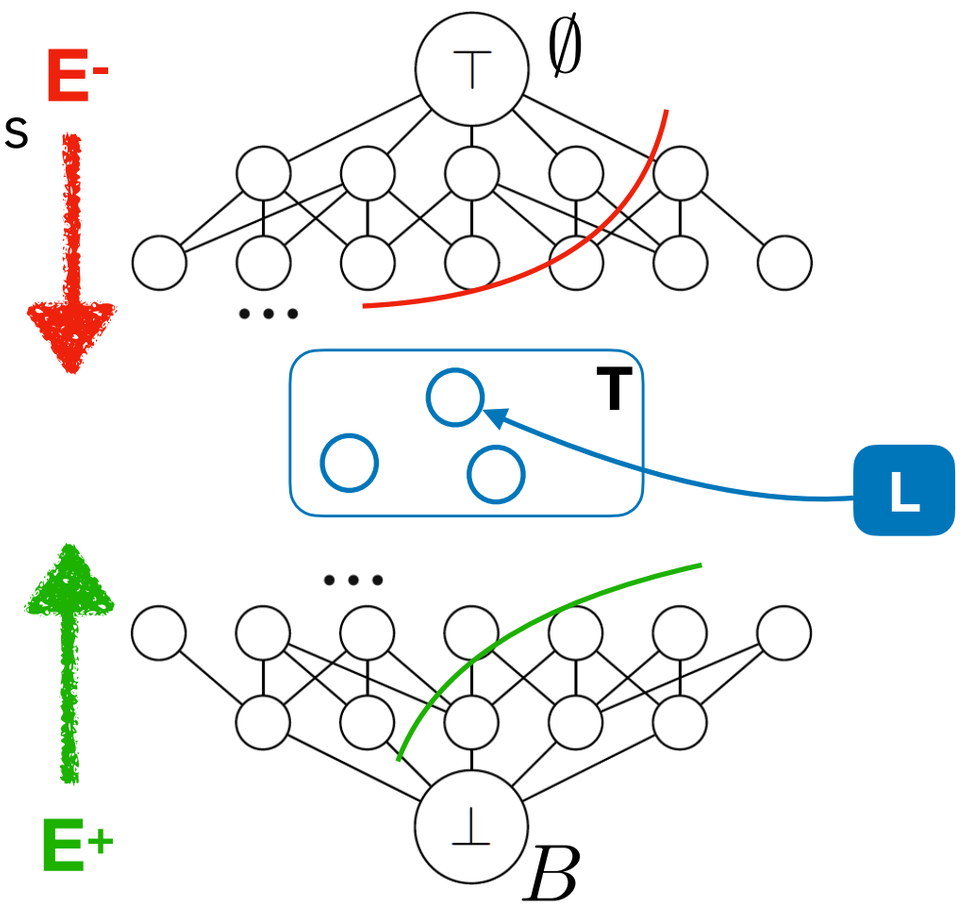


CONACQ

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
  - $l_i = 1 \Rightarrow C_i \in L$



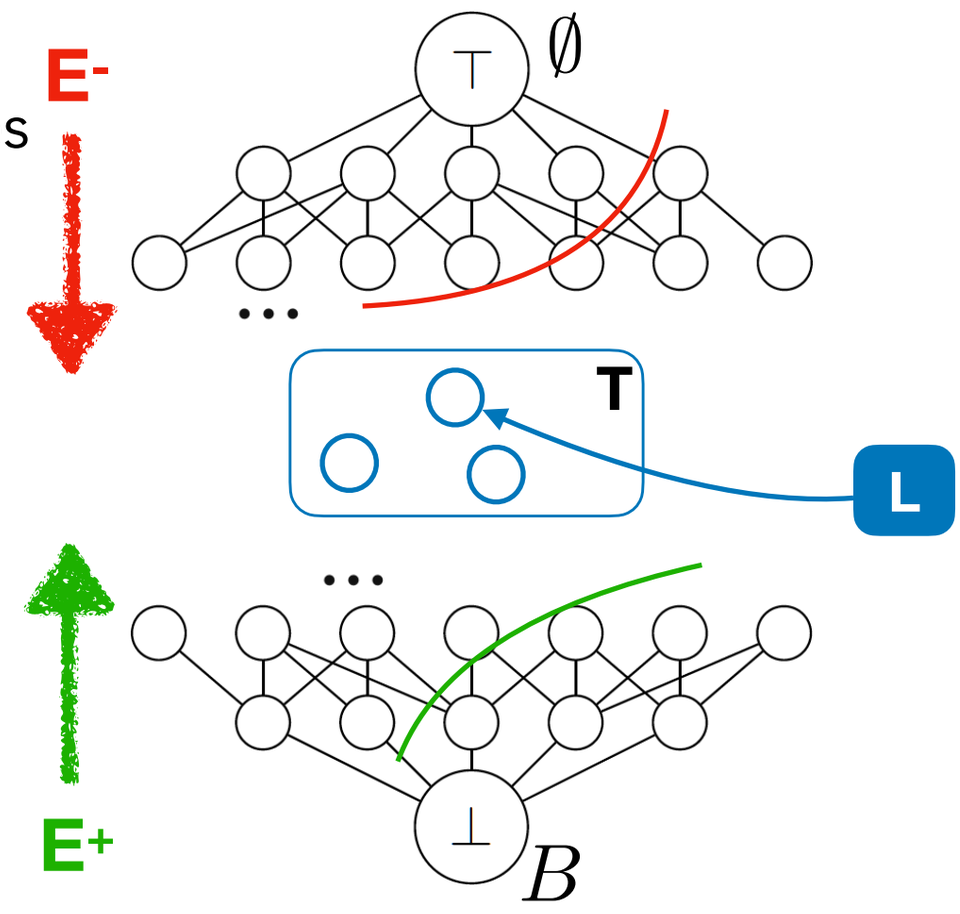
ACQUISITION USING MEMBERSHIP QUERIES

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
  - $l_i = 1 \Rightarrow C_i \in L$

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

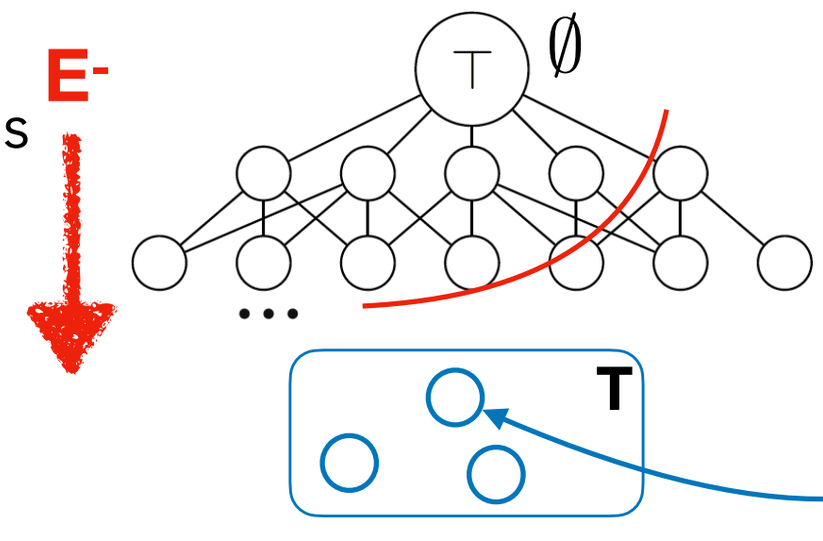
Positive example
Bottom-up inference



ACQUISITION USING MEMBERSHIP QUERIES

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

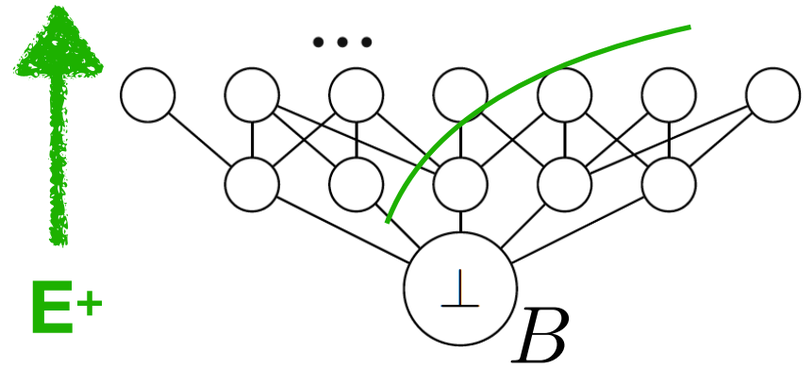
- SAT-Based constraint acquisition
- Bidirectional inference using Membership queries
- Conacq1.0 (passive learning)
- Conacq2.0 (active learning)
- Connexion between literals l_i and constraints C_i
  - $l_i = 1 \Rightarrow C_i \in L$



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

Positive example
Bottom-up inference

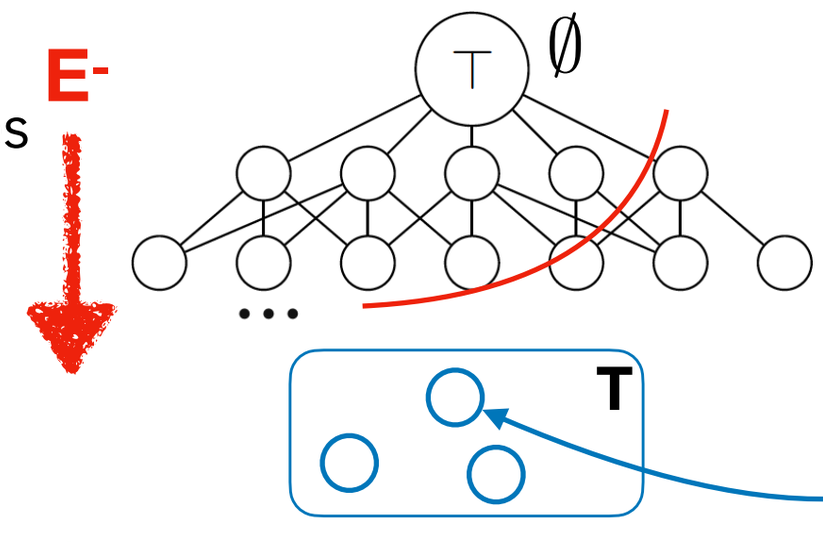
Negative example
Top-down inference



ACQUISITION USING MEMBERSHIP QUERIES

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

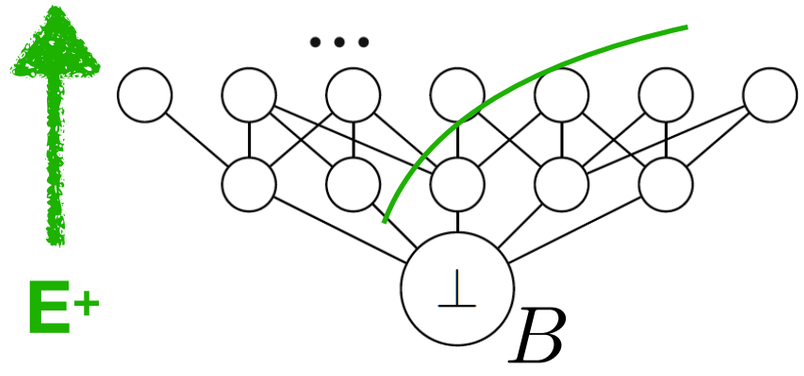
- SAT-Based constraint acquisition
- Bidirectional inference using Membership queries
- Conacq1.0 (passive learning)
- Conacq2.0 (active learning)
- Connexion between literals l_i and constraints C_i
  - $l_i = 1 \Rightarrow C_i \in L$



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

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 $\quad \kappa(e) \leftarrow \{c \in \mathbf{B} \mid \boldsymbol{x}(e)$ 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

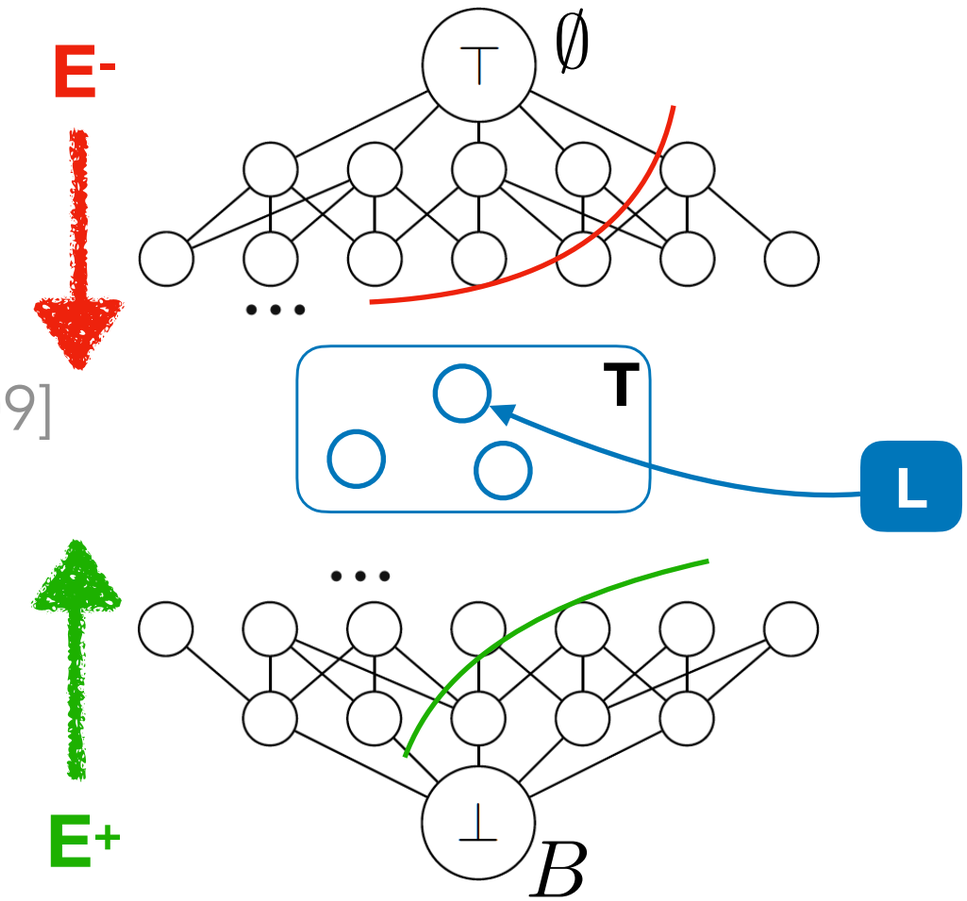
Input: a basis **B**, a background knowledge K , a strategy *Strategy*
Output: a clausal theory Ω encoding the target network

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

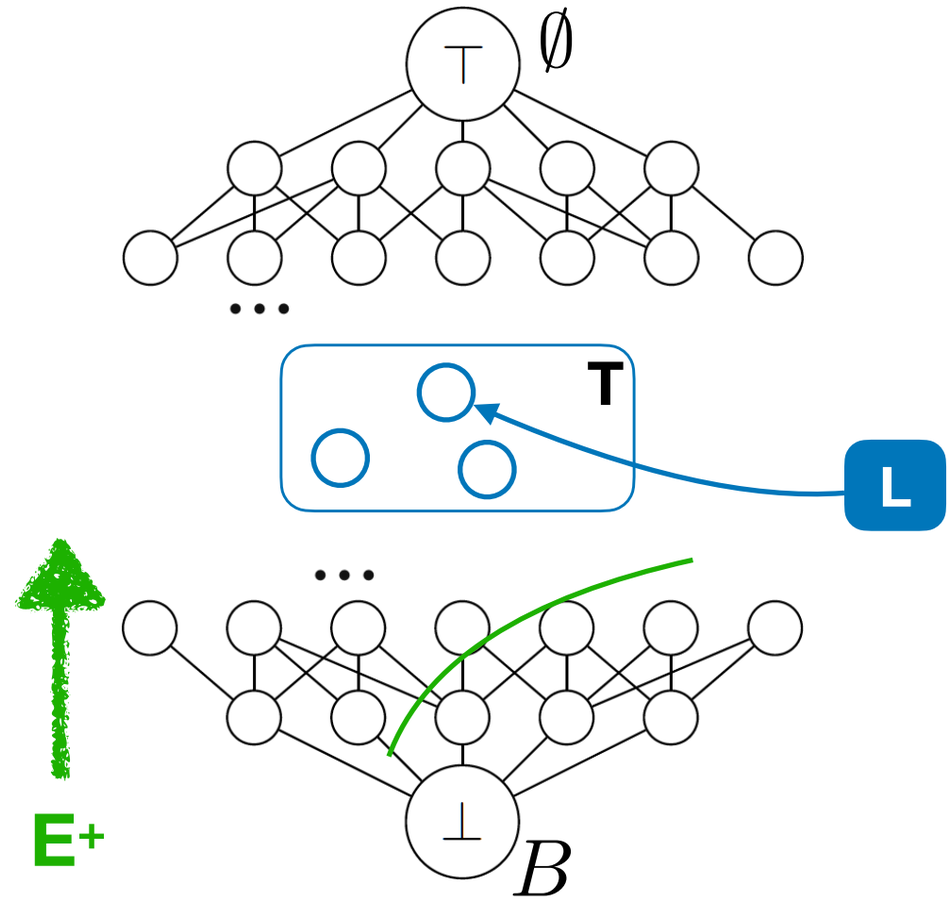
ACQUISITION USING COMPLEX QUERIES

- 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

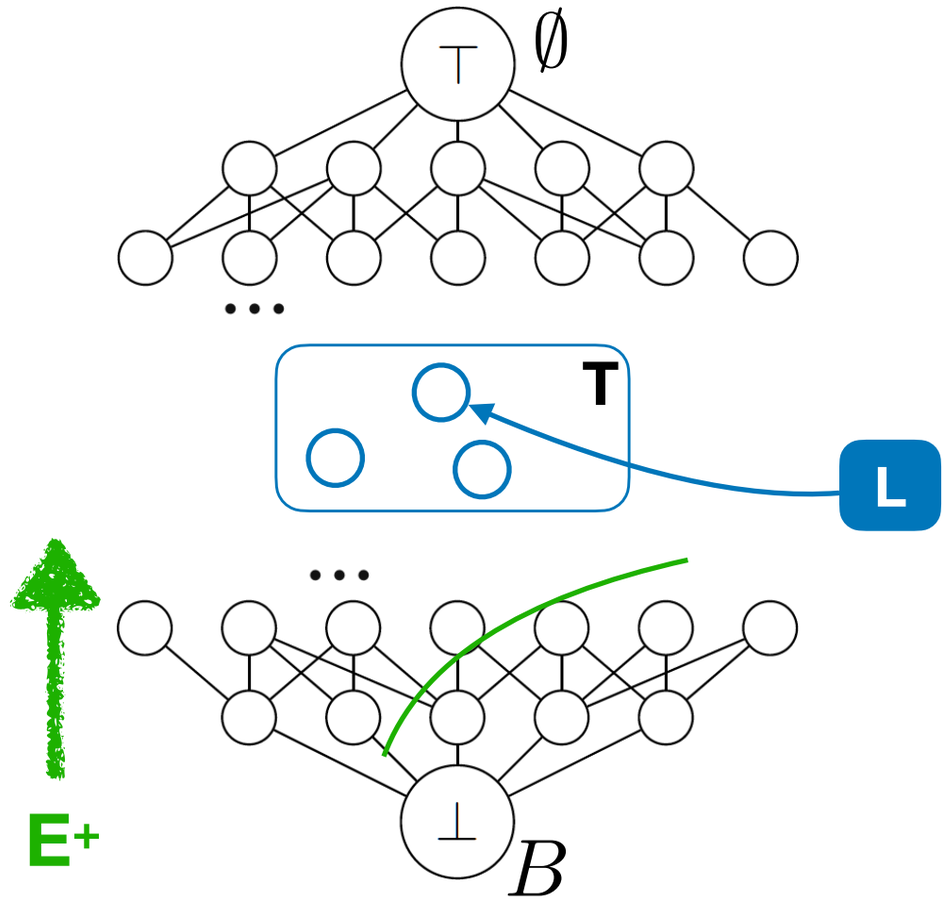
- ModelSeeker [Beldiceanu and Simonis, CP'11'12]
  - A passive learning
  - Based on global constraint catalogue (≈1000)
  - Buttom-up inference
  - ModelSeeker learns constraints underlying the scheduling of the Bundesliga (the German Football Liga) from just **one** example.



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.



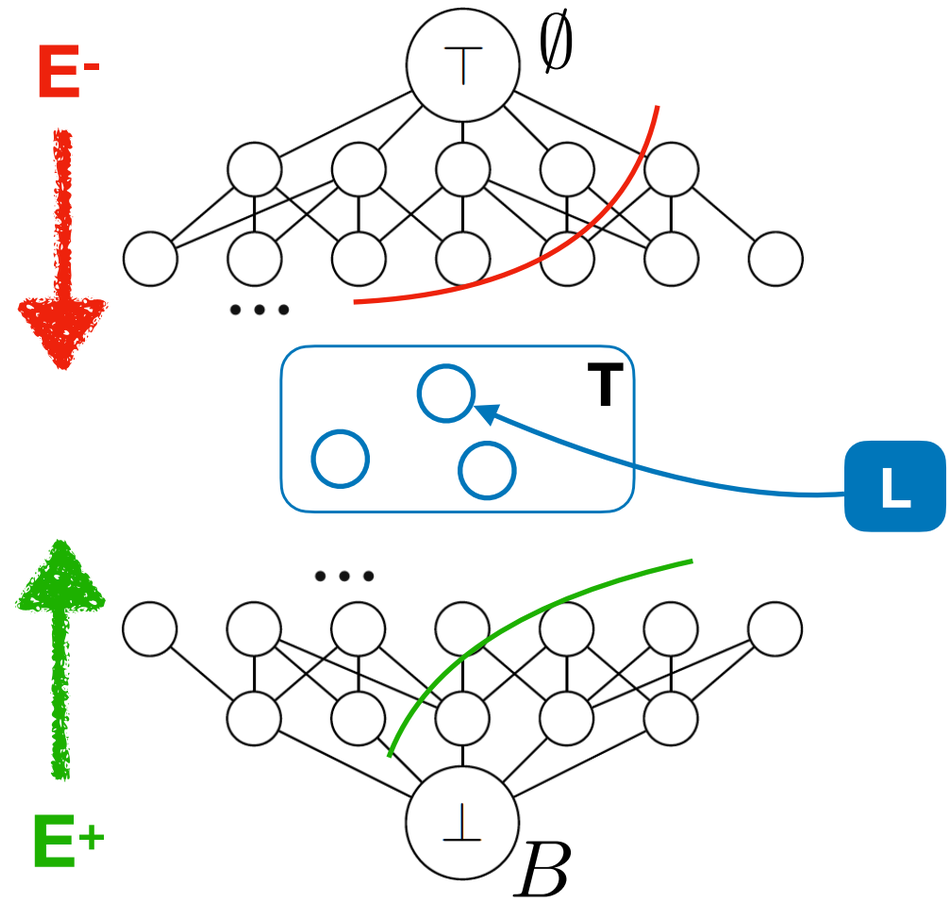
1						5	2	
				7	8			
						6		
	9			4				
			5			1		
	7							
		6	2					
	4						7	8
								3

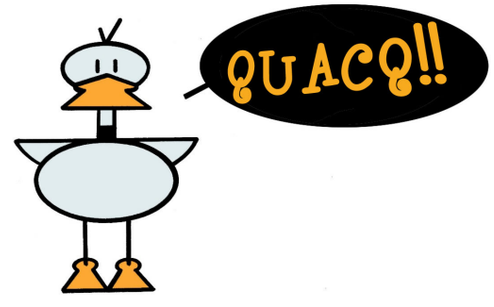
2	4	9			3			
	8			1	2	4		
						9		
						7	2	4
	1						3	
3	9	4						
		8						
		6	4	5			9	
			1			8	6	5

QUACQ

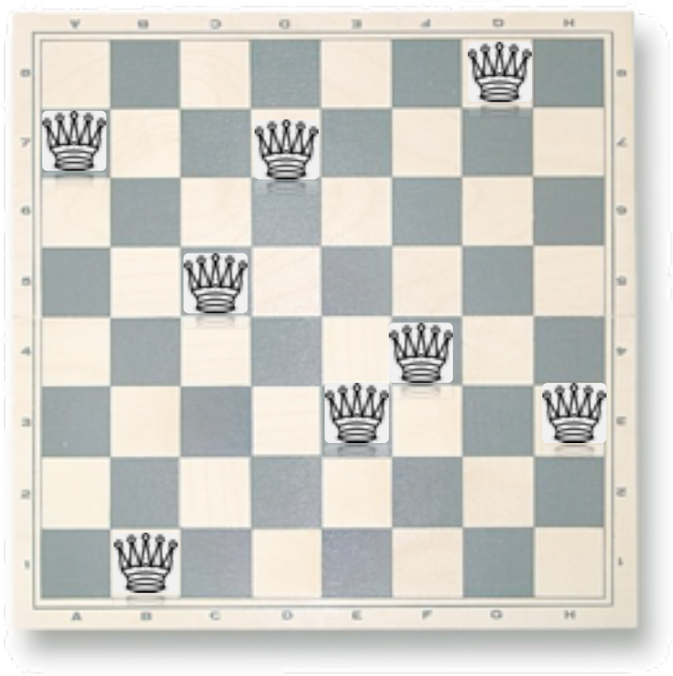
QUACQ: QUICK ACQUISITION

- QUACQ [Bessiere et al. IJCAI13]
  - 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



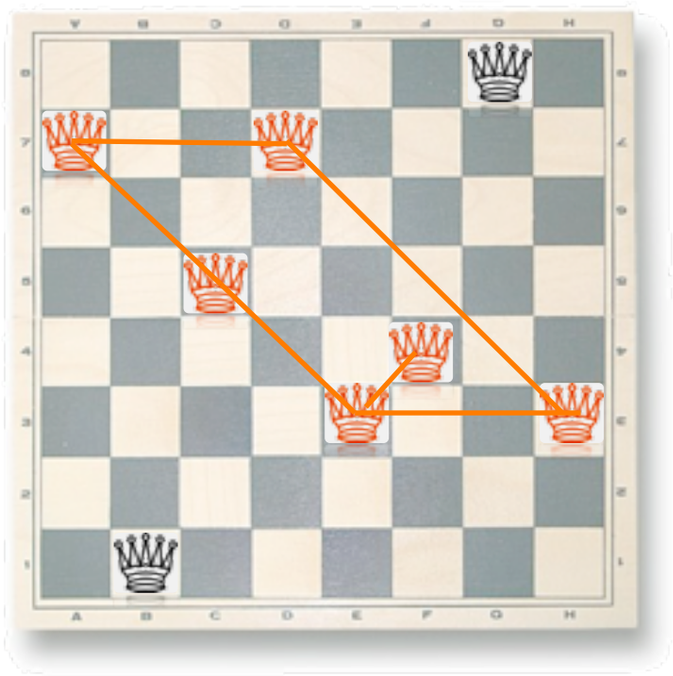


MEMBERSHIP QUERIES



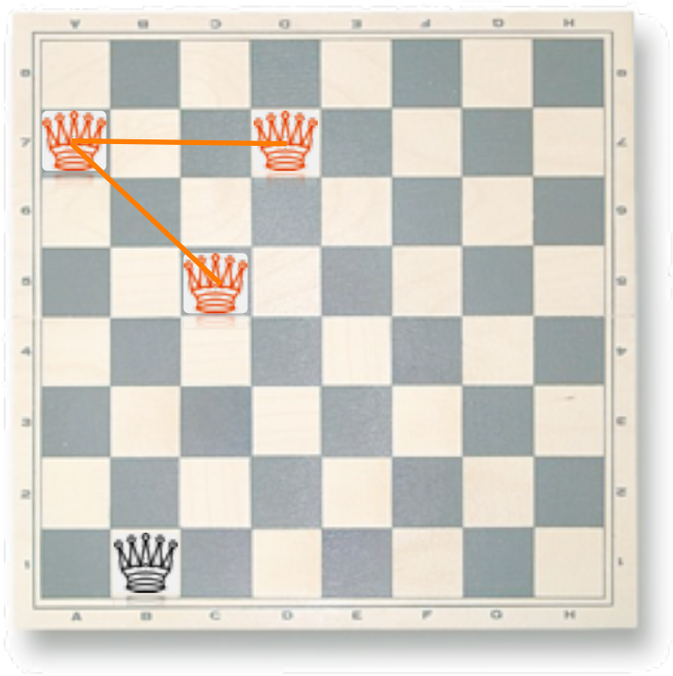
ask(2, 8, 4, 2, 6, 5, 1, 6)

MEMBERSHIP QUERIES



ask(2, 8, 4, 2, 6, 5, 1, 6) = No

PARTIAL QUERIES

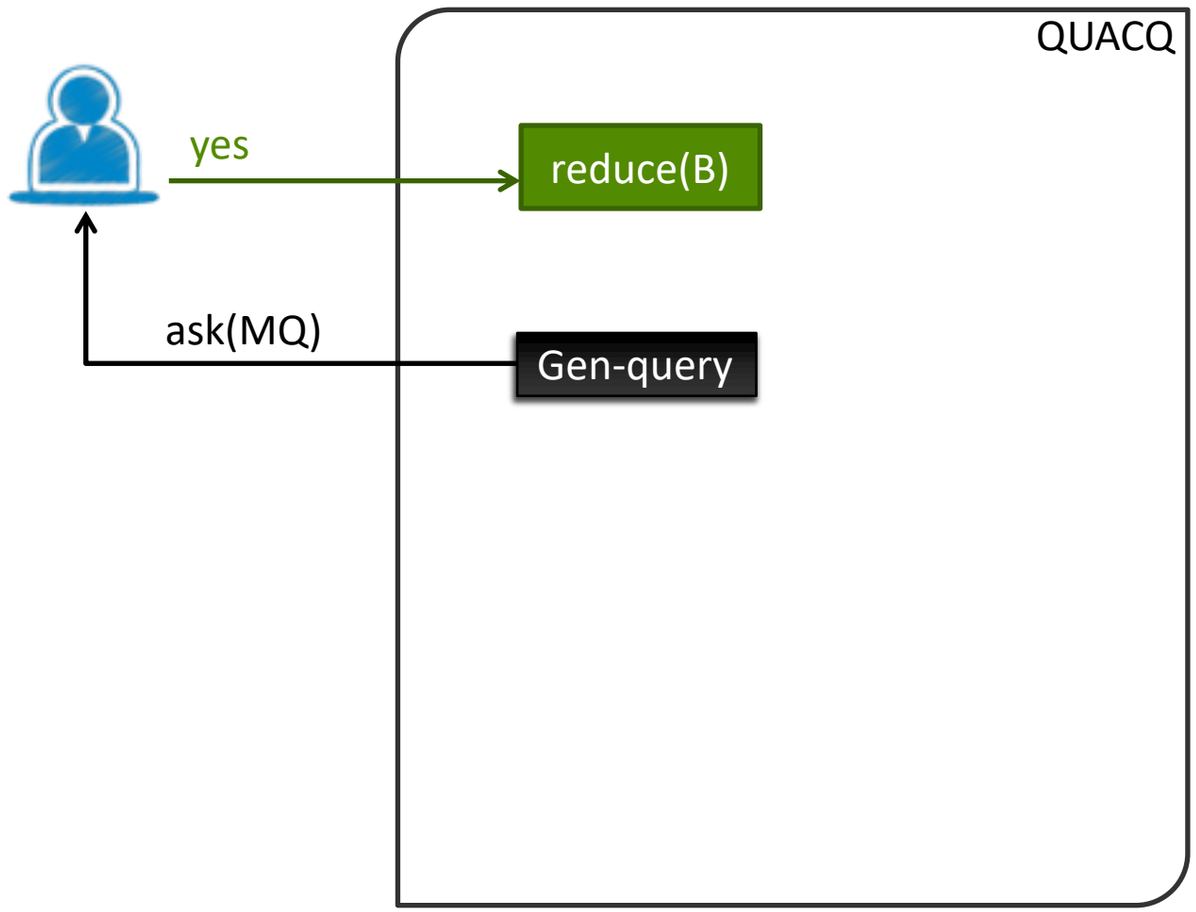


ask(2, 8, 4, 2, -, -, -, -) = No

PARTIAL QUERIES

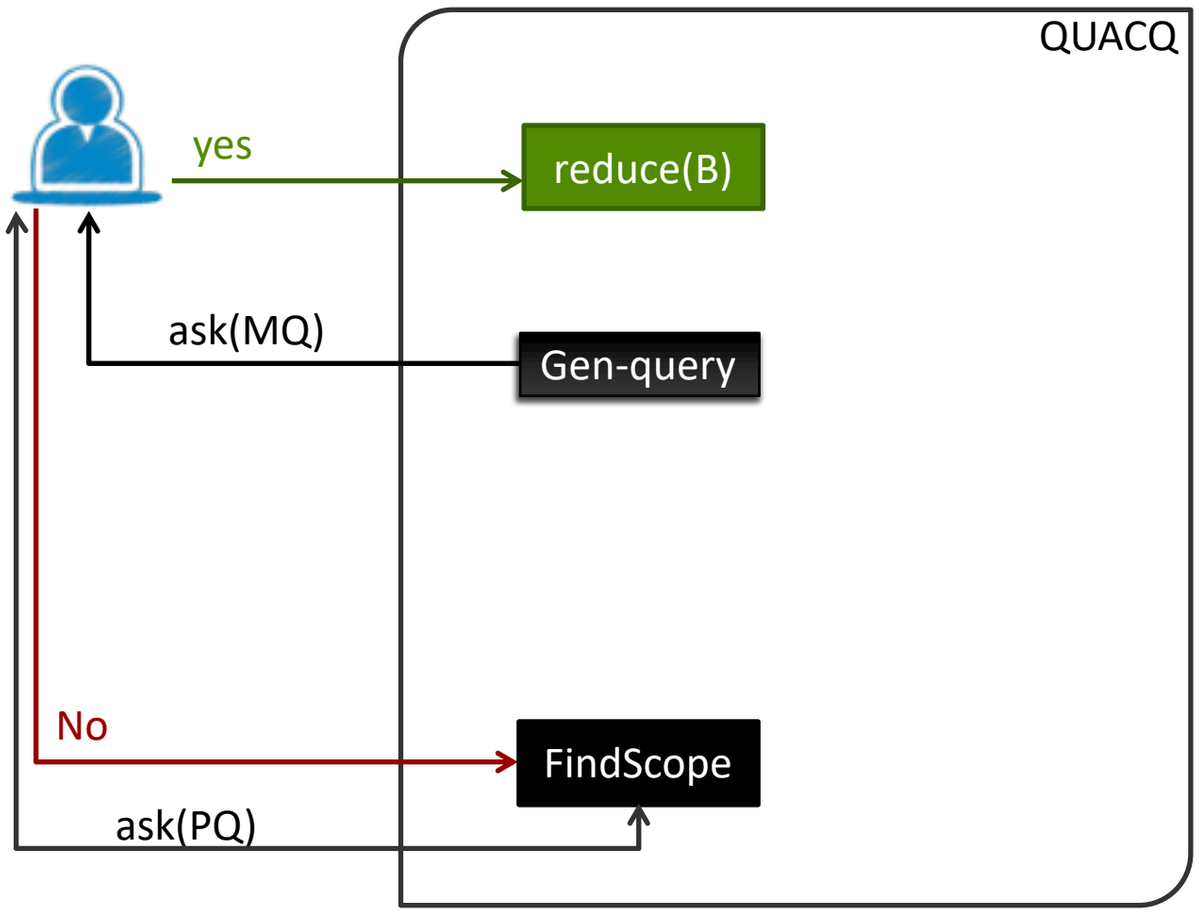
ask(2, 8, -, -, -, -, -, -) = Yes

QUACQ: QUICK ACQUISITION



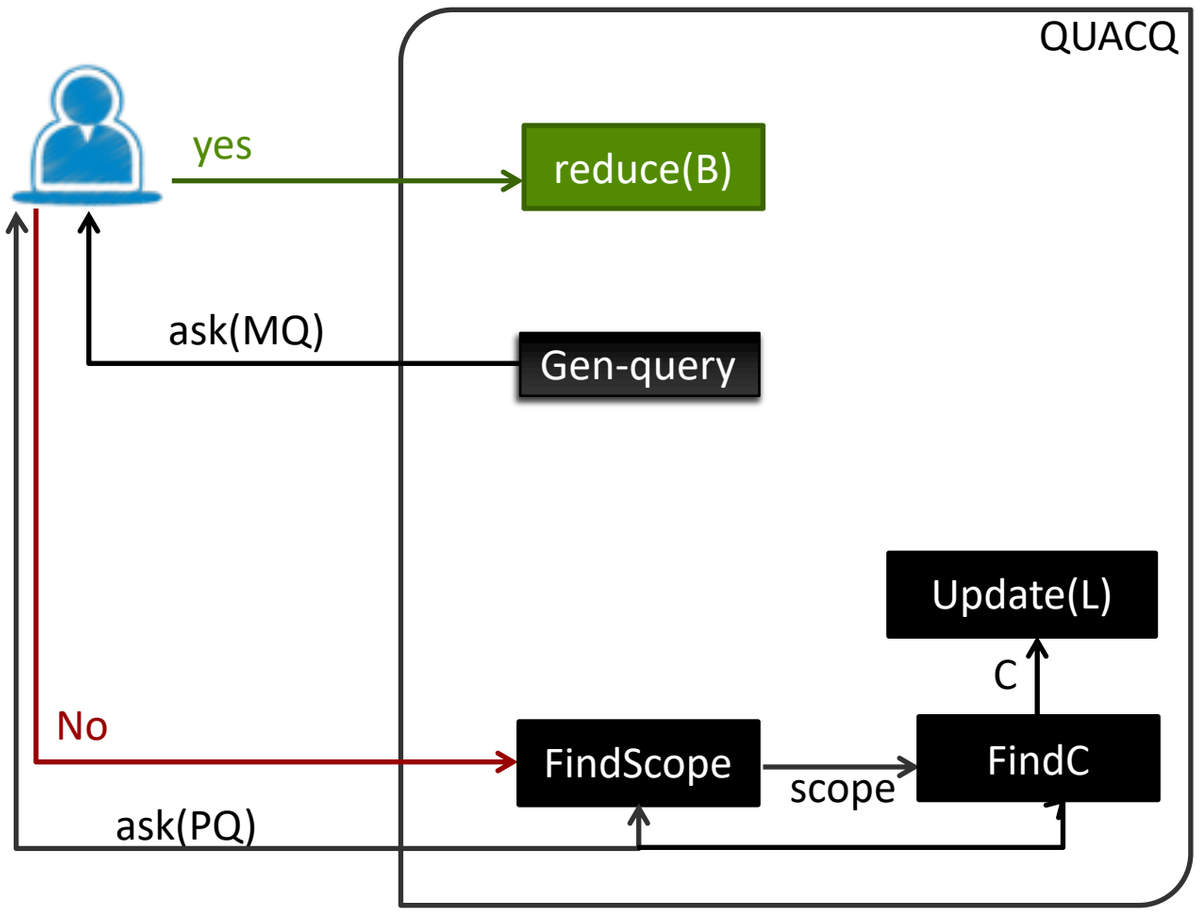
MQ: Membership Query
PQ: Partial Query

QUACQ: QUICK ACQUISITION



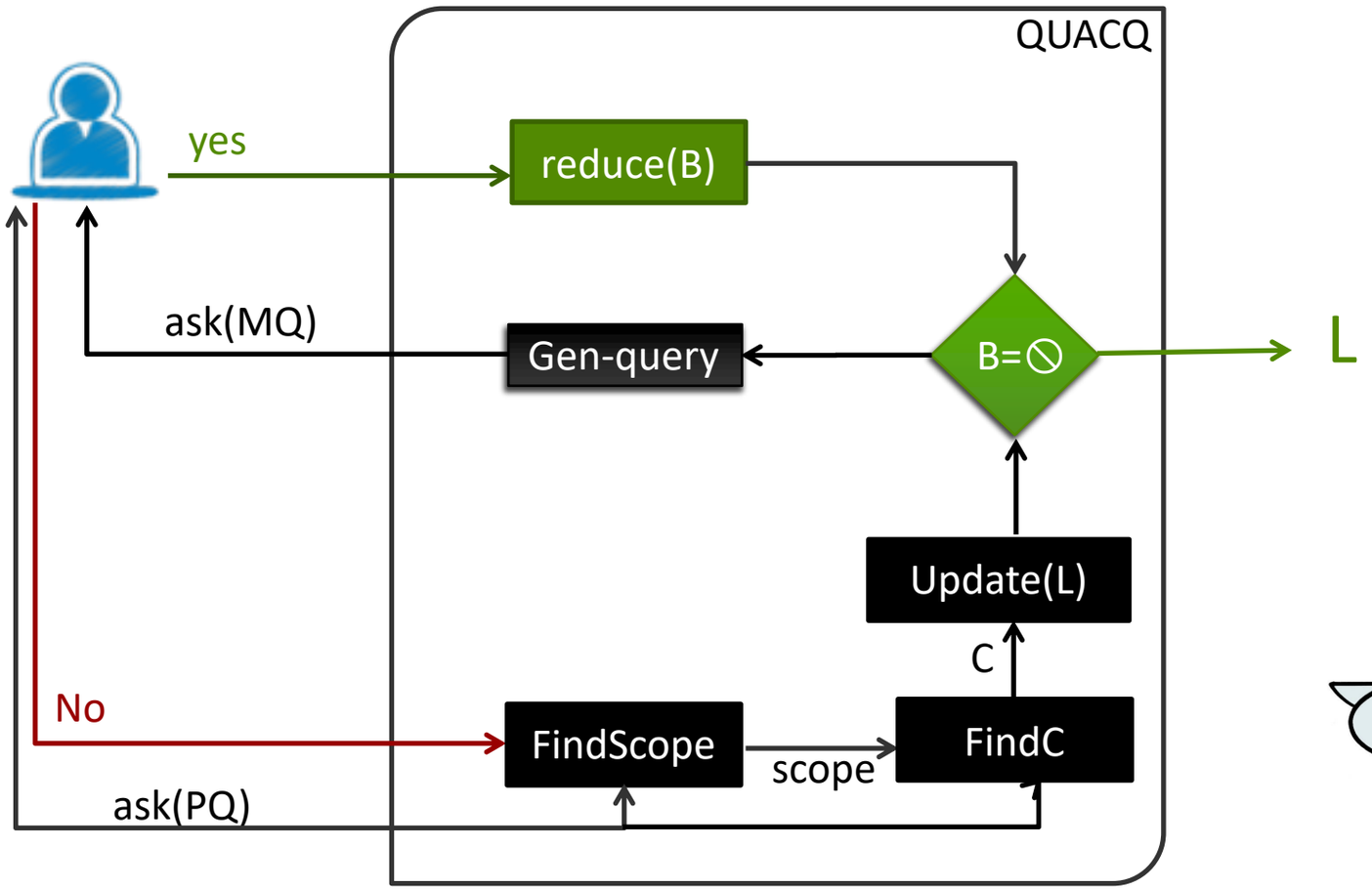
MQ: Membership Query
PQ: Partial Query

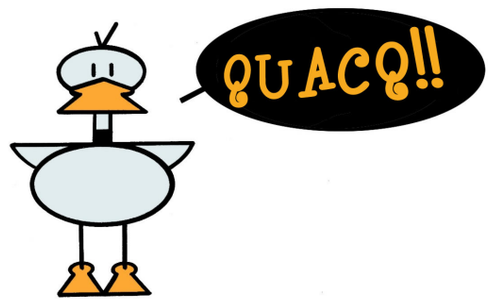
QUACQ: QUICK ACQUISITION



MQ: Membership Query
PQ: Partial Query

QUACQ: QUICK ACQUISITION





MQ: Membership Query
PQ: Partial Query

QUACQ: QUICK ACQUISITION

Algorithm 3 : QuAcq: asks partial queries and returns a network L

In : A basis B
Out : A learned network L

```
1 begin
2     L ← ∅;
3     while true do
4         e_Y ← GenerateExample(X, L, B) ;
5         if e_Y = nil then return "convergence on L";
6         if ASK(e_Y) = yes then
7             B ← B \ κ_B(e_Y);
8         else FindC(e_Y, FindScope(e_Y, ∅, Y), L) ;
```

QUACQ: QUICK ACQUISITION

Algorithm 3 : QuAcq: asks partial queries and returns a network L

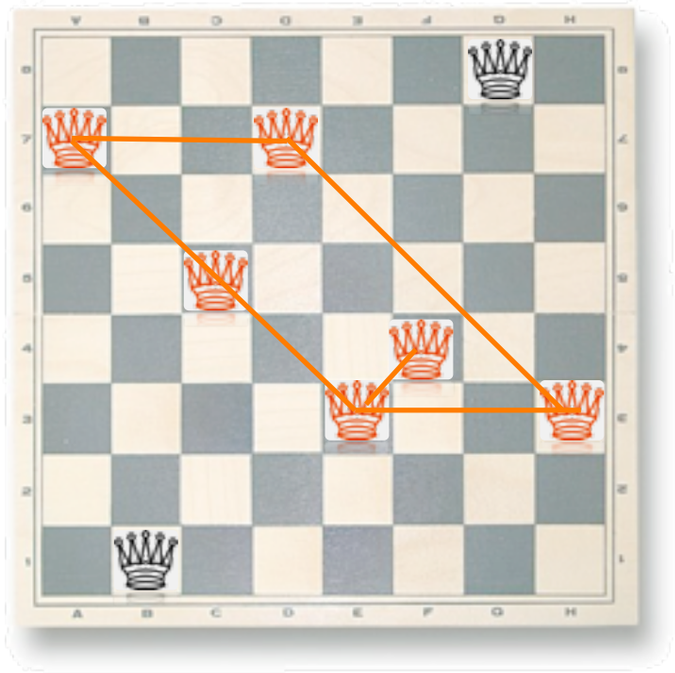
In : A basis B
Out : A learned network L

```
1 begin
2     L ← ∅;
3     while true do
4         e_Y ← GenerateExample(X, L, B) ;
5         if e_Y = nil then return "convergence on L";
6         if ASK(e_Y) = yes then
7             B ← B \ κ_B(e_Y);
8         else FindC(e_Y, FindScope(e_Y, ∅, Y), L) ;
```

Computes an assignment on a subset of variables Y satisfying the constraints of L that have a scope included in Y, but violating at least one constraint from B

findScope

QUACQ: FINDSCOPE (SKETCH)



ask(2, 8, 4, 2, 6, 5, 1, 6) = No

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)= 2 ✓

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],…)

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, CPAIOR'18])

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):**
  - Parallel Constraint Acquisition (future work)
  - Deep Constraint Acquisition (future work)

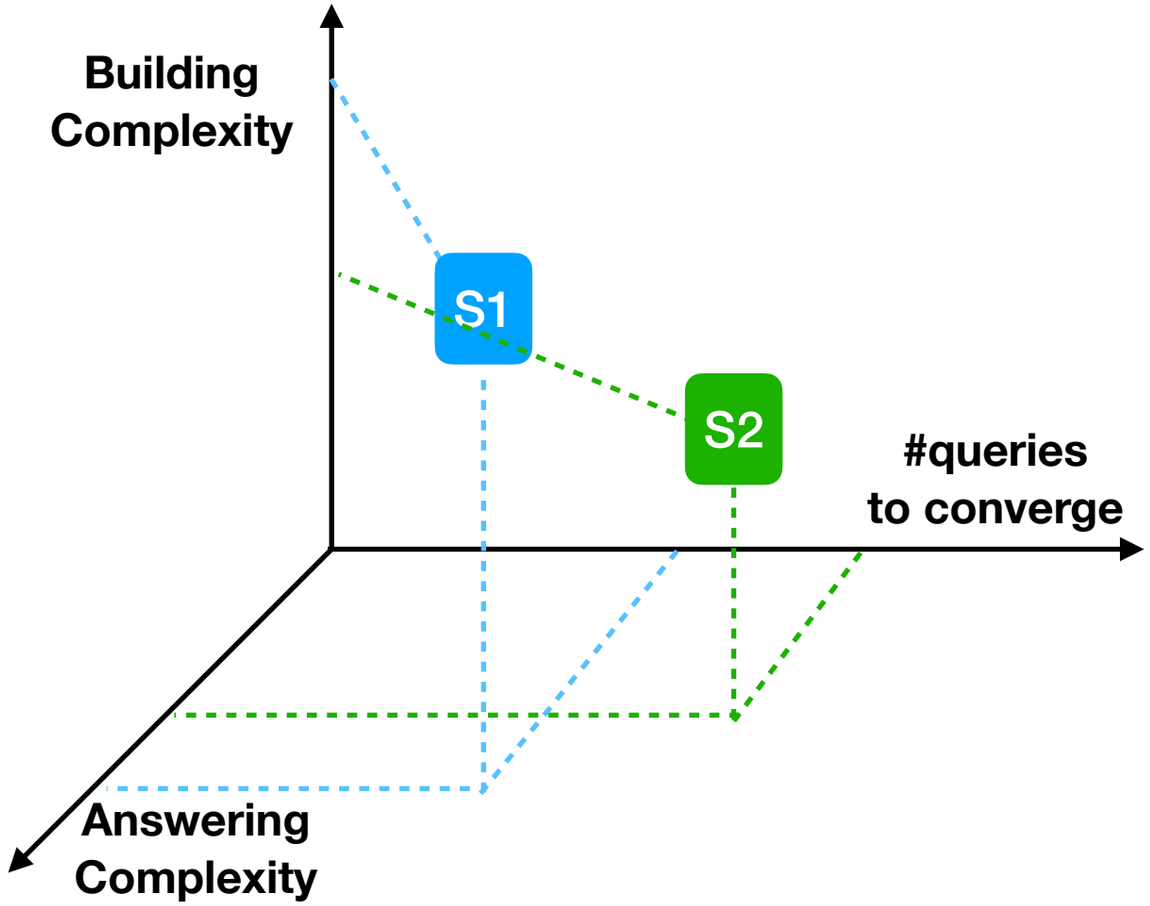
QUACQ LIMITATIONS

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

http://info-demo.lirmm.fr:8080/web-acq/

https://gite.lirmm.fr/constraint-acquisition-team/quacq

CONSTRAINT
PROGRAMMING
2021
MONTPELLIER