

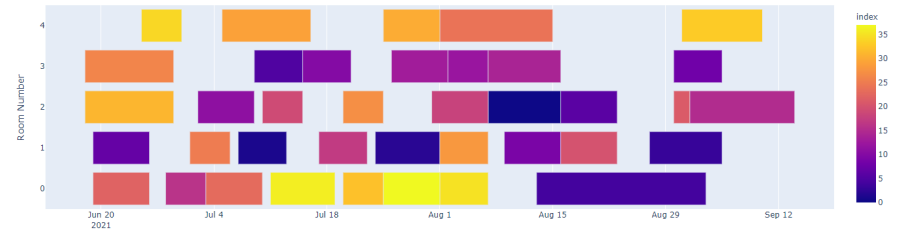
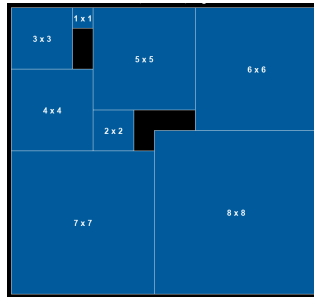
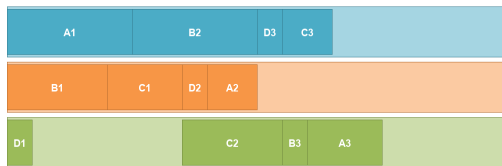
A Tutorial on Constraint Acquisition

Dimos Tsouros
dimos.tsouros@kuleuven.be

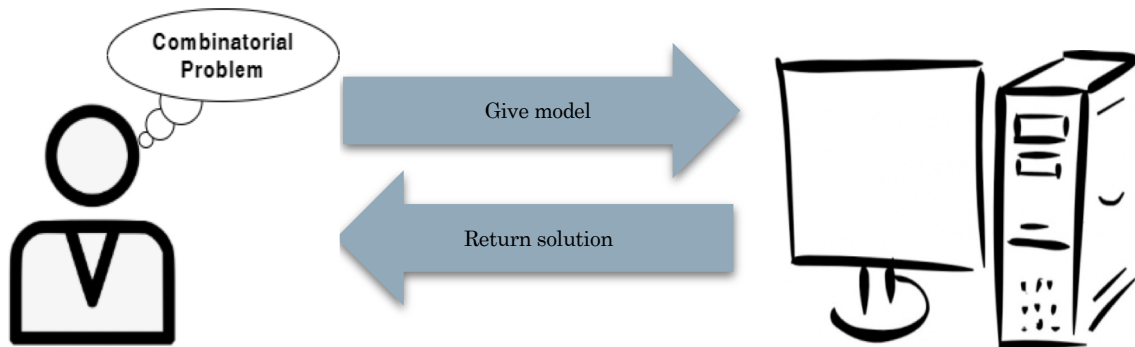
26/02/2025

Introduction

- ❖ Constraint programming (CP)
 - ❑ Solving combinatorial problems in AI

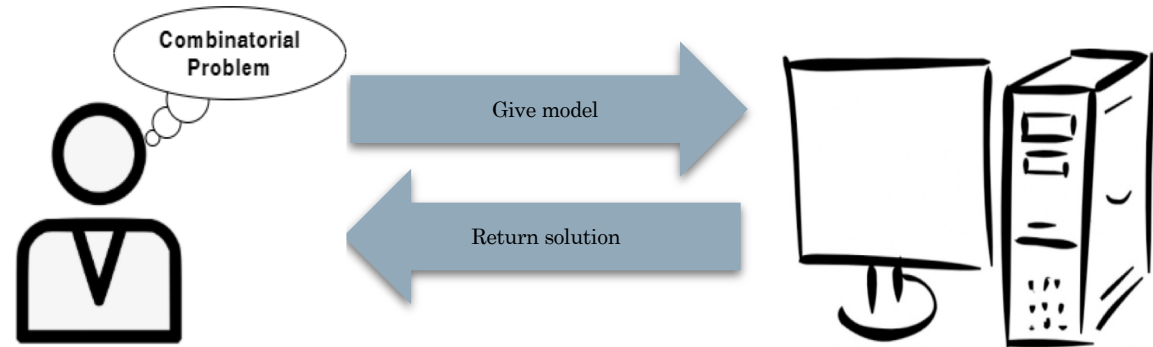


- ❖ Model + Solve paradigm



Introduction

- ❖ Model + Solve paradigm



- Variables with their domains

```
# Variables
grid = cp.intvar(1, grid_size, shape=(grid_size, grid_size), name="grid")
```

- Constraints of the problem:

```
# Constraints on rows and columns
for row in grid:
    model += cp.AllDifferent(row)

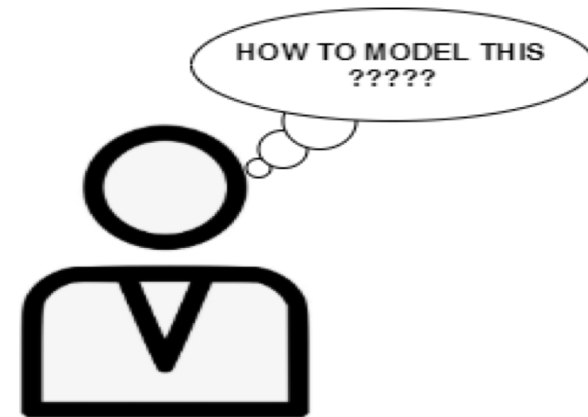
for col in grid.T: # numpy's Transpose
    model += cp.AllDifferent(col)

# Constraints on blocks
for i in range(0, grid_size, block_size_row):
    for j in range(0, grid_size, block_size_col):
        model += cp.AllDifferent(grid[i:i + block_size_row, j:j + block_size_col])
```

Introduction

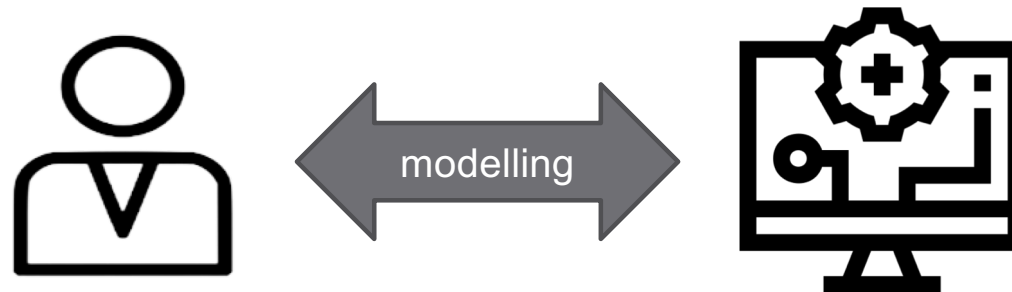
Modelling is not always trivial

- Requires expertise
- Bottleneck for the wider use of CP

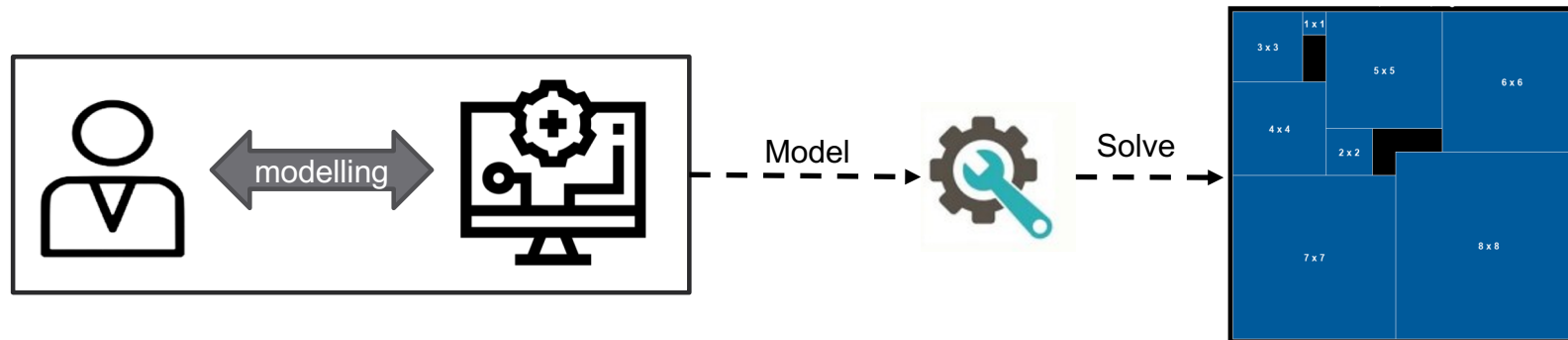


Constraint Acquisition

Structure Learning: learn the constraints of the problem, not parameters

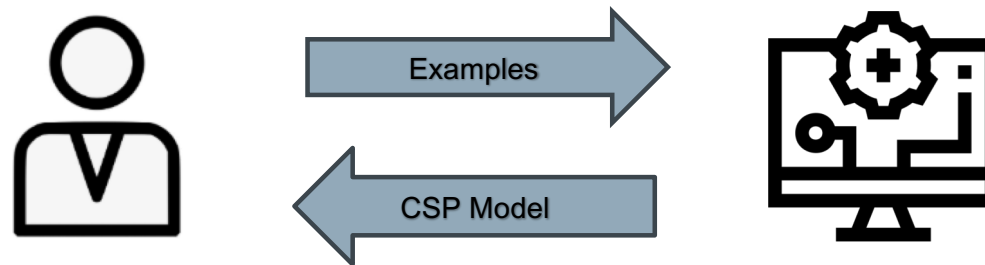


Introduction

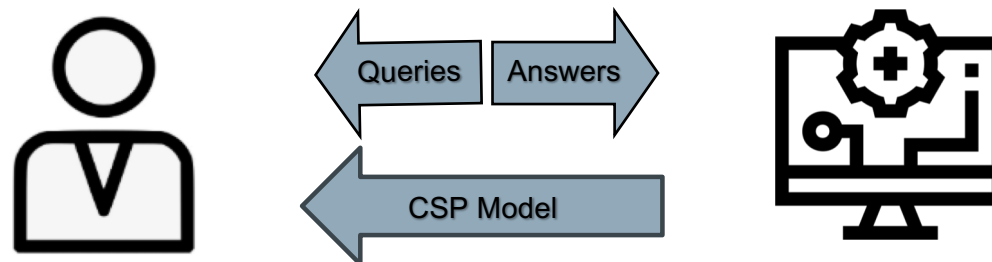


Constraint Acquisition

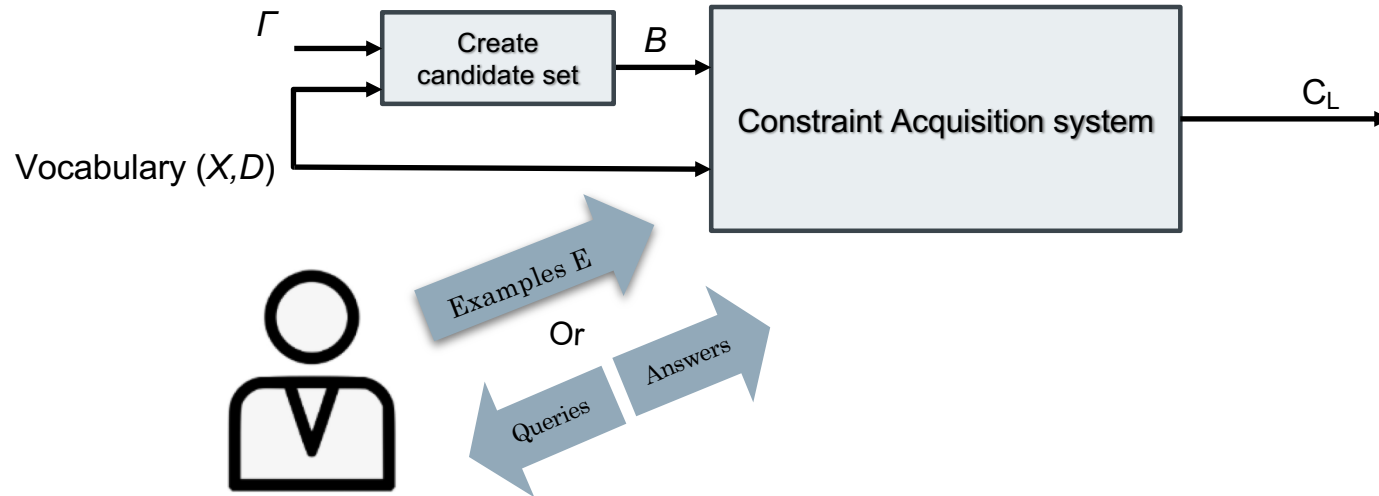
Passive acquisition: Using existing data



(Inter)active acquisition: Interact with the user



Constraint Acquisition



Terminology

X : Variables, D^X : Domains

Γ : Constraint Language: abstract relations

B : Set of candidate constraints (using relations from Γ)

C_L : Learned constraint set

Goal:

C_L is equivalent to the (unknown) target set C_T

Inputs and Outputs

Running Example: 4x4 Sudoku

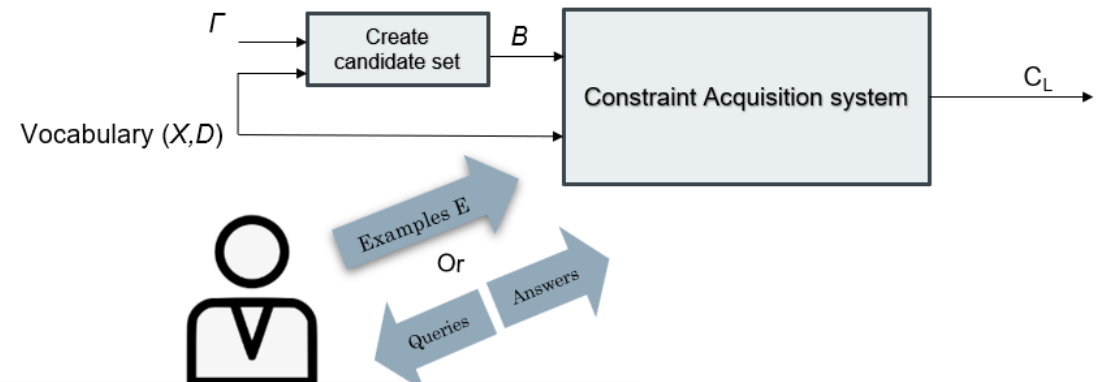
X : Variables of the problem, $X = \{x_{1,1}, x_{1,2}, x_{1,3}, \dots, x_{4,4}\}$, domains $D = \{1, 2, 3, 4\}$
 Γ : Constraint Language, $\Gamma = \{=, \neq, \leq, \geq\}$ ← The user may not know exactly what constraint relations appear in the model
 B : Set of candidate constraints, $B = \{x_{1,1} \neq x_{1,2}, x_{1,1} = x_{1,2}, \dots\}$
 C_L : learned constraints, ideally all the $\neq$ constraints in rows, cols and blocks

Problem Formulation

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$
$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$

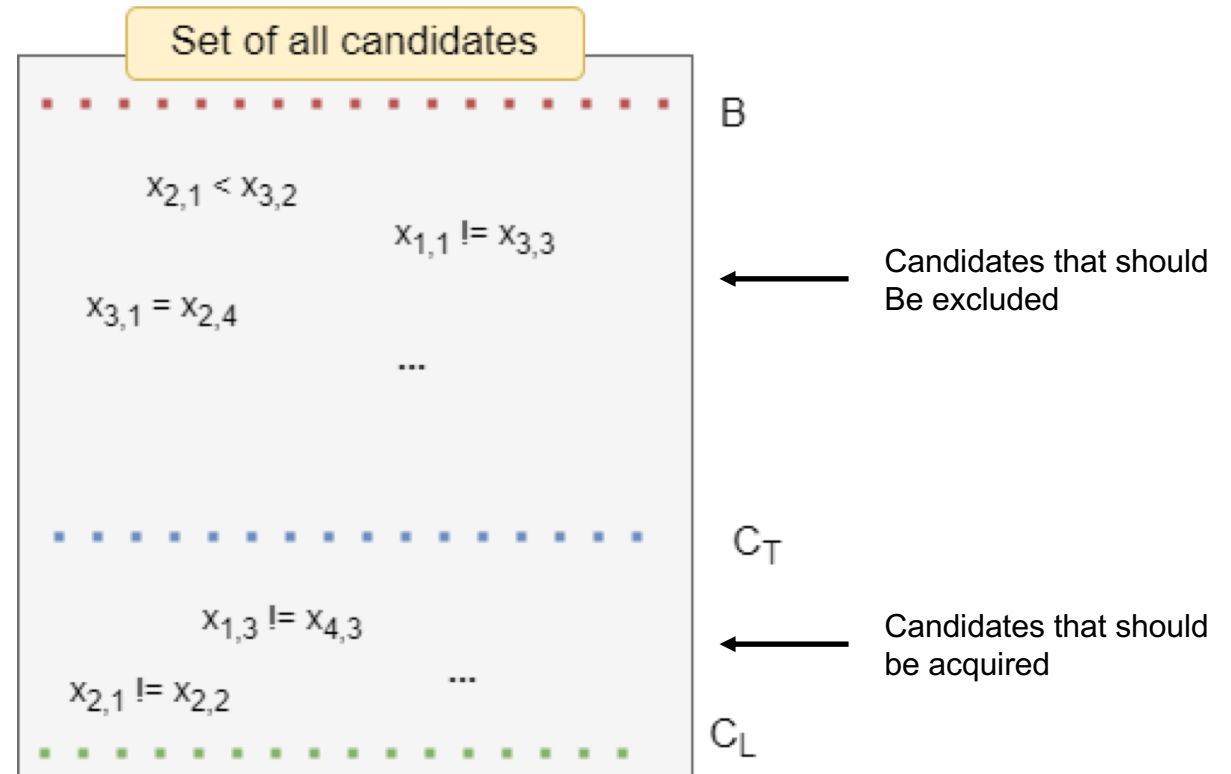
Example e_1

1	1	3	4
3	2	1	1
2	2	3	1
2	3	4	3



Adapting Candidate Elimination

- **B**: set of (remaining) candidate constraints
- **C_T**: target set of constraints
- **C_L**: learned set of constraints

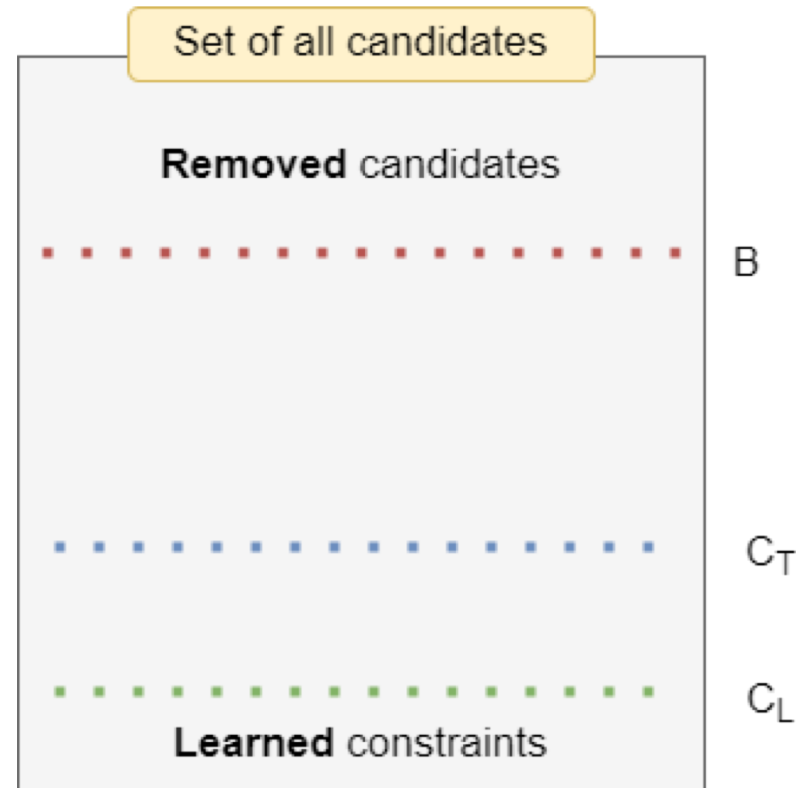


Adapting Candidate Elimination

During the learning process:

- Constraints are removed from B
- Constraints are added to C_L

- B : set of (remaining) candidate constraints
- C_T : target set of constraints
- C_L : learned set of constraints

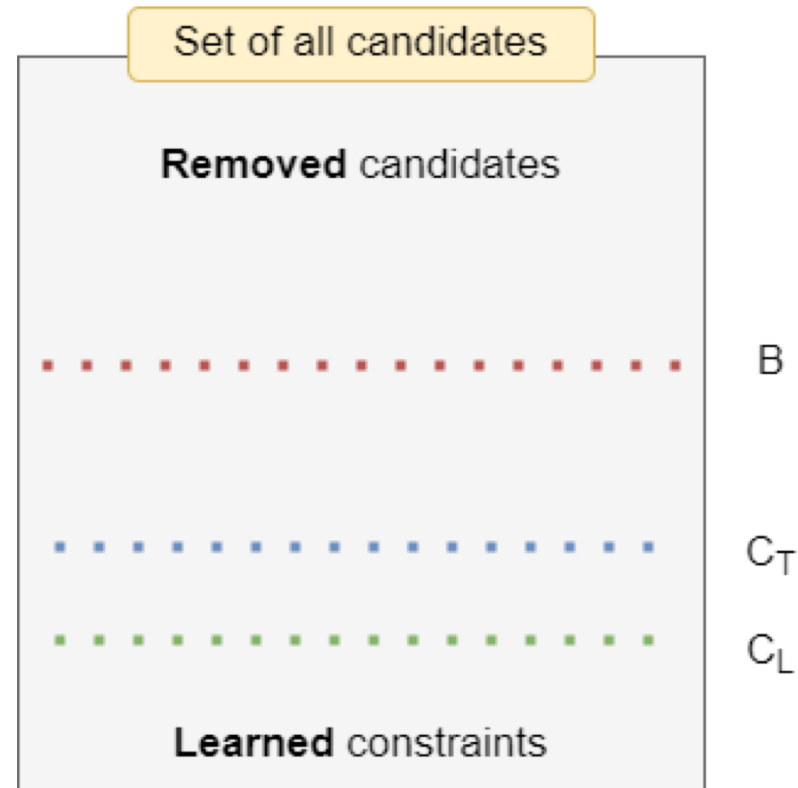


Adapting Candidate Elimination

During the learning process:

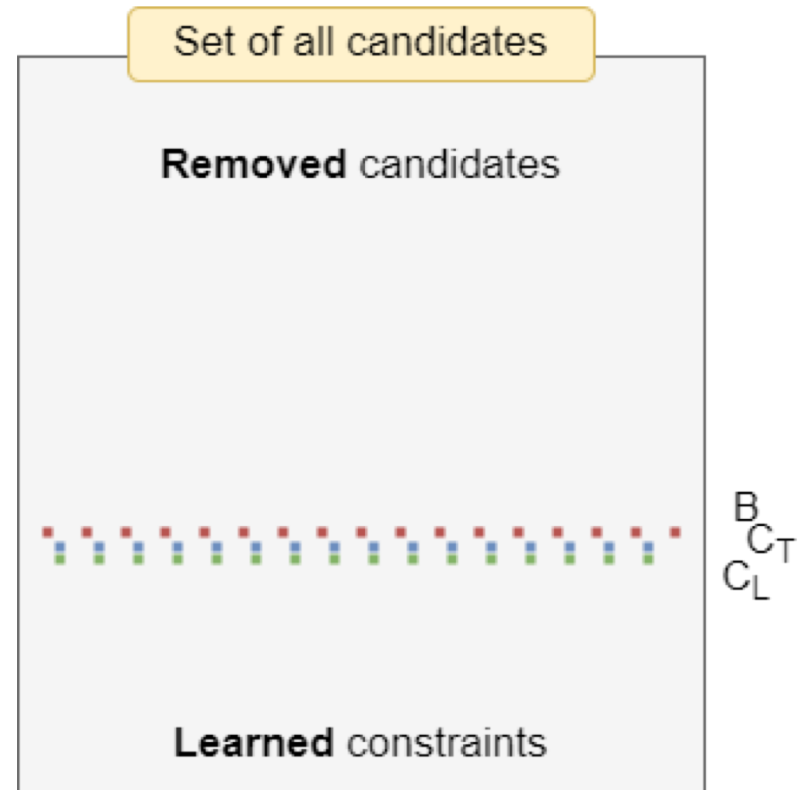
- Constraints are removed from B
- Constraints are added to C_L

- B : set of (remaining) candidate constraints
- C_T : target set of constraints
- C_L : learned set of constraints



Adapting Candidate Elimination

If C_T is representable by B the CA system will eventually *converge* to a C_L equivalent to C_T



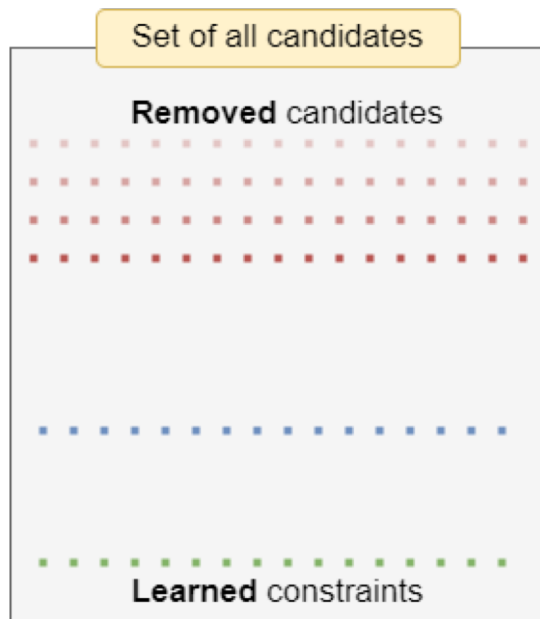
Adapting Candidate Elimination

- Examples: Assignments to the variables of the problem

Learning from *positive* examples (Solutions):

- Violated constraints cannot be part of the model
- Otherwise, it could not be a solution

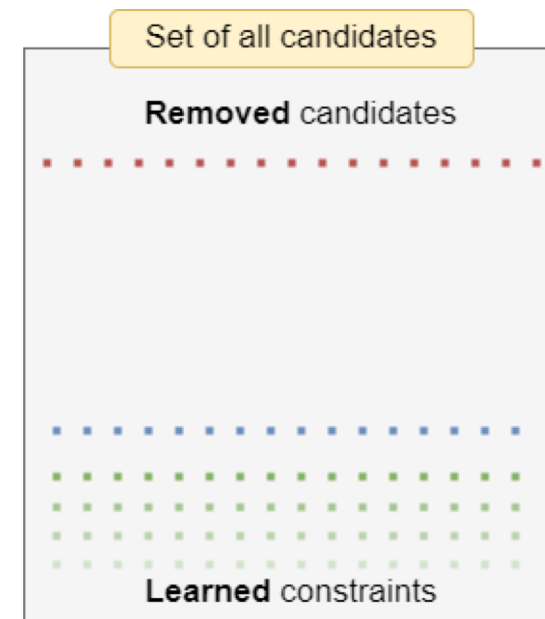
└→ Shrinking the bias



Learning from *negative* examples (Non-solutions):

- One (or more) violated constraint is a constraint of the problem
- Otherwise, it would be a solution

└→ Learning Constraints



Adapting Candidate Elimination

- Example: 4x4 sudoku, with $\Gamma = \{!=\}$

Learning from *positive* examples (Solutions):

- Violated constraints cannot be part of the model
- Otherwise, it could not be a solution

└→ Shrinking the bias

Query: is this a solution?

4	2	1	3
3	1	2	4
2	4	3	1
1	3	4	2

Violated candidates can be safely removed:
 $\{x_{11} \neq x_{24}, x_{13} \neq x_{22}, x_{14} \neq x_{21}, \dots\}$

Answer: Yes

Learning from *negative* examples (Non-solutions):

- One (or more) violated constraint is a constraint of the problem
- Otherwise, it would be a solution

└→ Learning Constraints

Query: is this a solution?

4	2	3	2
3	1	2	4
1	3	4	2
2	4	1	3

At least one of the violated constraints is a true constraint:
 $\{x_{12} \neq x_{14}, x_{14} \neq x_{34}, x_{14} \neq x_{23}\}$

Answer: No

Passive Acquisition



Assignment to all variables of the problem, labelled as:

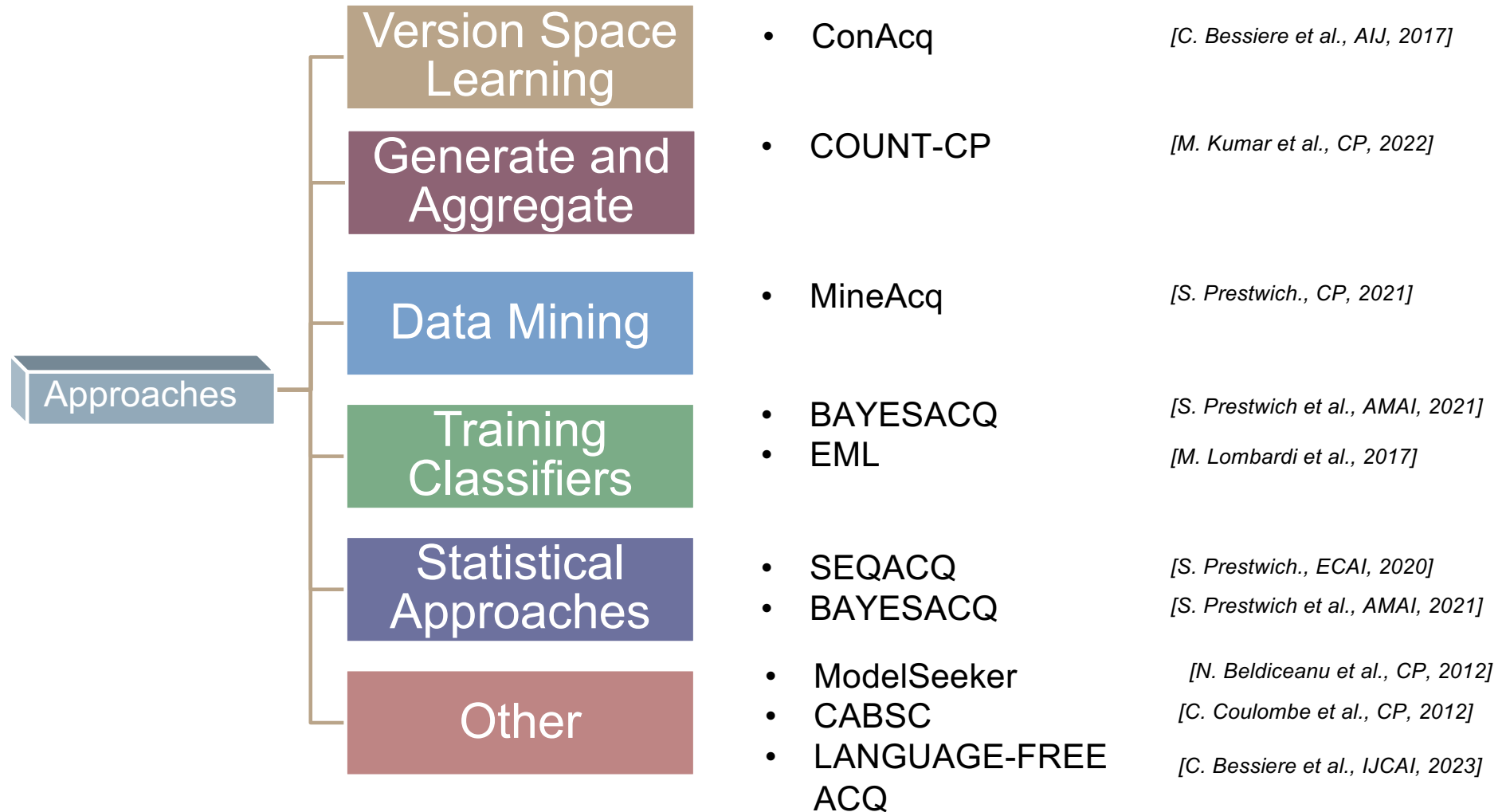
- a solution



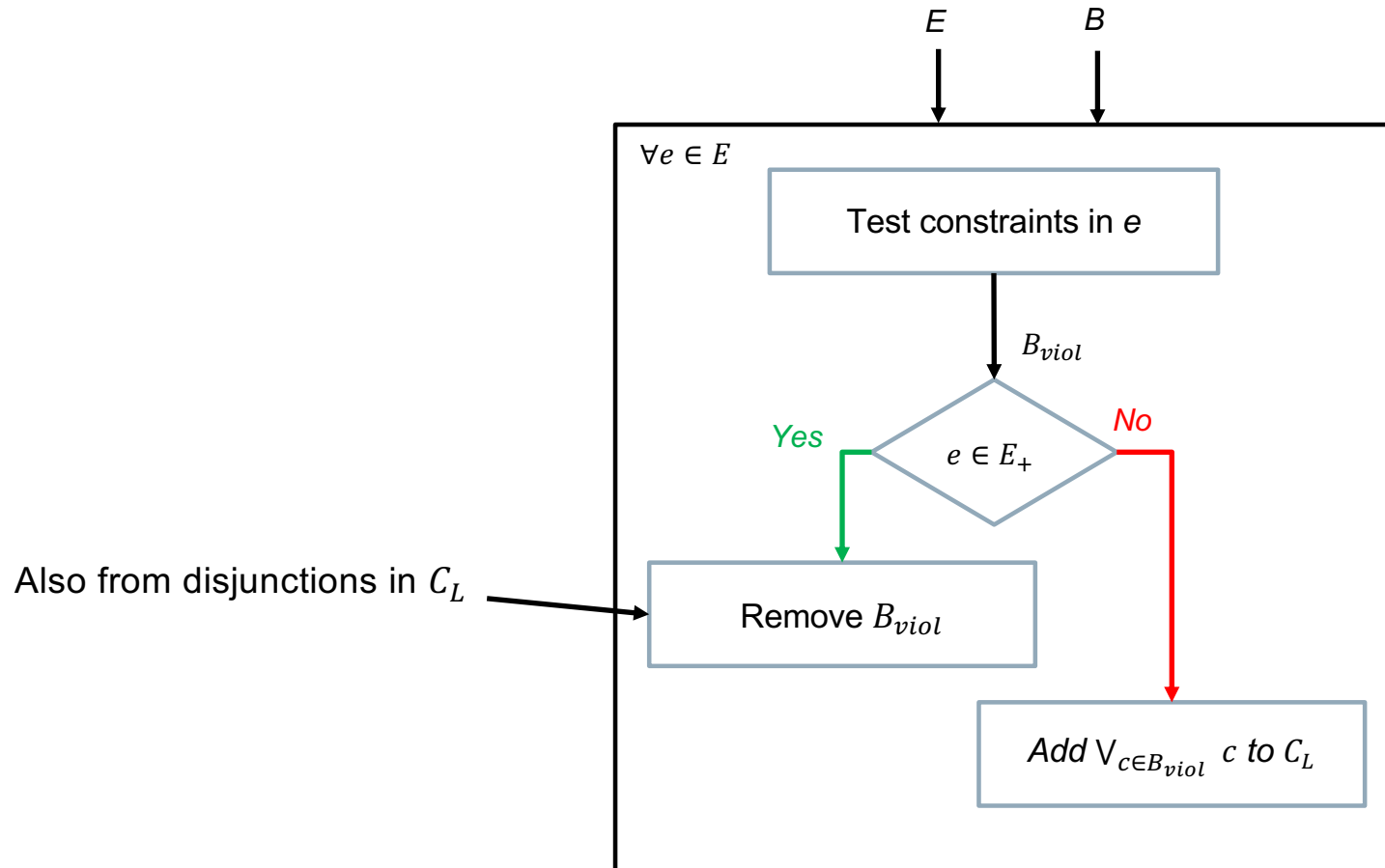
- or a non-solution



Passive Acquisition

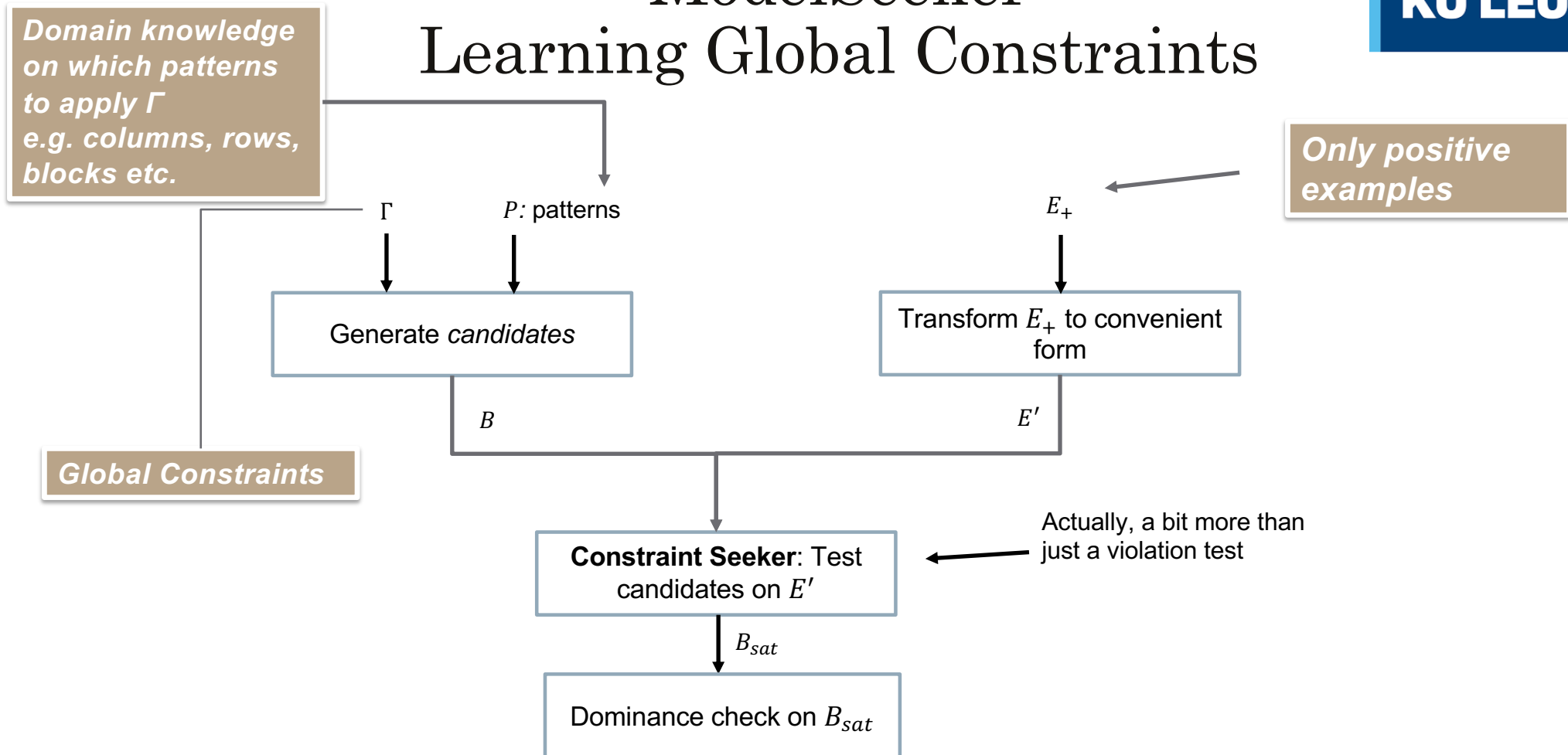


Baseline: ConAcq



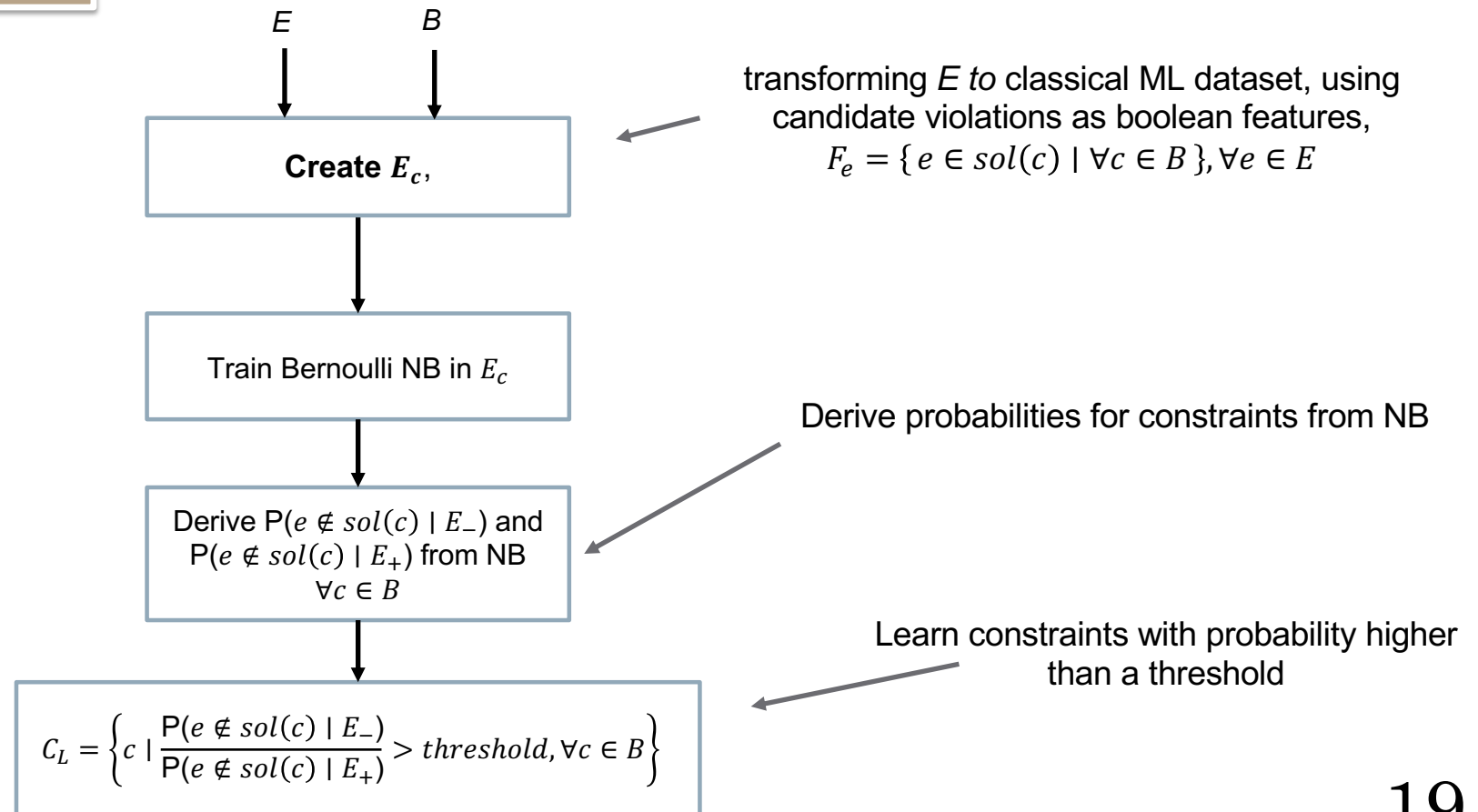
ModelSeeker

Learning Global Constraints



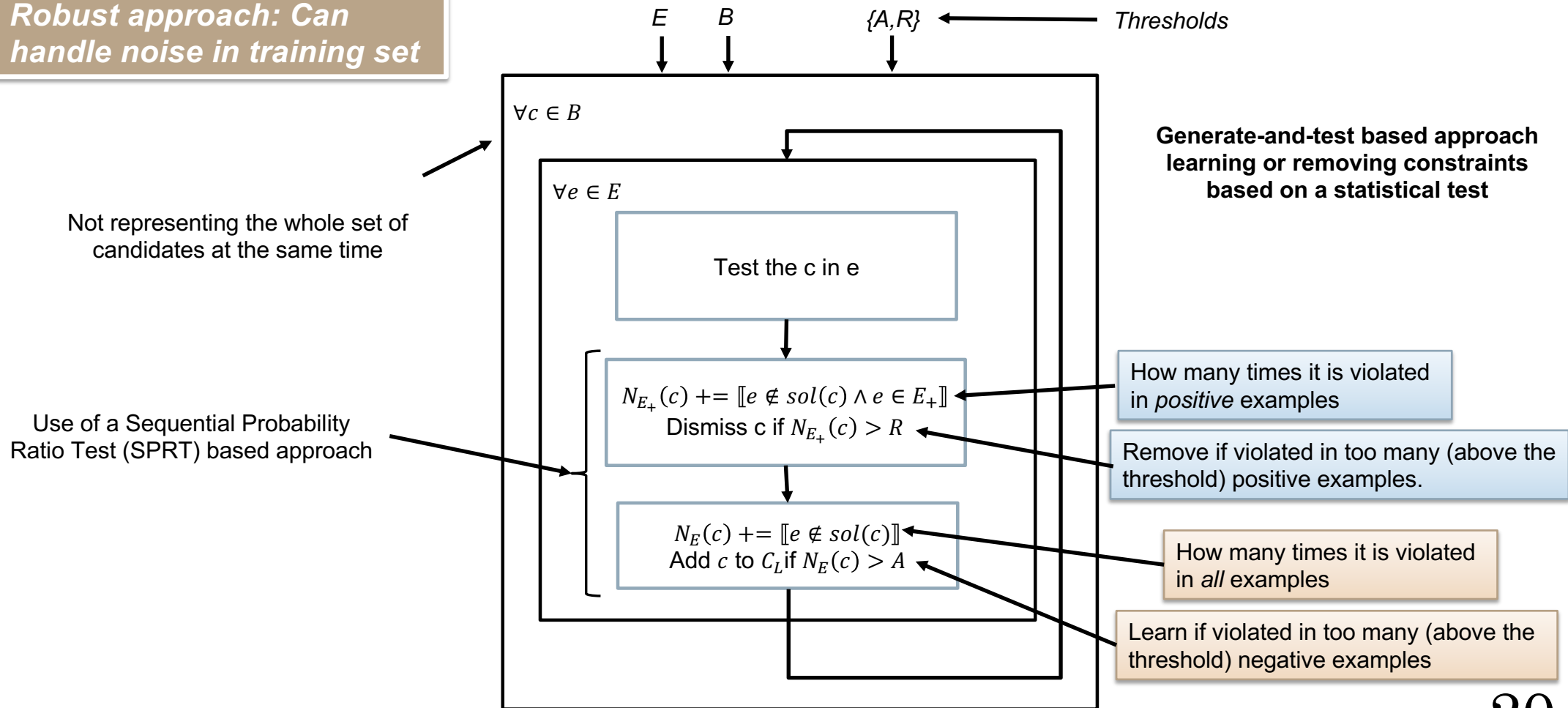
Robust approach: Can handle noise in training set

Statistical approaches – BayesAcq



Statistical approaches – SeqAcq

Robust approach: Can handle noise in training set



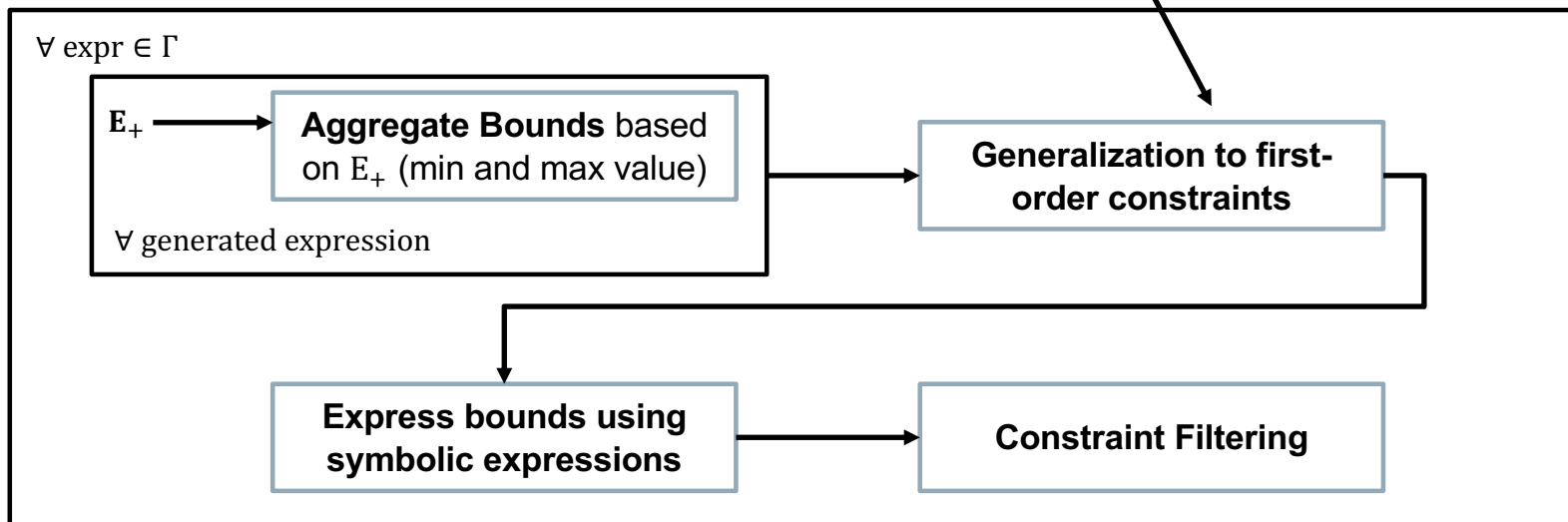
COUNT-CP: Learn using generate and aggregate

Learns constraints of the following form: $lb \leq expr \leq ub$

Aggregates the bounds of the expressions based on E_+

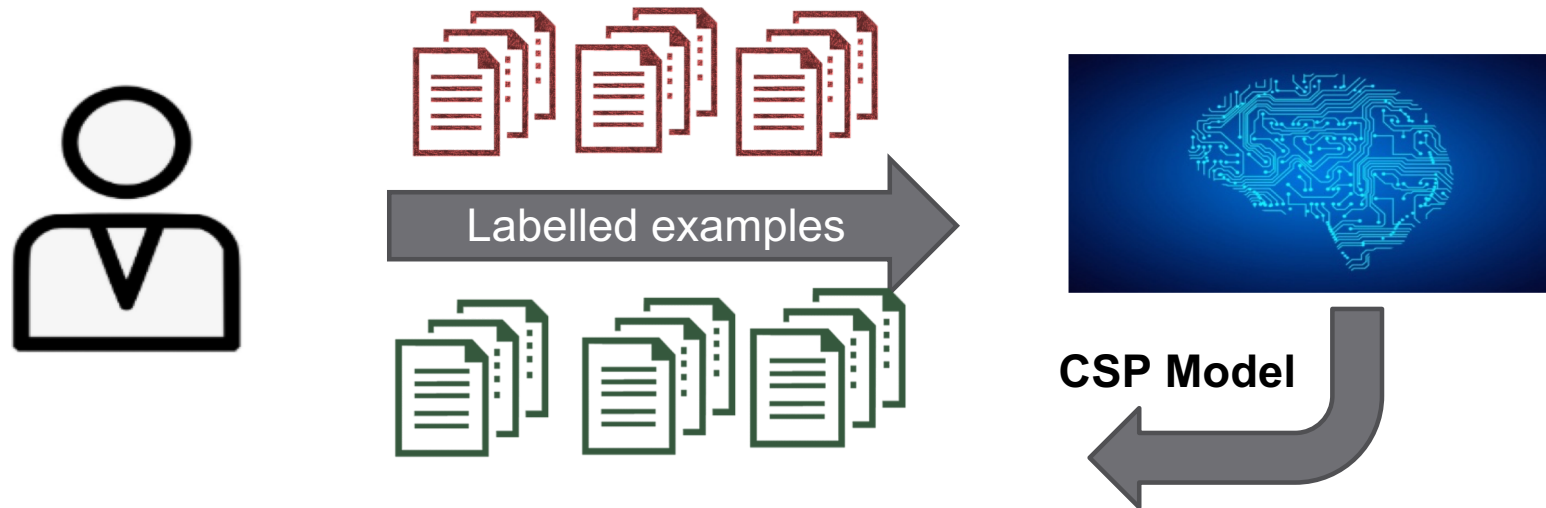
Language Γ contains **numerical** expressions and not constraints
e.g. $x_{i,j} - x_{k,l}$, $x_{i,j} + x_{k,l}$ etc.

Pattern-based
grouping like in
ModelSeeker



Passive Constraint Acquisition

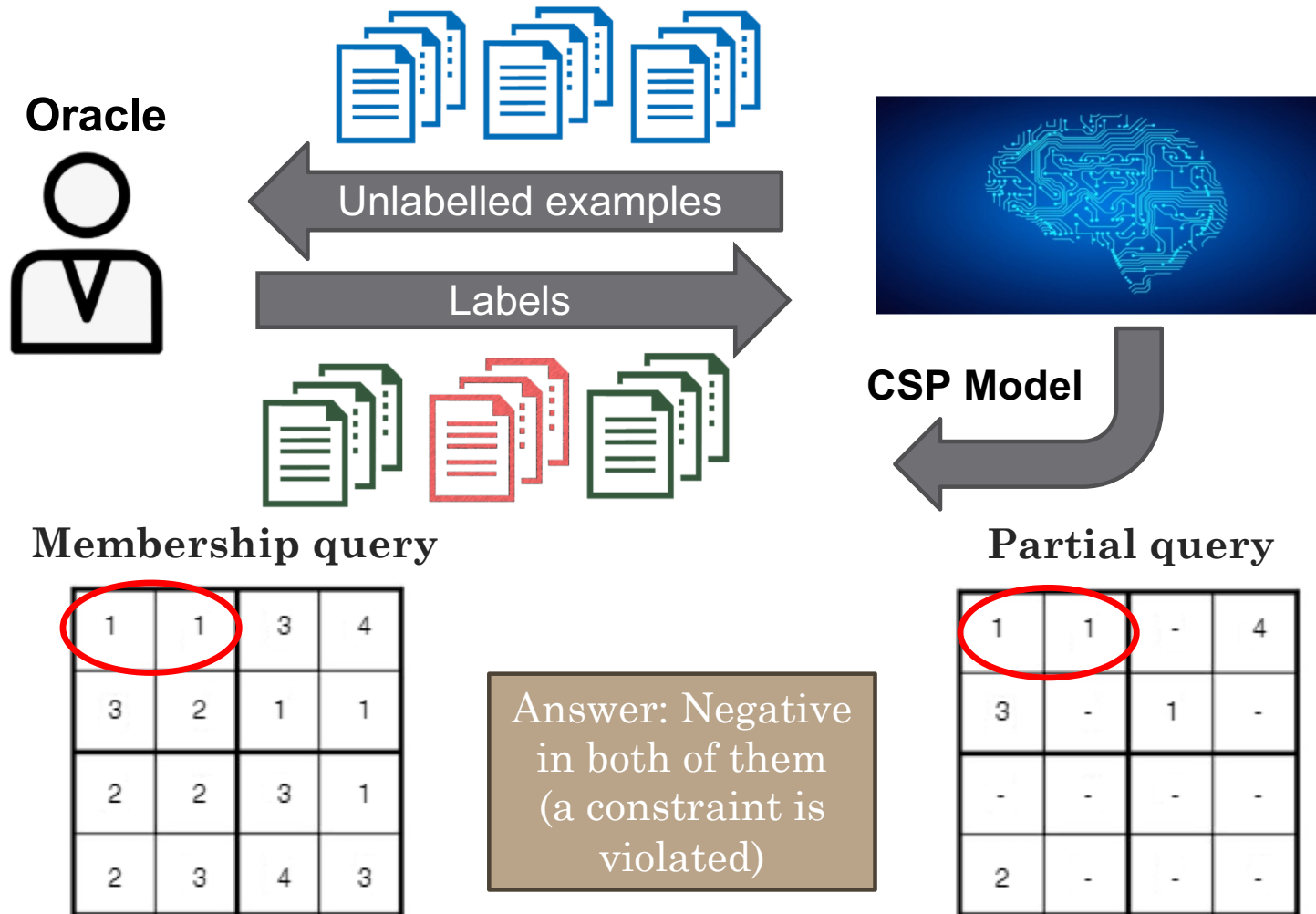
Small Summary



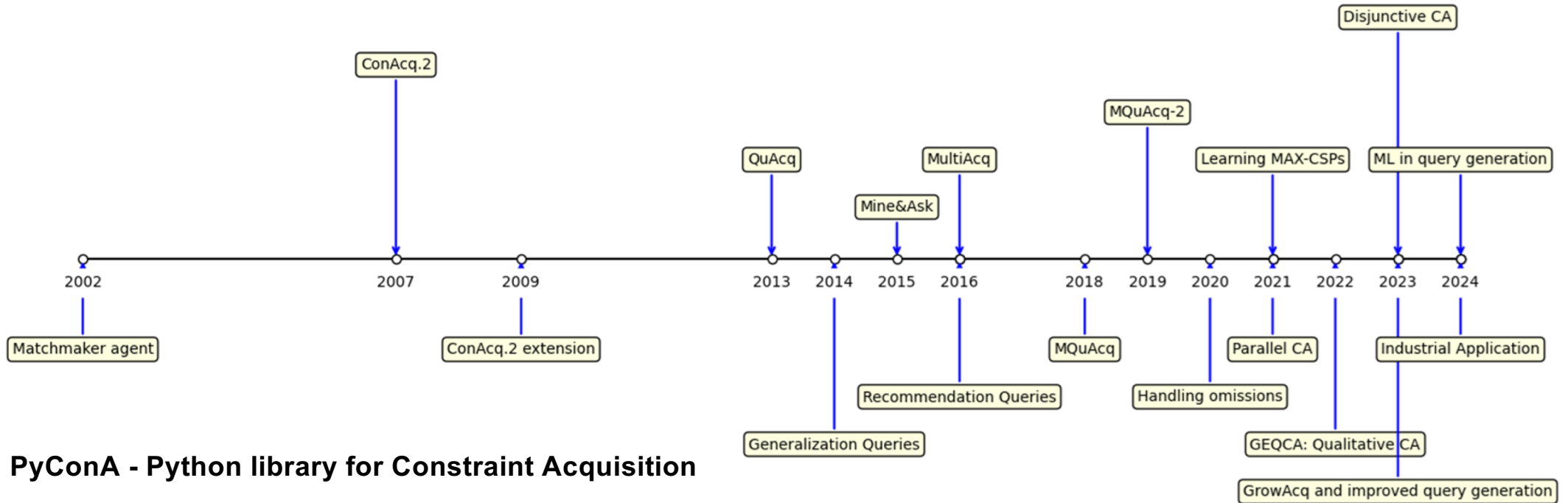
We discussed:

- Basics of Constraint Acquisition
- Passive Constraint Acquisition
- Different approaches tackling different cases

Interactive Constraint Acquisition



Progress in Interactive CA

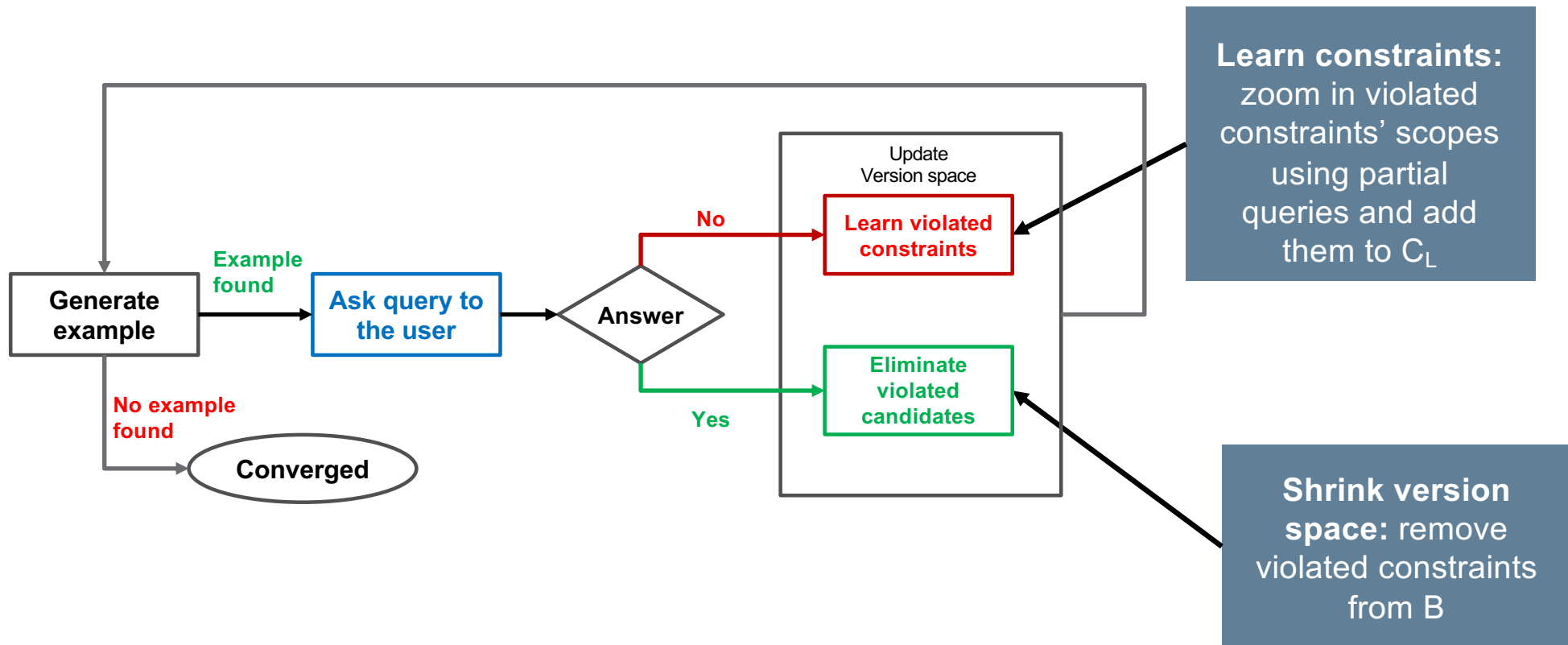


PyConA - Python library for Constraint Acquisition



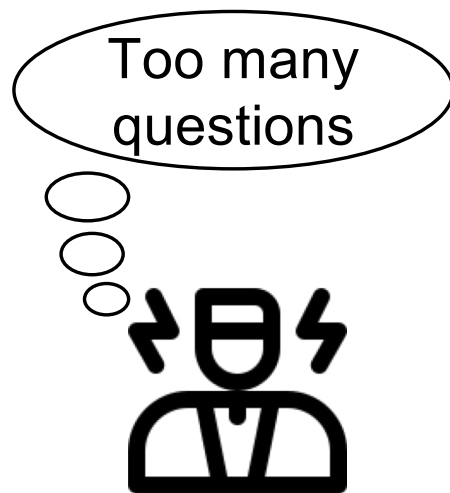
<https://github.com/CPMpy/PyConA>

Interactive Constraint Acquisition

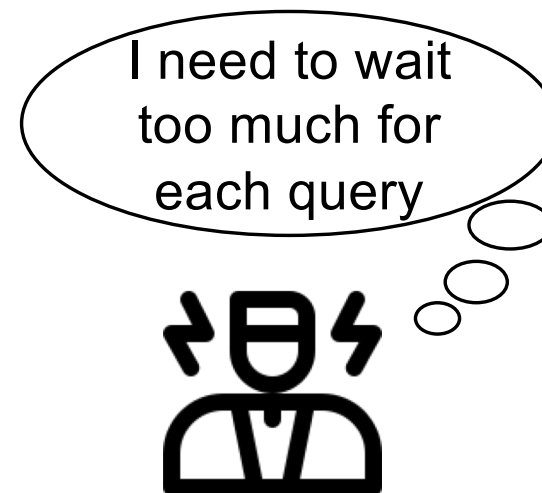


Challenges for interactive CA

Minimum number of queries



Minimum waiting time for the user



Query generation

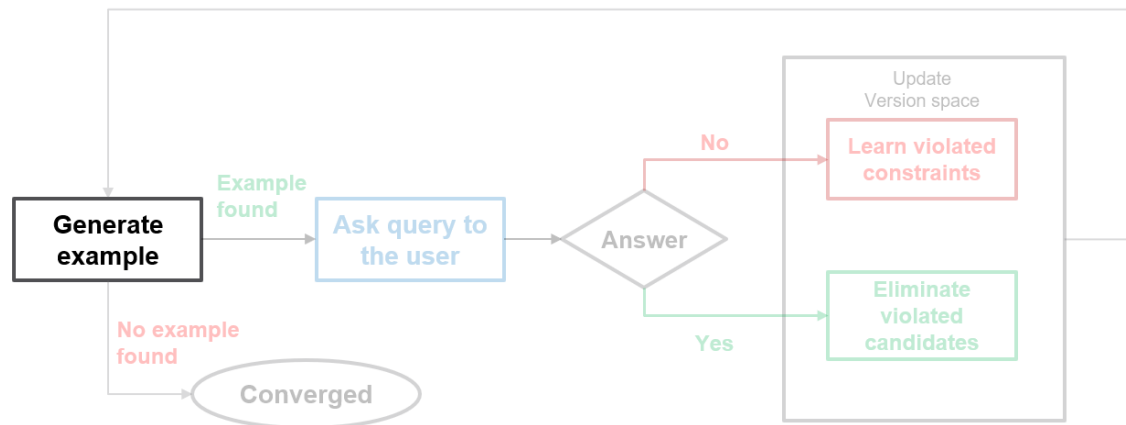
Informative query

- Generate “informative” examples

Quality of query

- Get the maximum amount of information

Convergence



Query generation

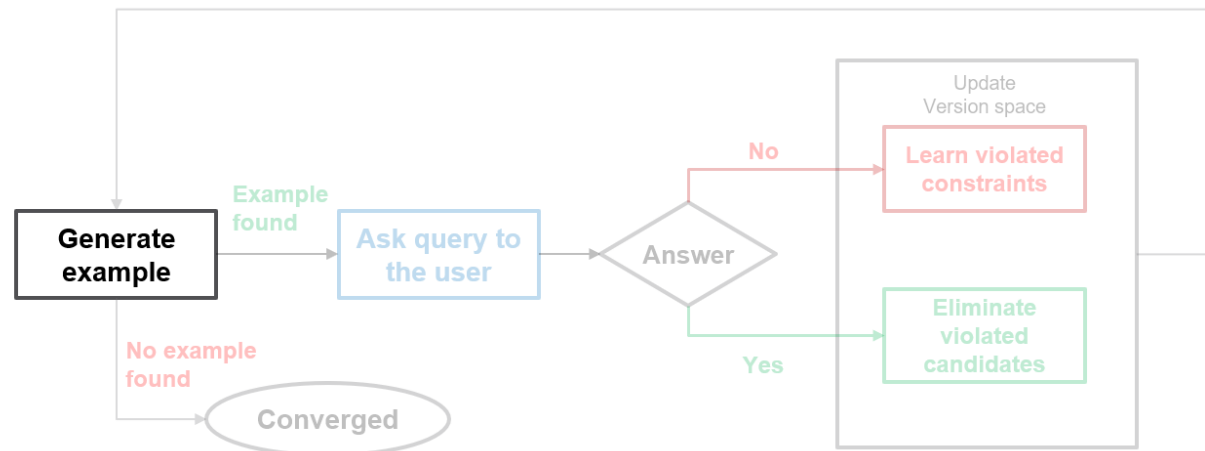
A (partial) membership query is called *Informative* or *irredundant* when the example e to be classified:

- Does not violate any learned constraint in C_L
- Violates at least one constraint from B

Find an Informative (“irredundant”) query

Finding an informative query is a CSP:

$$\text{Find } e \in \text{sol}(C_L \wedge \bigvee_{c \in B} \sim c)$$



Informative and redundant queries

Example: We have a 4x4 Sudoku, which we use a CA system on.

- It has learned all constraints from C_T (i.e., $C_L = C_T$)
- The remaining candidates are $B = \{x_{11} > x_{44}, x_{12} < x_{22}\}$

Redundant!

Violates a constraint from C_L

Query: Is this a solution?

2	2	1	3
1	3	2	4
2	4	3	1
3	1	4	2

Answer: No

Informative!

Satisfies all C_L and violates B

Query: Is this a solution?

4	2	3	1
1	3	4	2
2	4	1	3
3	1	2	4

Answer: Yes

Redundant!

Satisfies all C_L but also satisfies B

Query: Is this a solution?

4	1	2	3
3	2	4	1
2	3	1	4
1	4	3	2

Answer: Yes

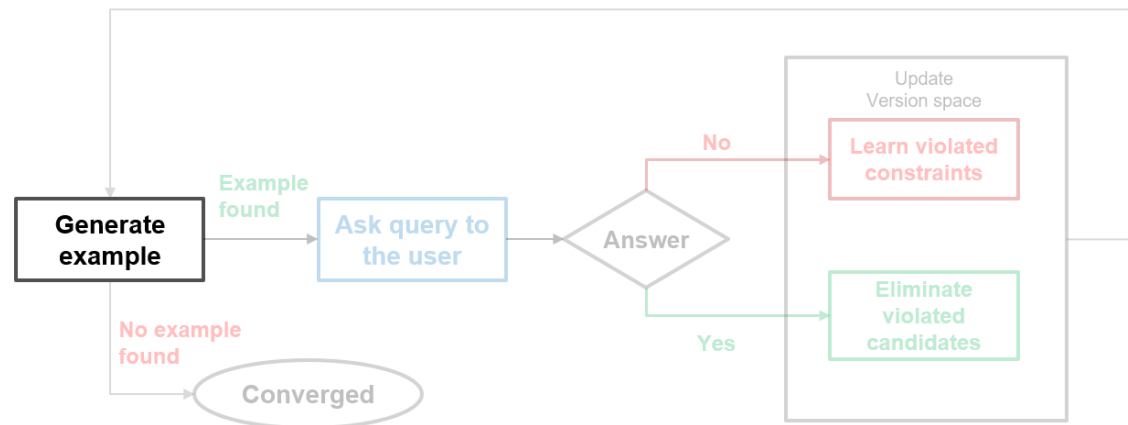
Query generation

Informative query

Quality of query

- Get the maximum amount of information

Convergence



What we can learn from examples

- Learning from *positive* examples (Solutions):

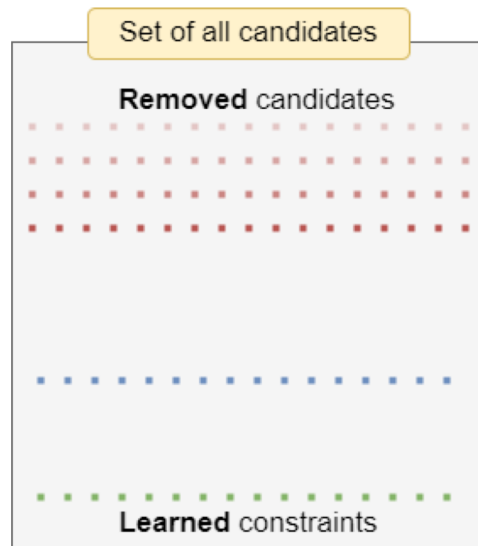
Query: is this a solution?

4	2	1	3
3	1	2	4
2	4	3	1
1	3	4	2

Answer: Yes

- Violated constraints cannot be part of the model
- Otherwise, it could not be a solution

Shrinking the bias



- Learning from *negative* examples (Non-solutions):

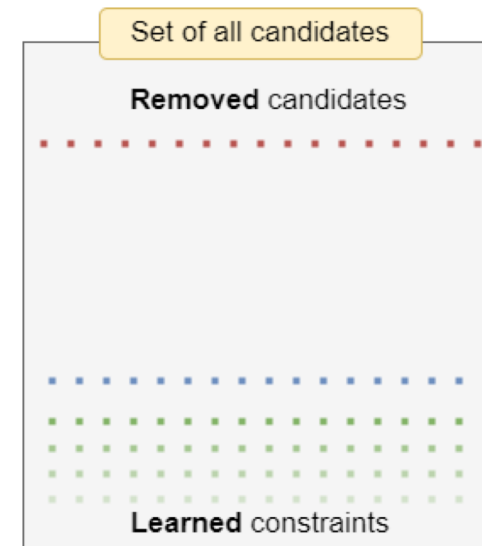
Query: is this a solution?

4	2	3	2
3	1	2	4
1	3	4	2
2	4	1	3

Answer: No

- One (or more) violated constraint is a constraint of the problem
- Otherwise, it would be a solution

Learning Constraints



In both cases, we get information only about the **violated** candidates and not about the satisfied ones!

Query generation

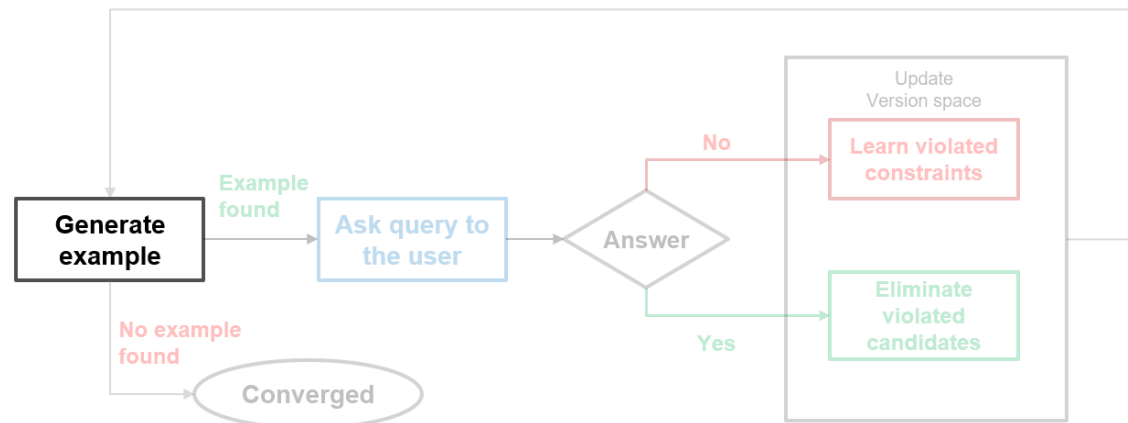
Quality of query

- *Better* generated examples lead to faster convergence
- More information per query -> less queries needed

Typically: maximize candidate violations

$$\max \sum_{c \in B} \sim c$$

Not fully aligning with the goal!!
We will discuss this later



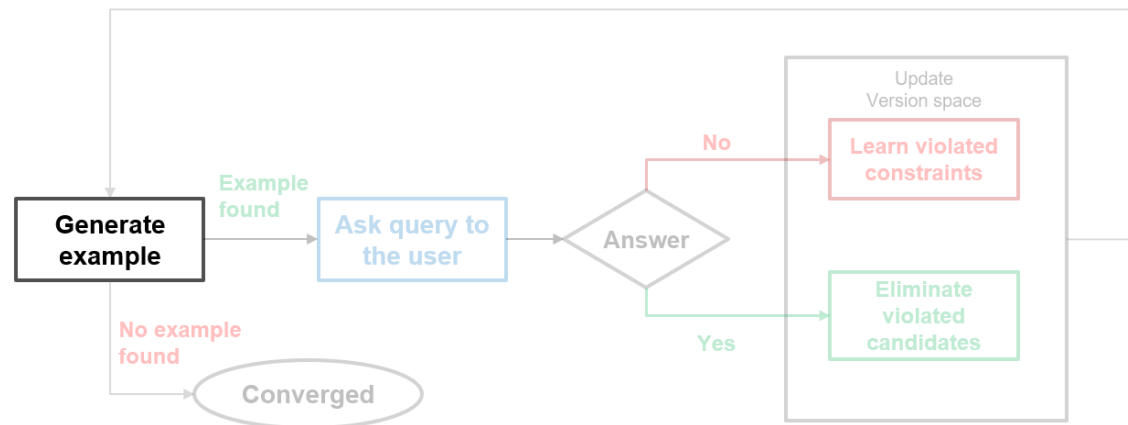
Query generation

Informative query

Quality of query

Convergence

- When no informative query exists
- Premature converge: if query generation failed but not proven UNSAT



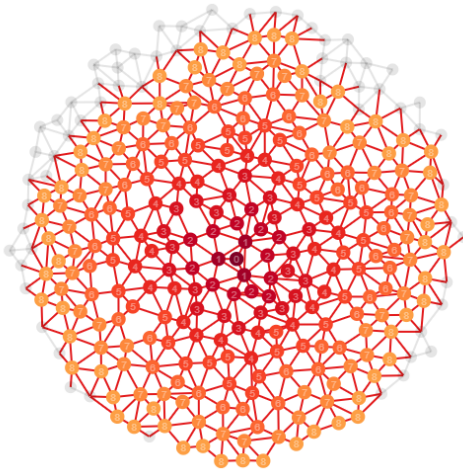
Query generation

Finding an Irredundant query

- Finding an informative query is not always easy...

B can be huge!!

- Can have millions of constraints



B can contain **indirectly** implied constraints:

- indirectly: being implied by combination of constraints from different scopes

Indirect implications are not detected with simple *propagation* from CP solvers used in query generation!!

Query generation

Custom solvers

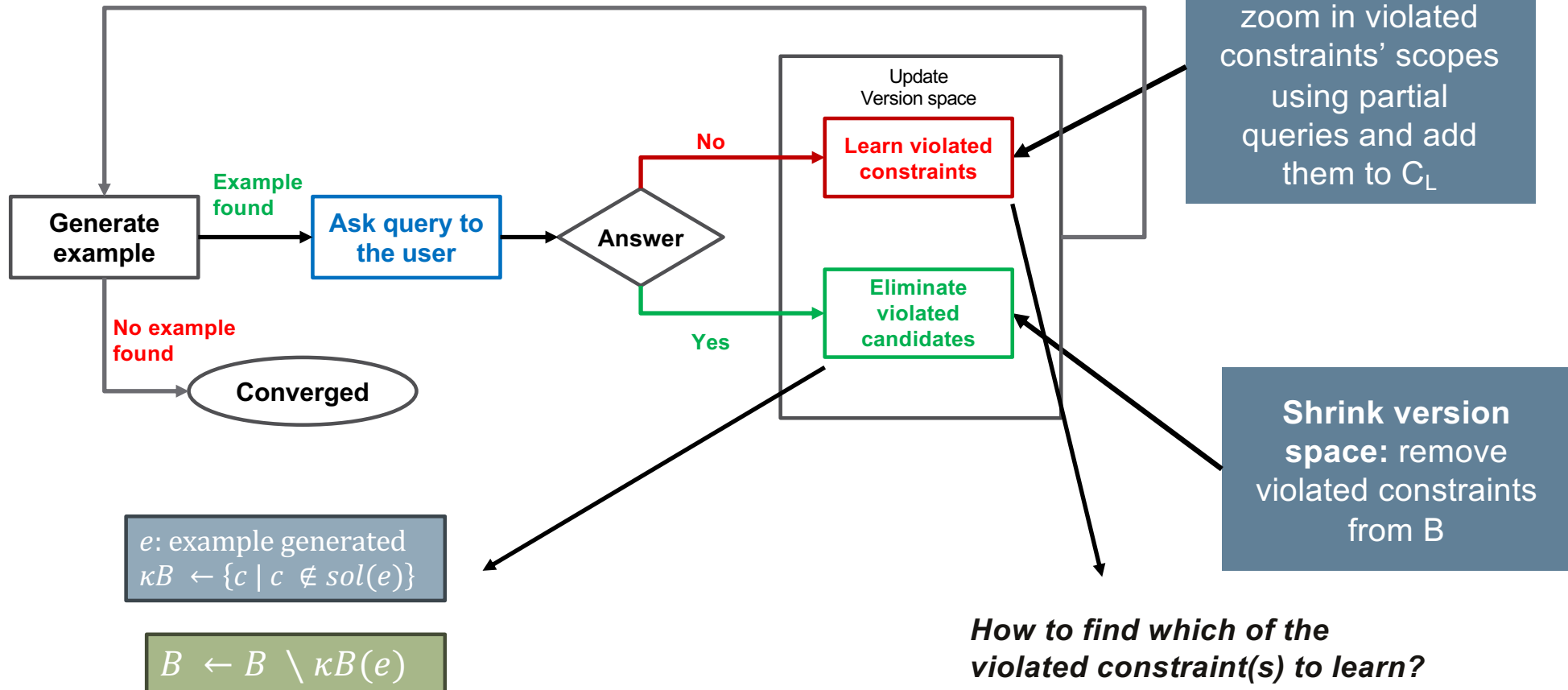
- **Custom solvers** were often employed to deal with this
 - Looking for a **partial** $e_Y \in \text{sol}(C_L[Y] \wedge \bigvee_{c \in B[Y]} \sim c)$
 - Custom branching – **optional variables**
 - Arbitrary $Y \subseteq X$

Projection-based query generation

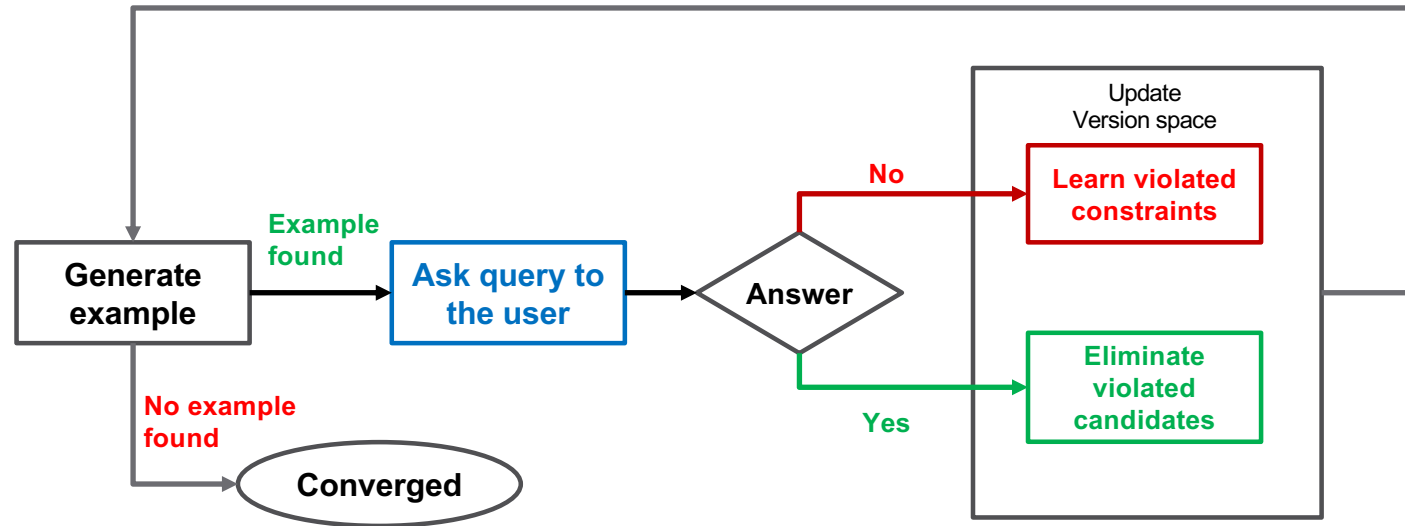
- Project down to the relevant variables $Y = \bigcup_{c \in B} \text{var}(c)$
 - We can only get information for constraints in B
 - Avoiding indirect implications
- Find $e_Y \in \text{sol}(C_L[Y] \wedge \bigvee_{c \in B} \sim c)$

Interactive Constraint Acquisition

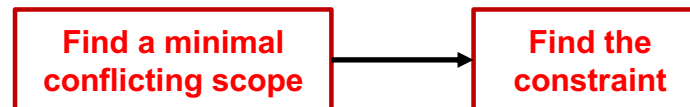
What happens after query generation?



Learning a constraint



➤ 2-step process:



1. **FindScope**: exploit partial (sub)queries to find the problematic part of the assignment
2. **FindC**: Try different assignments to find the specific constraint in the scope

Finding the scope of a constraint

Find a minimal
conflicting scope

Find the
constraint

1	3	4	2
2	1	3	4
4	2	1	3
3	4	2	1

Negative

1	3	4	2
2	1	3	4
-	-	-	-
-	-	-	-

Negative

1	3	-	-
2	1	-	-
-	-	-	-
-	-	-	-

Negative

1	3	-	-
-	-	-	-
-	-	-	-
-	-	-	-

Positive

1	3	-	-
2	-	-	-
-	-	-	-
-	-	-	-

Positive

...

Finding the relation of a constraint

1	3	4	2
2	1	3	4
4	2	1	3
3	4	2	1

We have found the scope: $\{x_{1,1}, x_{2,2}\}$
 Assume that the candidate constraints for this scope are:

$$\{x_{1,1} \neq x_{2,2}, x_{1,1} > x_{2,2}, x_{1,1} < x_{2,2}\}$$

What is the real conflict?

Find a minimal
conflicting scope

Find the
constraint

Try different assignments to find the specific
constraint in the scope

1	-	-	-
-	2	-	-
-	-	-	-
-	-	-	-

Positive

Remove $x_{1,1} > x_{2,2}$

2	-	-	-
-	1	-	-
-	-	-	-
-	-	-	-

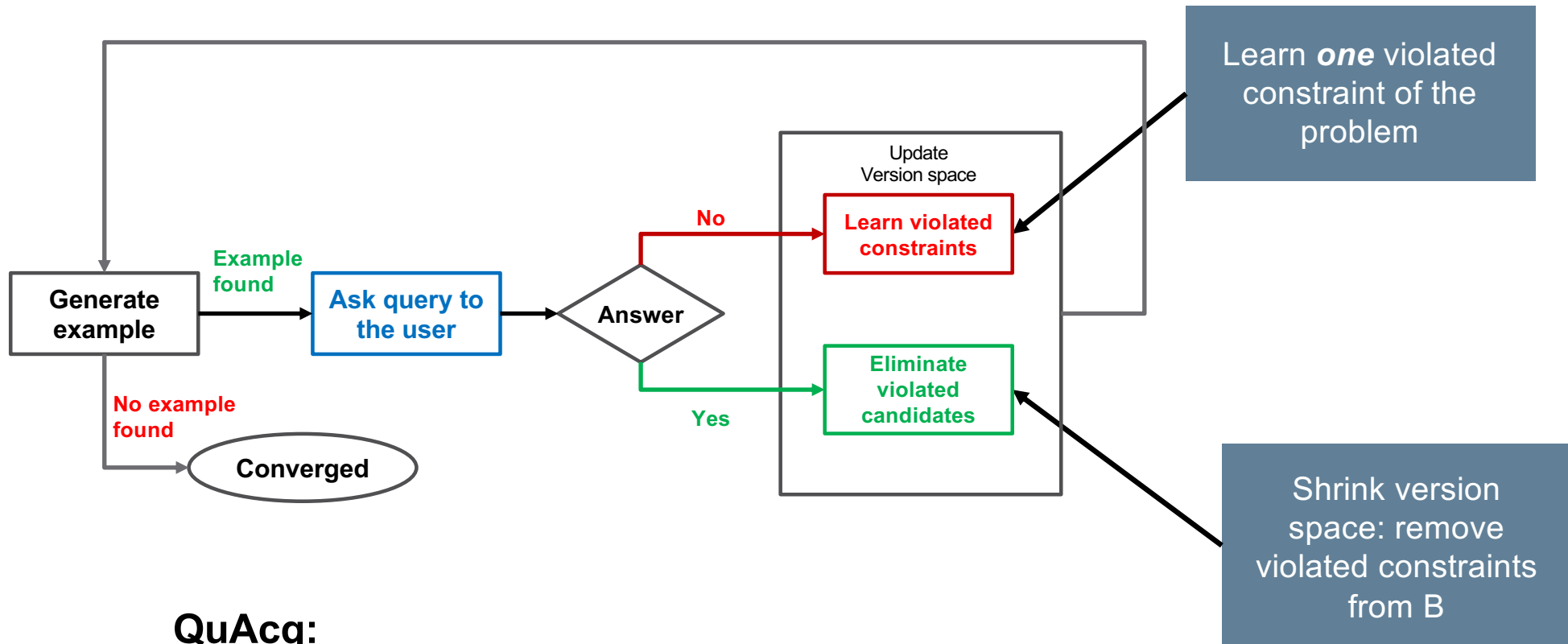
Positive

Remove $x_{1,1} < x_{2,2}$

Only $x_{1,1} \neq x_{2,2}$ left as
candidate: Learn it
 $C_L \leftarrow C_L \cup \{x_{1,1} \neq x_{2,2}\}$

Interactive Constraint Acquisition

QuAcq

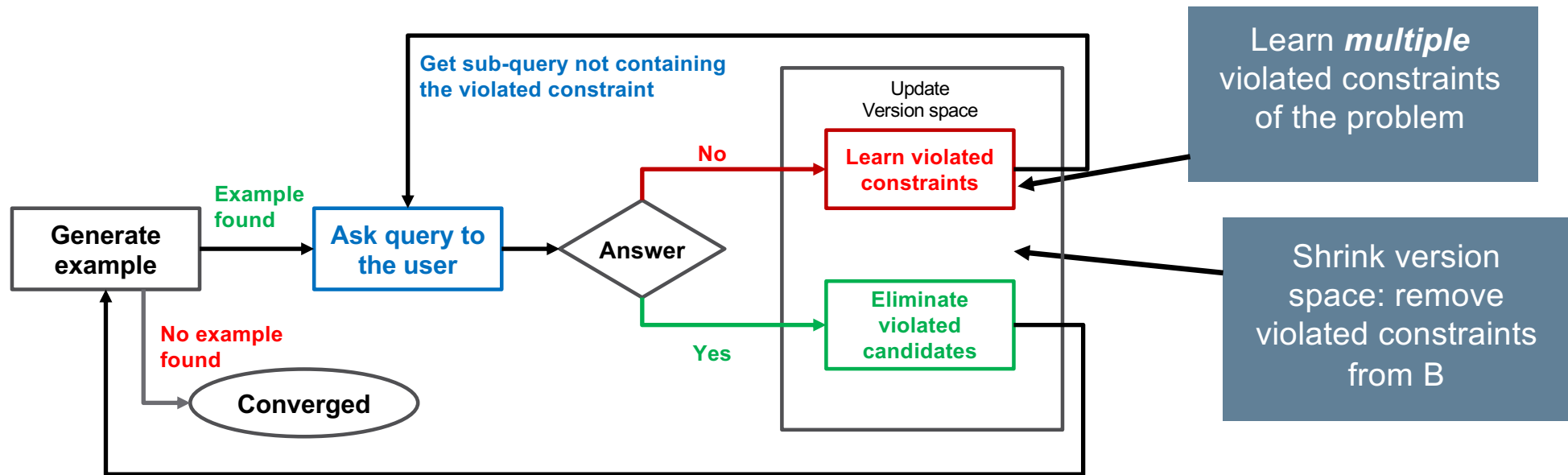


QuAcq:

- Learning one violated constraint per generated example
- Logarithmic number of queries for each constraint

Interactive Constraint Acquisition

Multiple Acquisition



Multiple Acquisition:

- Learn multiple constraints in each loop instance
- Don't generate a new example when a constraint is learnt
 - Instead, get an example in a subset of variables not violating the constraint found

What if the set of candidates is too large??

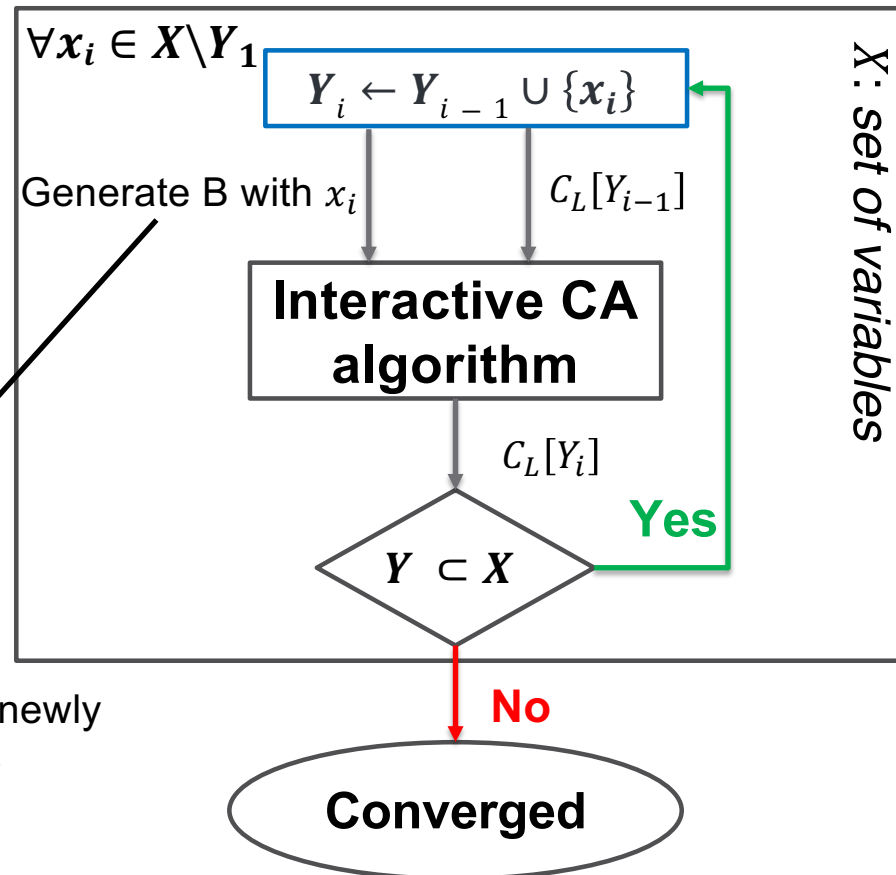
- Can't store all of it at the same time??
- Too slow query generation??

GrowAcq: Growing Acquisition

Start with $Y_1 \leftarrow \emptyset$, or a small subset of X

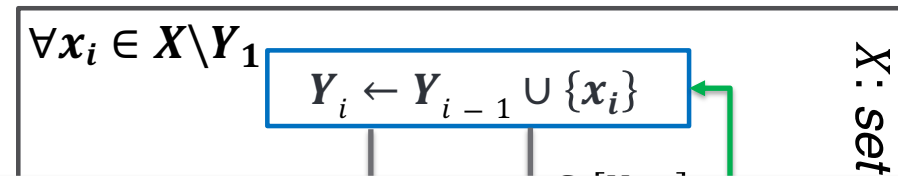
Bottom-up approach: Using Interactive CA algorithms, on an (incrementally growing) subset of variables

Need only constraints that newly added variable participates



GrowAcq: Growing Acquisition

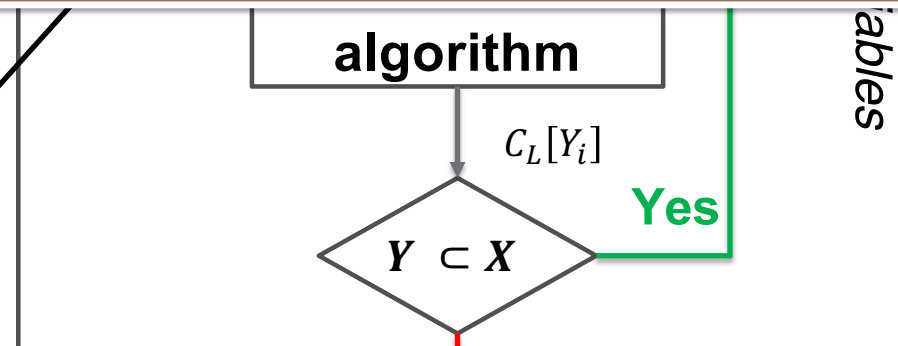
Start with $Y_1 \leftarrow \emptyset$, or a small subset of X



Saves time during query generation! We can use more efficiently the time to guide better Constraint Acquisition

Bot

Interactive CA algorithms, on an (incrementally growing) subset of variables

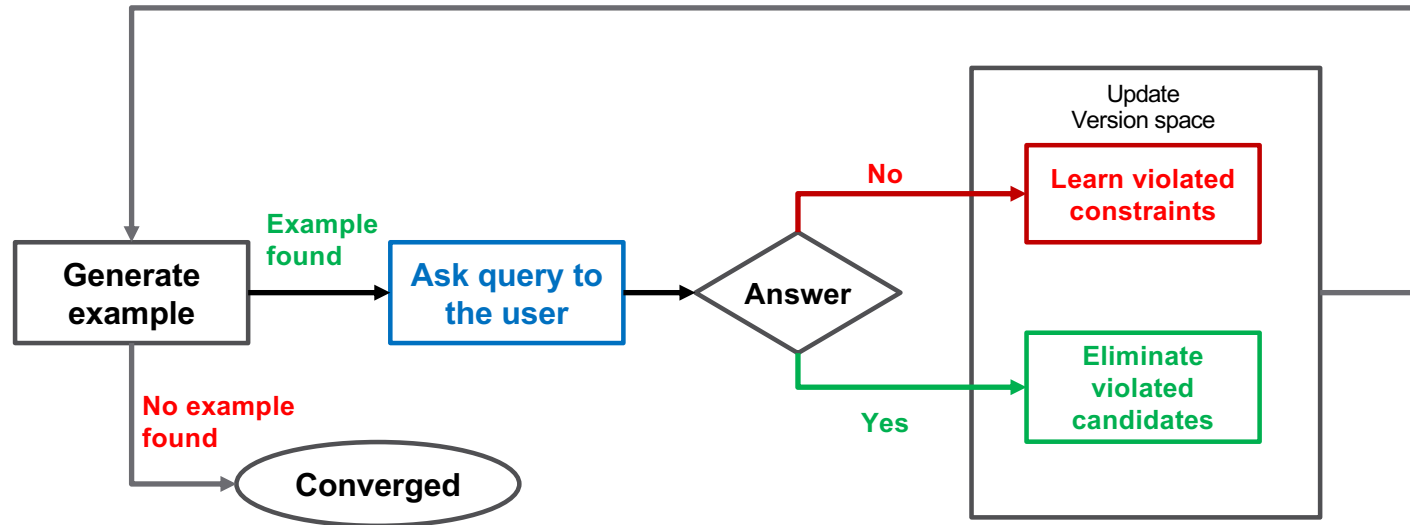


Need only constraints that newly added variable participates

Converged

Interactive Constraint Acquisition

Small summary

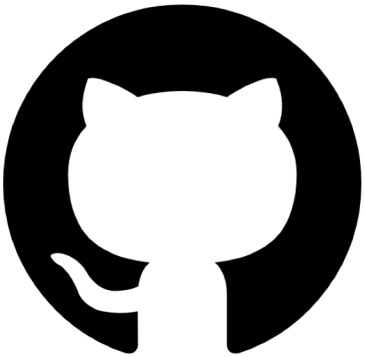


We discussed:

- Interactive CA basics
- Query generation
- Finding the scope of problem constraints
- Finding the relation of the constraints
- Growing Acquisition

KU LEUVEN

PyConA



<https://github.com/CPMpy/PyConA>

pycona 0.2

```
pip install pycona
```



Using the CPMpy modeling library

<https://github.com/CPMpy/CPMpy>

Hands-on on interactive CA Demo: (also in Github: notebooks/Tutorial Demo.ipynb)

<https://colab.research.google.com/github/CPMpy/PyConA/blob/main/notebooks/Tutorial%20Demo.ipynb>

Prediction-Based Interactive Constraint Acquisition

Guiding Query Generation

Quality of query

- *Better* generated examples lead to faster convergence
- More information per query -> less queries needed

Typically: maximize candidate violations

$$\max \sum_{c \in B} \sim c$$

Not fully aligning with the goal!!



Guiding Query Generation

Example: We use a CA system on a simple problem, where we have 4 variables, where all variables need to have a different value

- CA has not learned any constraint yet (i.e., $C_L = \emptyset$).
- The candidates are $B = \{x_i \neq x_j, x_i \leq x_j \mid \forall 0 < i < j < 4\}$

Better generated examples lead to faster convergence

Positive answers: shrink B fast

Violated few candidates:
 $\{x_1 \leq x_4, x_2 \leq x_4, x_3 \leq x_4\}$

Query: Is this a solution?

2	3	4	1
---	---	---	---

Answer: Yes

Violated many candidates:
 $\{x_1 \leq x_4, x_2 \leq x_4, x_3 \leq x_4, x_1 \leq x_3, x_2 \leq x_3, x_1 \leq x_2\}$

Query: Is this a solution?

4	3	2	1
---	---	---	---

Answer: Yes



- Negative answers: Find the conflict fast

Violated few candidates:
 $\{x_2 \neq x_3\}$

Query: Is this a solution?

1	2	2	3
---	---	---	---

Answer: No



With the first query, CA directly learns $x_2 \neq x_3$!

Violated many candidates:
 $\{x_1 \neq x_4, x_2 \neq x_4, x_3 \neq x_4, x_1 \neq x_3, x_2 \neq x_3, x_1 \neq x_2\}$

Query: Is this a solution?

1	1	1	1
---	---	---	---

Answer: No

The more we have violated the faster B will shrink

The less candidates we have violated, the less queries we need to find the constraint(s)

Guiding Query Generation

Better generated examples lead to faster convergence

Positive answers: shrink B fast



The more we have violated the faster B will shrink



$$\max \sum_{c \in B} \llbracket e \notin \text{sol}(c) \rrbracket$$

Opposite objectives
based on the (future)
answer

- Negative answers: Find the conflict fast



The less candidates we have violated, the less queries we need to find the constraint(s)



$$\min \sum_{c \in B} \llbracket e \notin \text{sol}(c) \rrbracket$$

Guiding Query Generation

Better generated examples lead to faster convergence

Positive answers: shrink B fast

- Negative answers: Find the conflict fast

We cannot know the answer of the user before we ask the query $\rightarrow$ max violations

The more we have violated the
fa

The less candidates we have violated, the less

But what if we can predict if a candidate is a constraint of the problem or not?

$$\max \sum_{c \in B} \llbracket e \notin \text{sol}(c) \rrbracket$$

Opposite objectives
based on the (future)
answer

$$\min \sum_{c \in B} \llbracket e \notin \text{sol}(c) \rrbracket$$

Guiding Query Generation

- Positive answers: shrink B fast $\rightarrow \max \sum_{c \in B} \llbracket e \notin \text{sol}(c) \rrbracket$
- Negative answers: Find the conflict fast $\rightarrow \min \sum_{c \in B} \llbracket e \notin \text{sol}(c) \rrbracket$

What if we can predict if a candidate is a constraint of the problem or not?

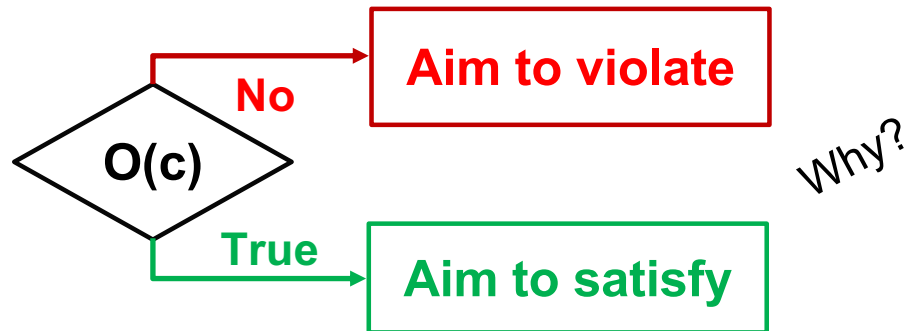
If the constraint c is **violated**

Increase objective value by 1

If it is a constraint predicted to be true: reduce objective value significantly

Use of Oracle $O(c) = (c \in C_T)$, to guide query generation based on the prediction of the constraint

$$e = \operatorname{argmax}_{e \in \text{Sol}(C_L \wedge B)} \sum_{c \in B} \llbracket e \notin \text{sol}(c) \rrbracket \cdot (1 - |T| \cdot \llbracket O(c) \rrbracket)$$



1. Aim for positive answers first:
 $\max(\sum_{c \in B} \llbracket e \notin \text{sol}(c) \rrbracket)$
2. When a (probably true) constraint has to be violated, leading to a negative answer
 $\min(\sum_{c \in B} \llbracket e \notin \text{sol}(c) \rrbracket)$

Guiding CA when finding the scope

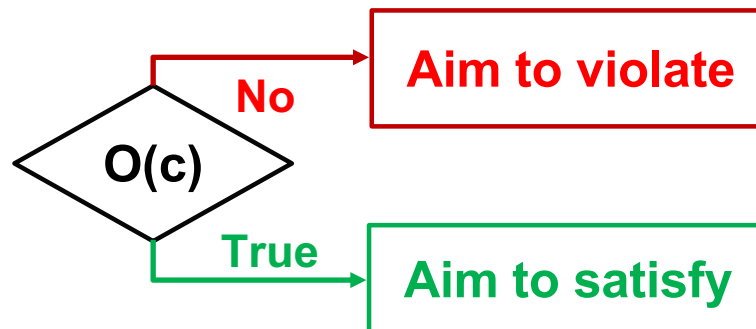
Exploit partial (sub)queries to find the conflicting part

How are the removed variable assignments decided???

Follow the same logic

- But the new example is a sub-example of previous one
- Instead of deciding variable assignments, decide which variables to keep in the assignment

$$e_Y = \operatorname{argmax}_{e_Y \mid Y \subseteq \operatorname{var}(e)} \sum_{c \in B} \llbracket e_Y \notin \operatorname{sol}(c) \rrbracket \cdot (1 - |I| \cdot \llbracket O(c) \rrbracket)$$



Guiding CA when finding the relation

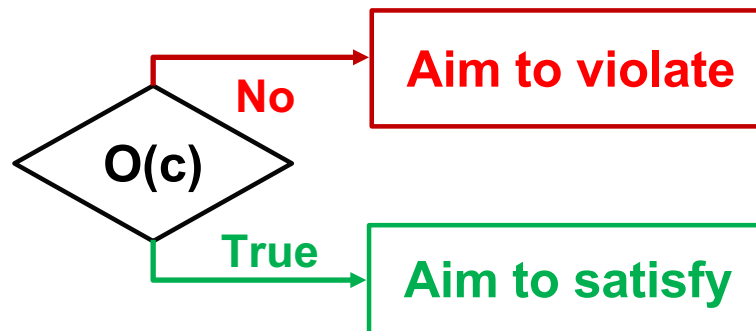
Try different assignments to find the specific constraint in the scope

How are the assignments decided???

Follow the same logic

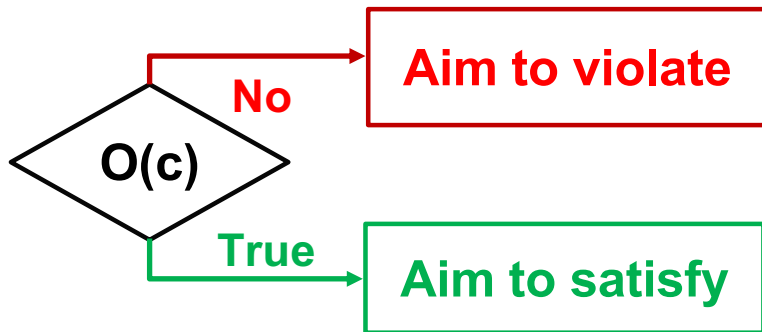
- But the new example has to be an assignment only to the scope S found

$$e_S = \operatorname{argmax}_{e_S \in \operatorname{Sol}(C_L[S] \wedge B[S])} \sum_{c \in B} \llbracket e_S \notin \operatorname{sol}(c) \rrbracket \cdot (1 - |\Gamma| \cdot \llbracket O(c) \rrbracket)$$



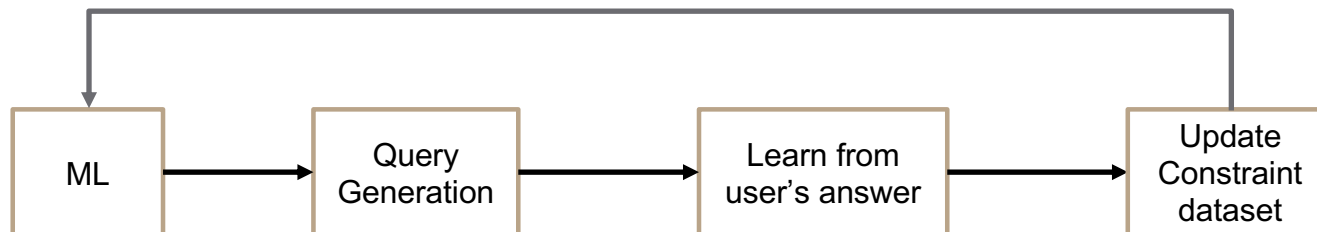
Do we have an oracle $O(c)$ to guide CA

The oracle $O(c)$ “classifies” a candidate as a problem constraint or not



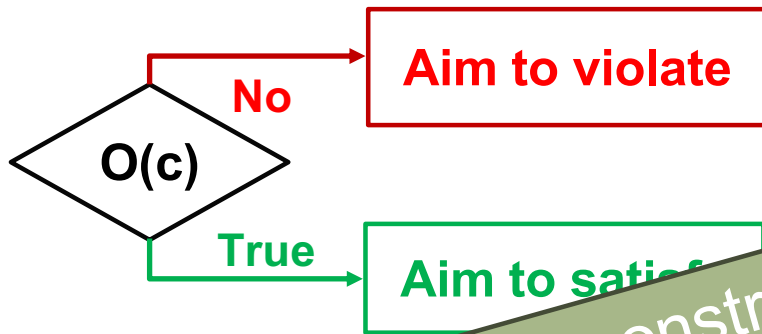
It is a prediction problem

Use Machine Learning!!



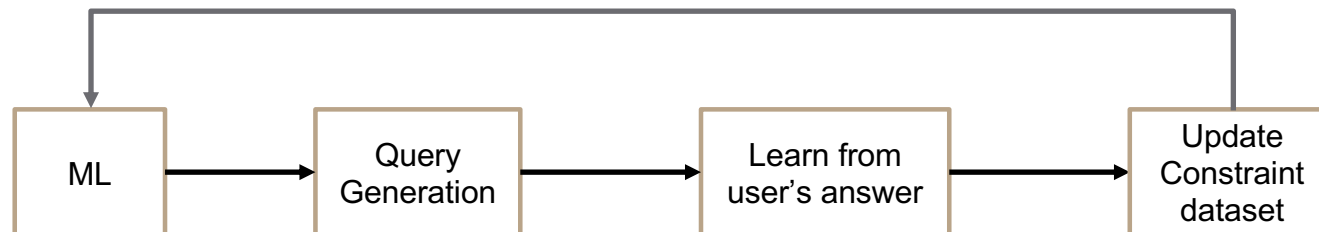
Do we have an oracle $O(c)$ to guide CA

The oracle $O(c)$ “classifies” a candidate as a problem constraint or not



Predictions for constraints not for variable assignments!!

Use Machine Learning!!



Using Machine Learning for the prediction

Dataset: Constraint features and class (True or False)

- Constructing during the acquisition process
- Constraints that we know are part of the problem or not
- When a constraint is learned add a positive instance
- When a constraint is removed from B add a *negative instance*
- Use both relation and scope features

Relation-based features	Scope-based features
Relation name (string)	Dim[i] same_val (Bool)
Has constant (Bool)	Dim[i] avg (float)
Constant value (int)	Dim[i] distance (int)
Arity (int)	...

Using Machine Learning for the prediction

Dataset: Constraint features and class (True or False)

- Constructing during the acquisition process
- Constraints that we know are part of the problem or not
- When a constraint is learned add a positive instance
- When a constraint is removed from B add a *negative instance*
- Use both relation and scope features

Example for constraint
 $x_{1,1} \neq x_{1,2}$ in Sudoku

Relation-based features		Scope-based features		For the 2 dimensions
Relation name (string)	$\neq$	Dim[i] same_val (Bool)	True, False	
Has constant (Bool)	False	Dim[i] avg (float)	1, 1.5	
Constant value (int)	-1	Dim[i] distance (int)	0, 1	
Arity (int)	2	...		

Using Machine Learning in the Oracle

The oracle $O(c)$ “classifies” a candidate as a problem constraint or not

Use of any classification technique to simulate the Oracle

$$O(c) = \text{Class}(c) \longleftarrow \text{classifier.predict()}$$

Use of probabilities?

$$O(c) = \frac{1}{\text{expected_queries}} \leq P(c) \longleftarrow \text{classifier.predict_proba()}$$

Minimize the expected number of queries

expected_queries : estimation of number of queries for each constraint when not guided

Using Machine Learning in the Oracle

The oracle $O(c)$ “classifies” a candidate as a problem constraint or not

Use of any classification technique to simulate the Oracle

$$O(c) = \text{Class}(c) \leftarrow \text{classifier.predict()}$$

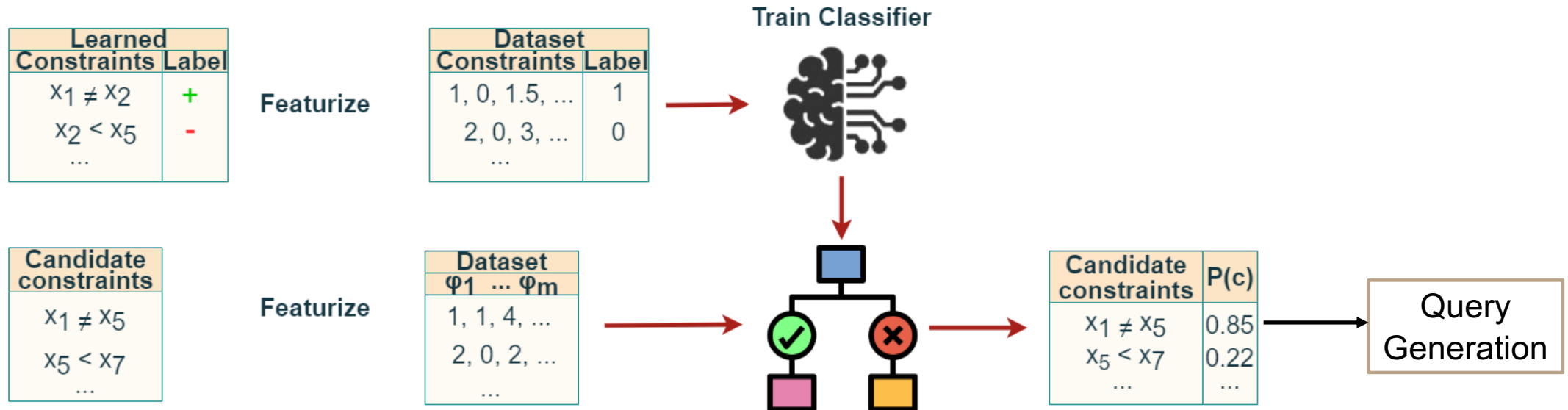
Use of probabilities?

$$O(c) = \frac{1}{\text{expected_queries}} \leq P(c) \leftarrow \text{classifier.predict_proba()}$$

- Using predicted probabilities instead of class works way better!!
- Assuming that a candidate is a constraint of the problem even if the probability is less than 50% (but above the threshold)
- Threshold defined to minimize number of queries!

Minimize the expected number of queries
 expected_queries : estimation of number of queries for each constraint when not guided

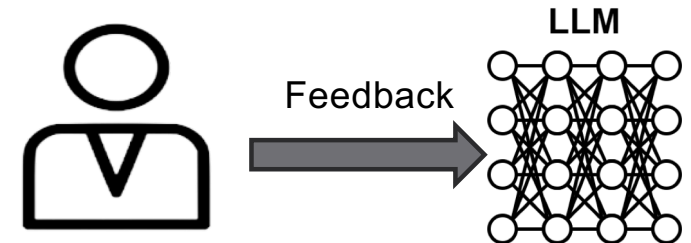
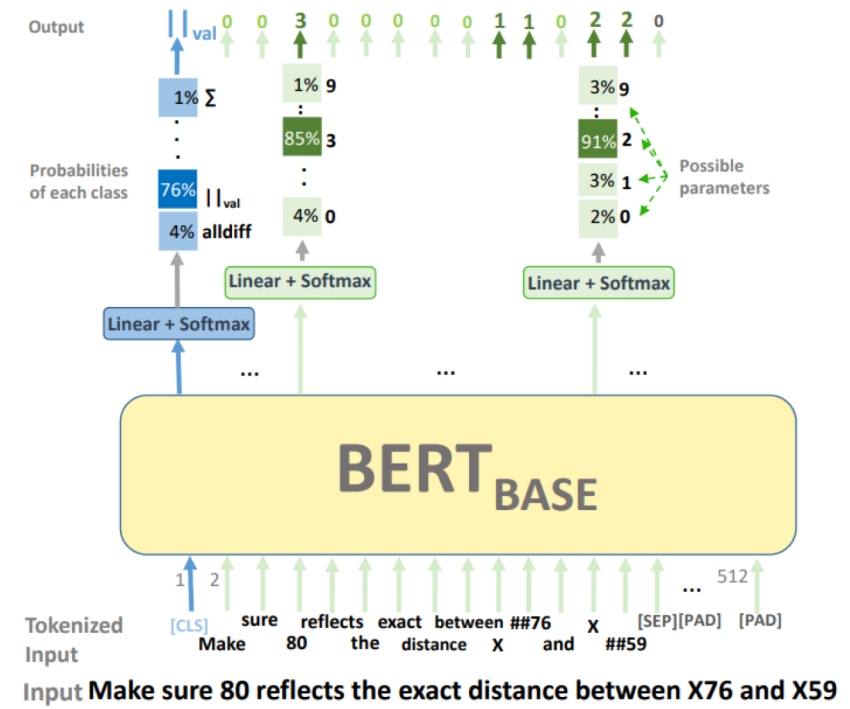
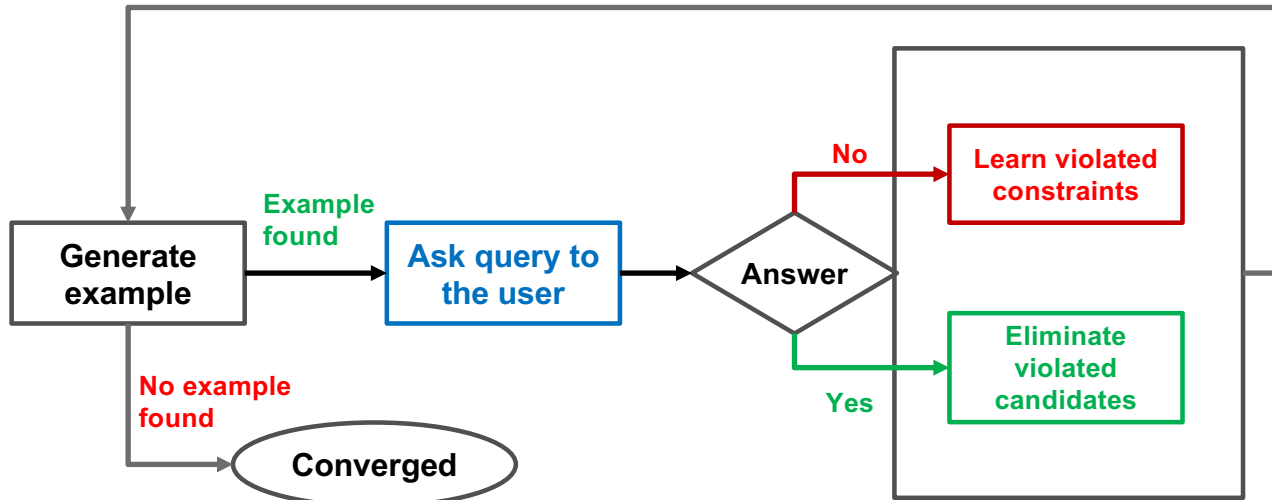
Learning to Learn in Interactive CA



Statistical ML learns the structure implicitly, query-based learning makes it explicit

Exploiting LLMs to process user feedback

- Accept user feedback after negative answers,
- fine-tuned LLMs process the feedback
- No candidate constraints



Open challenges in Interactive CA

Challenges

Number of queries

- Number of queries needed to converge is still large.
- Use also more expressive types of queries
- Allow more information to be given by the user
- Generalize during learning!



Challenges

Specific classes of constraints

- **Global constraints:** Exploding the set of candidate constraints
- **Linear inequalities** with constants: Need to consider all possible constants -> Exploding the set of candidate constraints

COUNT

$$x_1 + 5 < x_2$$

ALLDIFFERENT

$$|x_1 + 12| > x_4$$

CUMULATIVE

SUM

$$x_1 - x_2 \neq 238$$

CIRCUIT

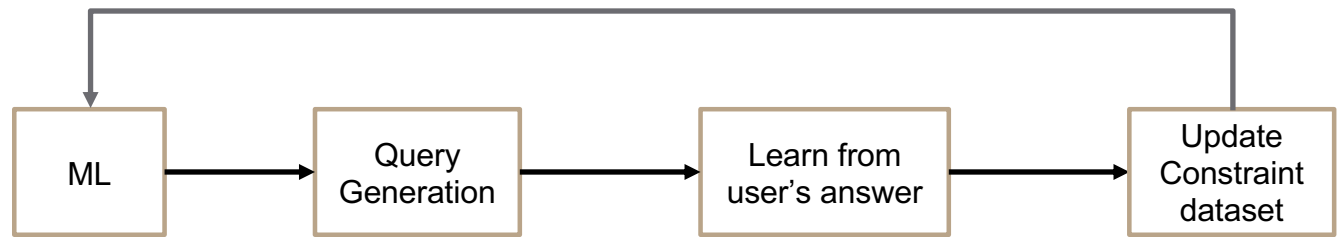
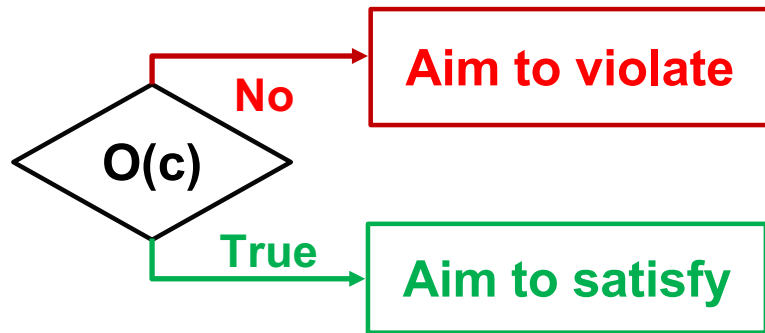
Challenges

Noisy data

- Unlike in machine learning, most constraint acquisition techniques still assume the user always (knows how to) answer correctly
- Tighter integration with modern machine learning techniques



Conclusions



- Significant **advancements** in interactive CA the last years.
- **Statistical ML** can be successfully be used to capture patterns in constraints!
- But still much **more are needed**.
- **PyConA** – Python library for Constraints Acquisition: Goal to promote research in CA