Constraint Acquisition

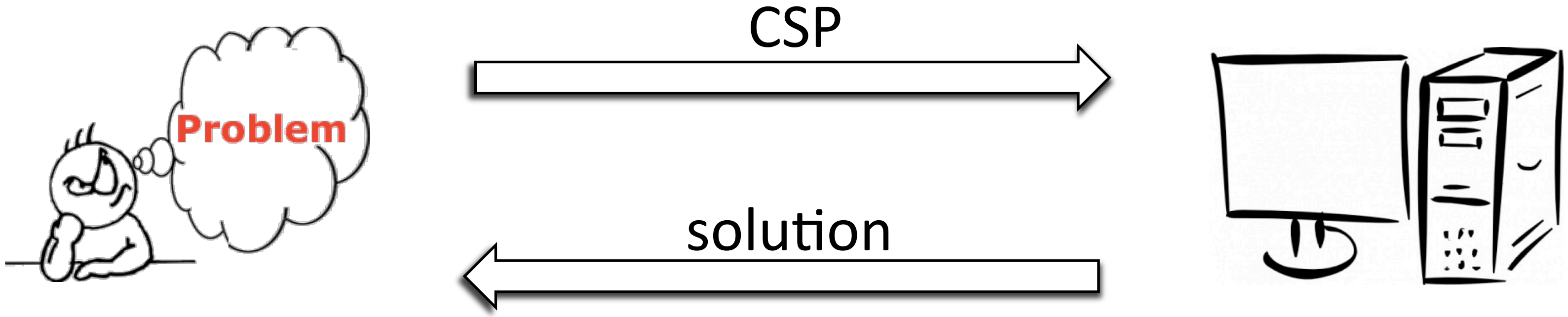
Nadjib Lazaar

U. Montpellier, France
LIRMM - COCONUT team

24-11-17
CAVIAR - Jussieu

lazaar@lirmm.fr

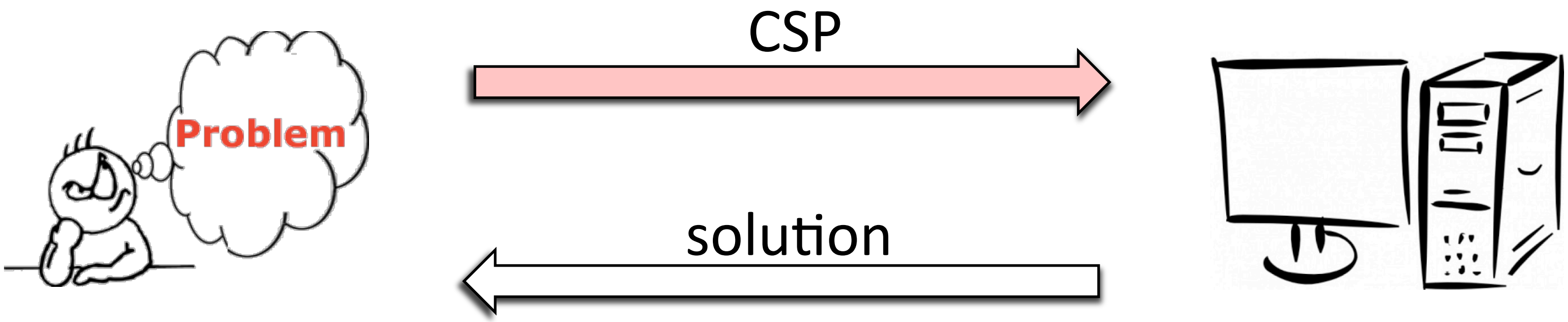
Motivations



CSP

solution

Motivations

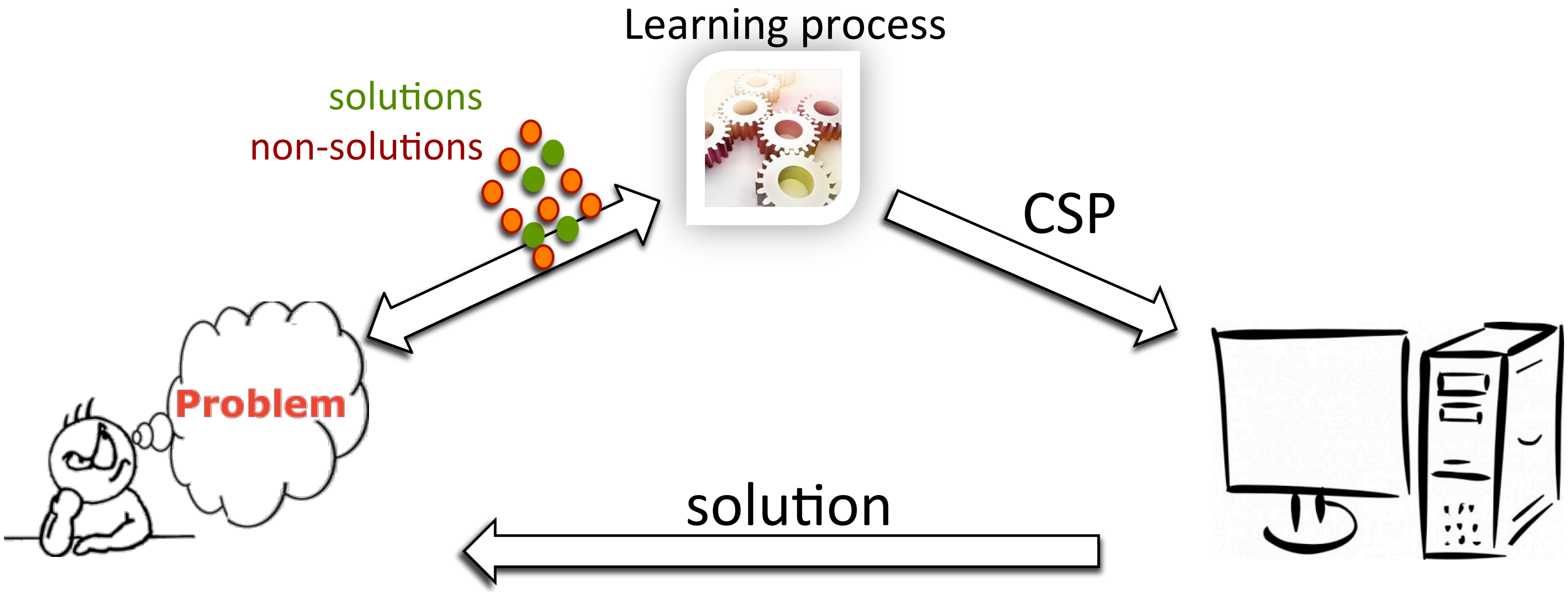


- Question: How does the user write down the constraints of a problem?
- Limitations: modelling constraint networks require a fair expertise

[Freuder99, Frisch et al.05, Smith06]

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

Motivations

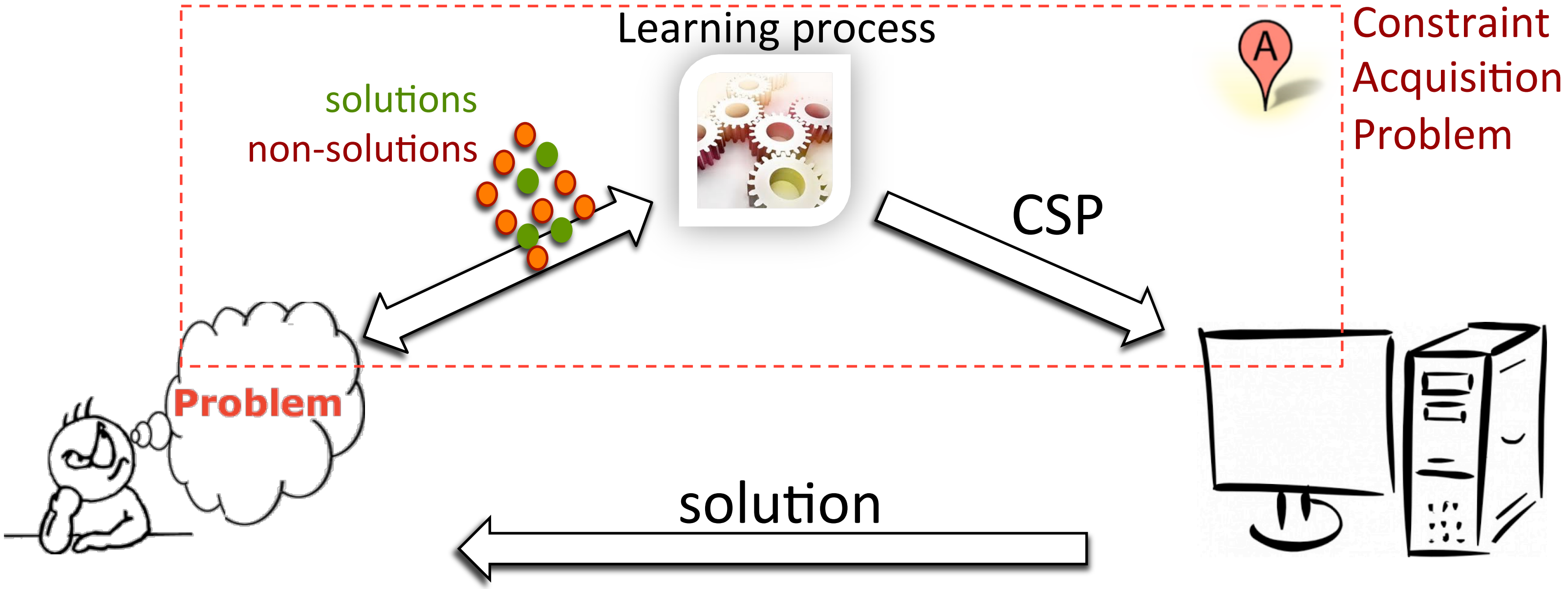


- Question: How does the user write down the constraints of a problem?
- Limitations: modelling constraint networks require a fair expertise

[Freuder99, Frisch et al.05, Smith06]

- Need: Simple way to build constraint model ➔ Modeller-assistant
- How: In a Machine Learning way (passive/active, offline/online, by reinforcement…)

Motivations



- 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]

- Need: Simple way to build constraint model ➔ Modeller-assistant
- How: In a Machine Learning way (passive/active, offline/online, by reinforcement…)

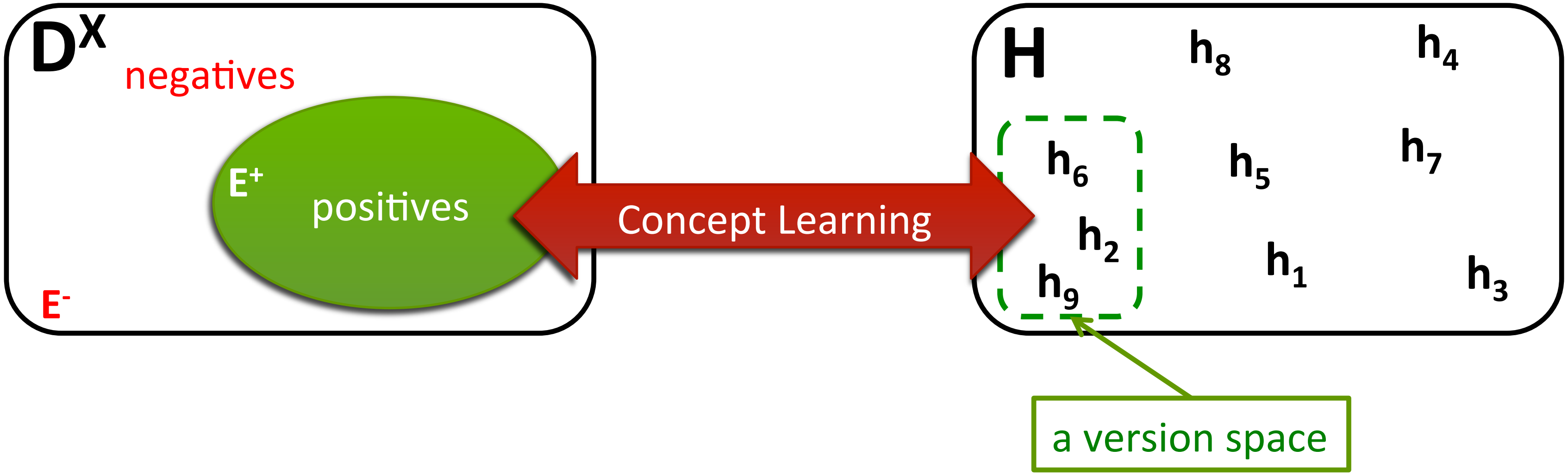
Version Space Learning (Overview) [Mitchell82]

- Let X=x_1,..,x_n a set of attributes of domains D=D_1,..,D_n
- A concept is a Boolean function $f : X \rightarrow \{0, 1\}$
  - *f(xi)=0* => *xi* is a negative instance
  - *f(xj)=1* => *xj* is a positive instance

Given a set of hypothesis **H**, any subset of **H** represents **a version space**

- A concept **to learn** is the set of **positive instances** that can be represented by **a version space**

Version Space Learning (Overview) [Mitchell82]

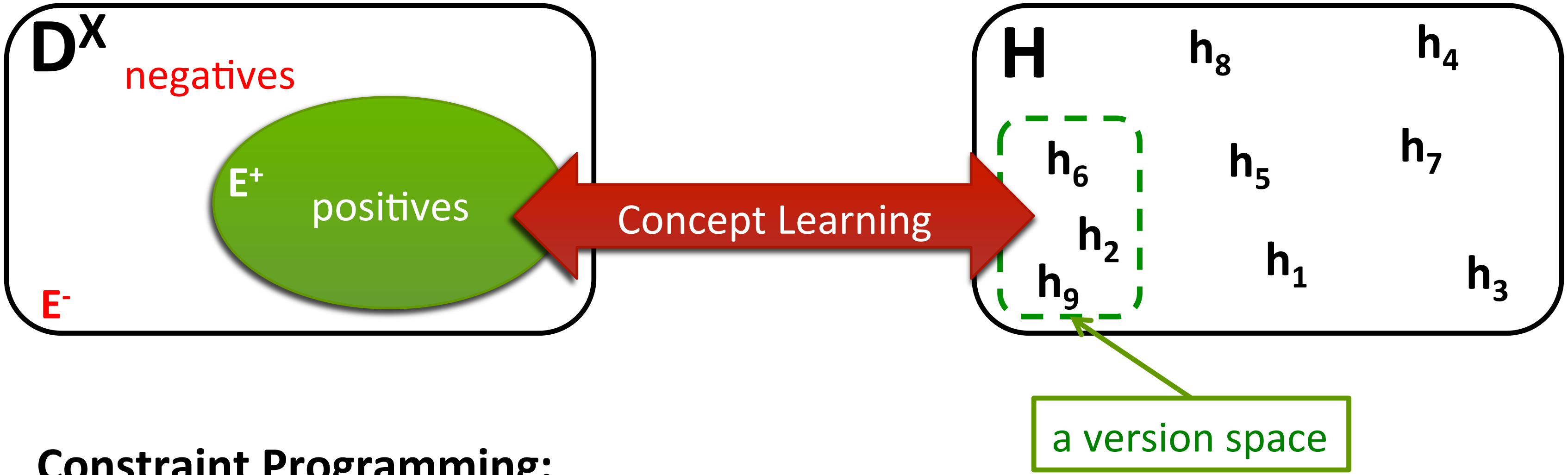


- Concept to learn:

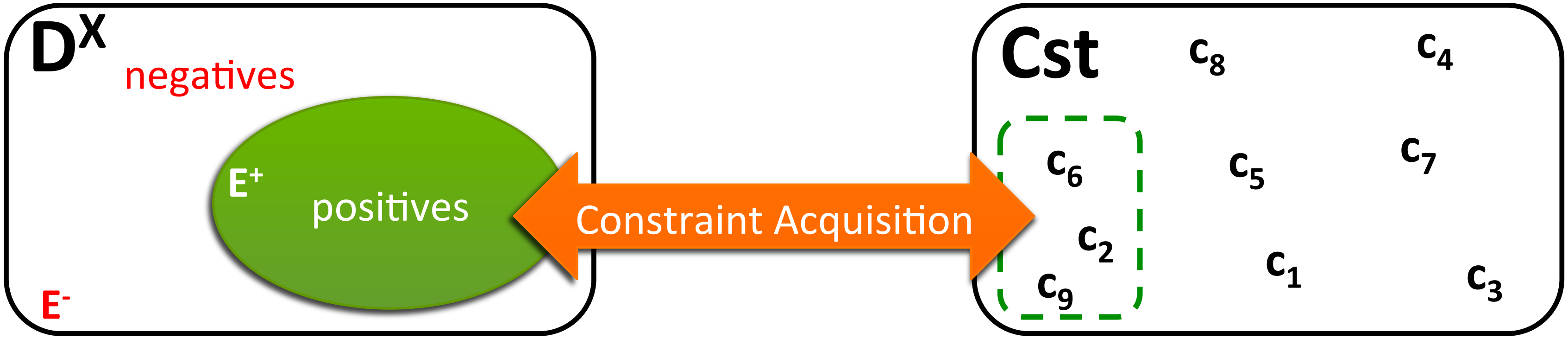
$$f: \quad (\forall x_i \in E^+ : f(x_i) = 1) \wedge (\forall x_i \in E^- : f(x_i) = 0)$$

$$\boxed{f \equiv h_2 \wedge h_6 \wedge h_9}$$

Constraint Acquisition as Version Space Learning

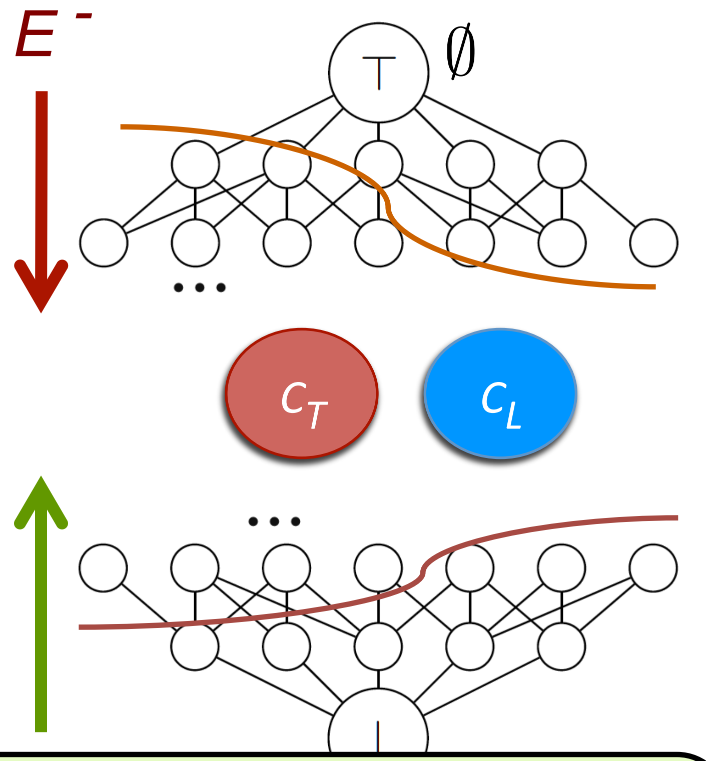


Constraint Programming:



Constraint Acquisition Problem

- Inputs:
  - (X,D): Vocabulary
  - Γ: Constraint language
    - ➔ B: Bias (constraints/hypothesis)
  - C_T: Target Network (concept to learn)
  - (E^+,E^-): training set
- Output:
- C_L: Learned network such that:



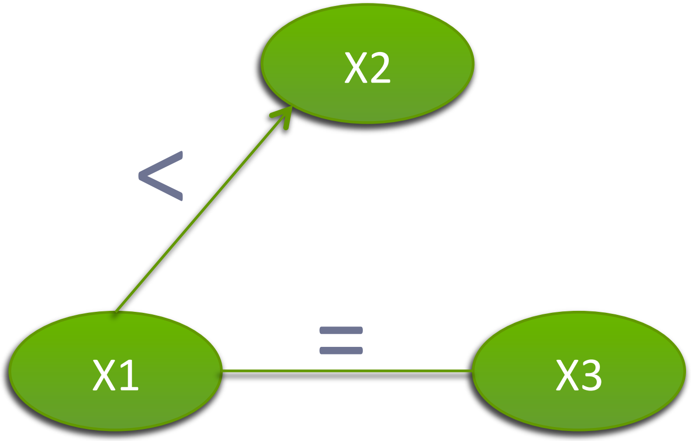
Convergence Pb:

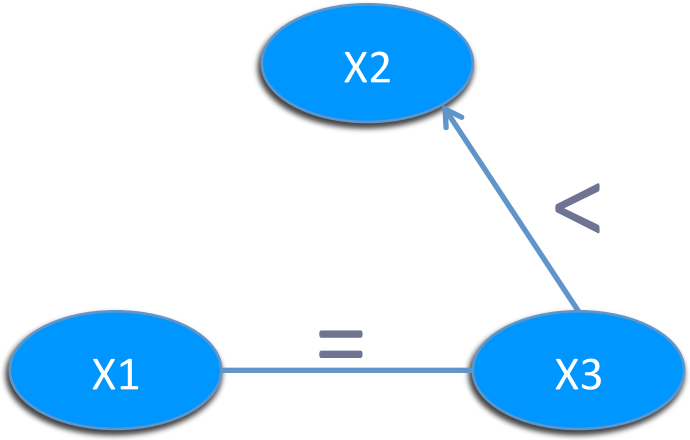
$$(C_L \subset B) \wedge (\forall e_i \in E^+ : e_i \in sol(C_L)) \wedge (\forall e_i \in E^- : e_i \notin sol(C_L))$$

coNP-complete **[Constraint Acquisition, AIJ17]**

Example

- $\Gamma = \{<, =\}$
- $B = \{x_i < x_j, x_i = x_j, \forall i, j\}$
- $C_T = \{x_1 = x_3, x_1 < x_2\}$
- $C_L = \{x_1 = x_3, x_3 < x_2\}$

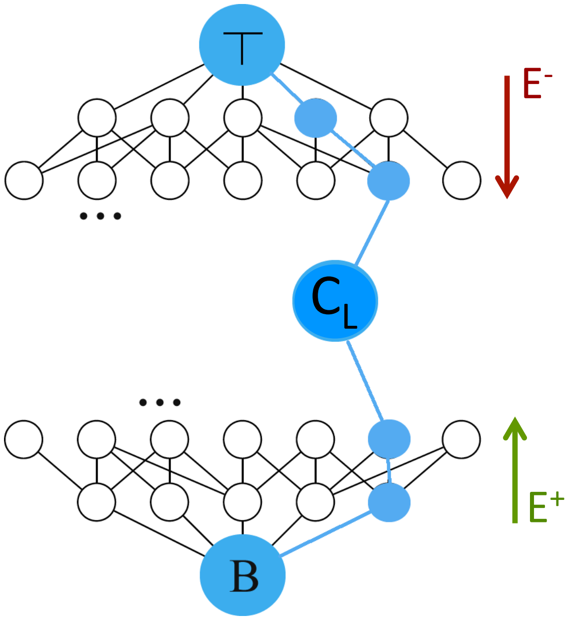




State of the art

- Matchmaker agents [Freuder and Wallace wAAAI97]
- CONACQ
  - SAT-Based constraint acquisition
  - Bidirectional search using Membership queries
  - Conacq1.0 (passive learning) [Bessiere et al. ECML05]
  - Conacq2.0 (active learning) [Bessiere et al. IJCAI07]

$$\mathcal{K} = \underbrace{(\neg x_1 \wedge \neg x_2 \wedge \neg x_3)}_{e^+} \bigwedge \underbrace{(x_4 \vee x_5 \vee x_6 \vee x_7)}_{e^-} \dots$$

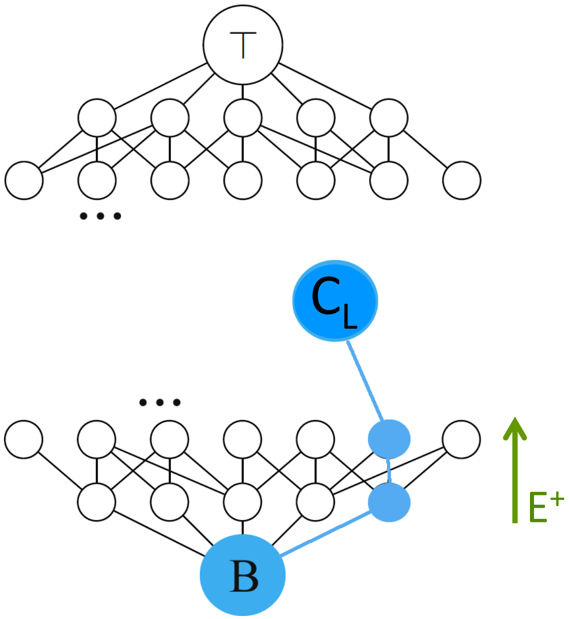


- Argument based CONACQ [Friedrich et al.09]

No-learnability using Membership queries [Constraint Acquisition, AIJ17]

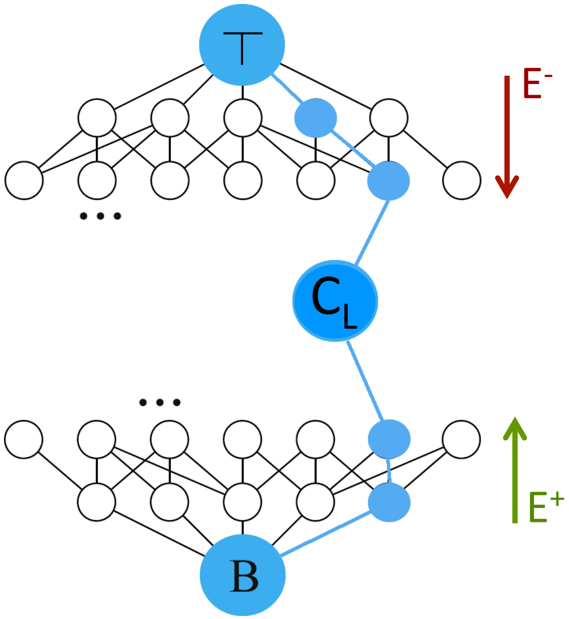
State of the art

- ModelSeeker [Beldiceanu and Simonis, CP11’12]
  - A passive learning
  - Based on global constraint catalogue (≈1000)
  - Buttom-up search

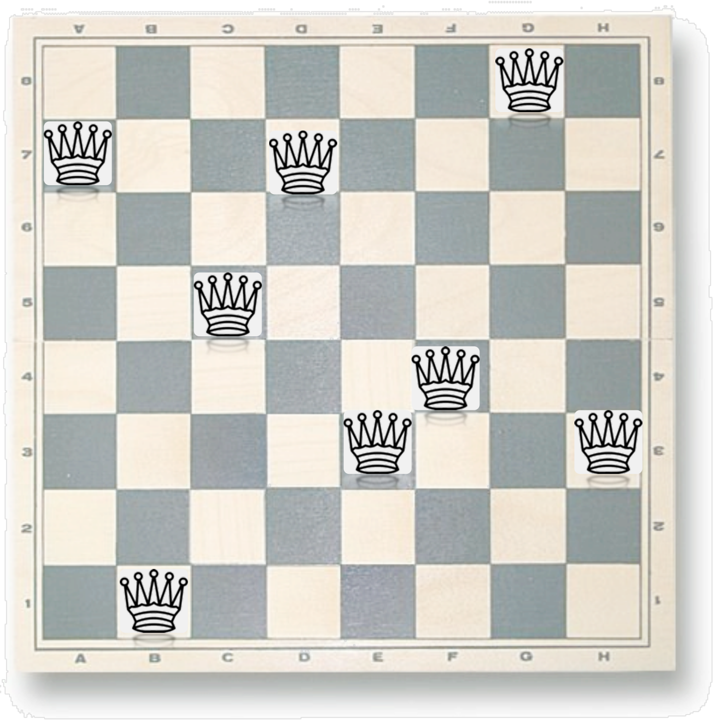


QUACQ: Quick Acquisition

- QUACQ [Bessiere et al. IJCAI13]
  - Active learning approach
  - Bidirectional search
    - But it can be top-down search only if no positive example
  - Based on partial queries to elucidate the scope of the constraint to learn

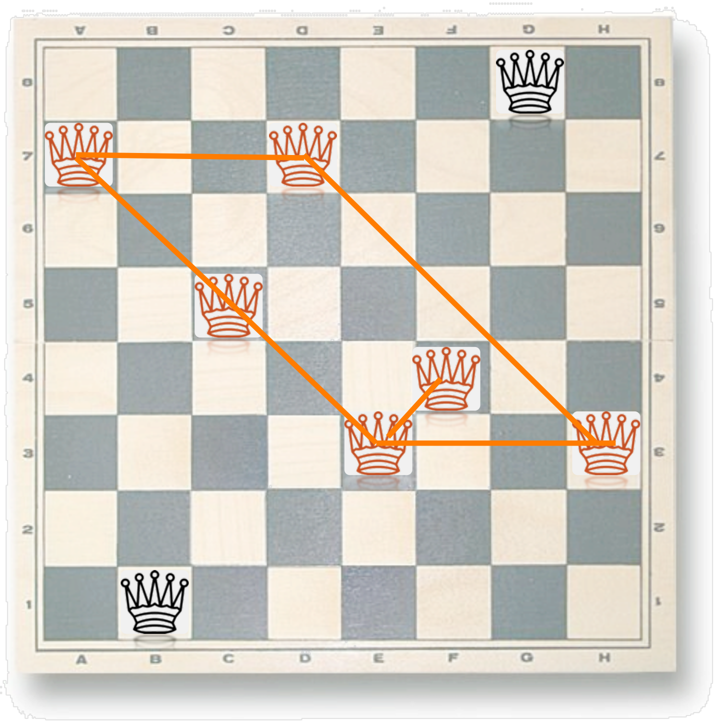


Membership Queries



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

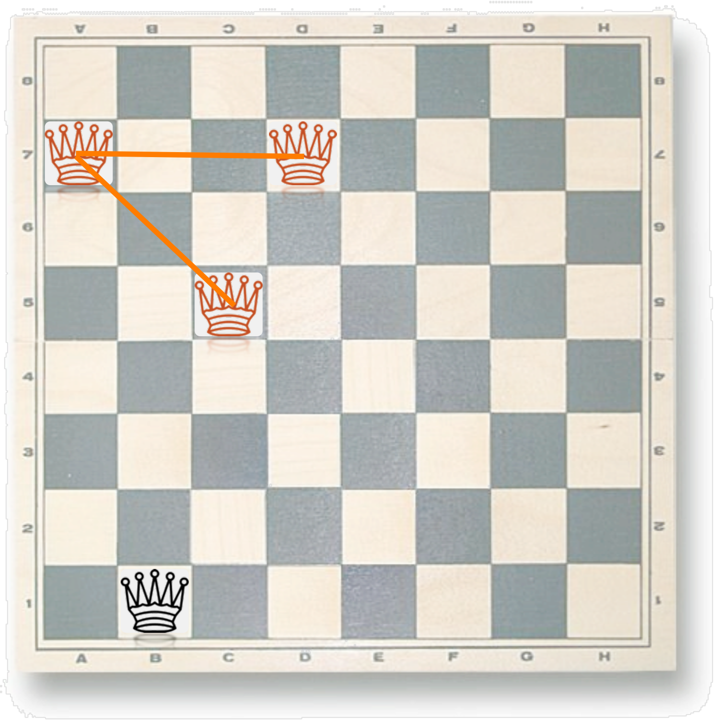
Partial Queries



✗

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

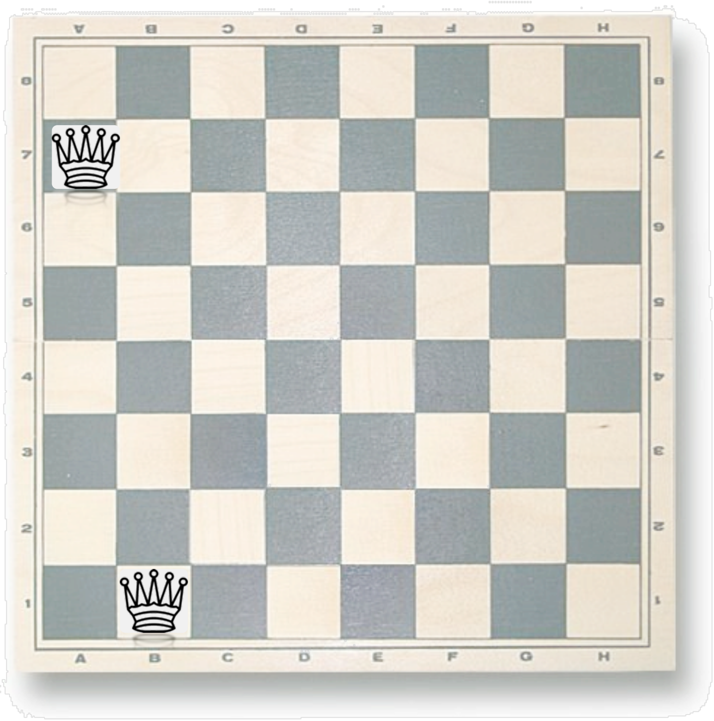
Partial Queries



✗

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

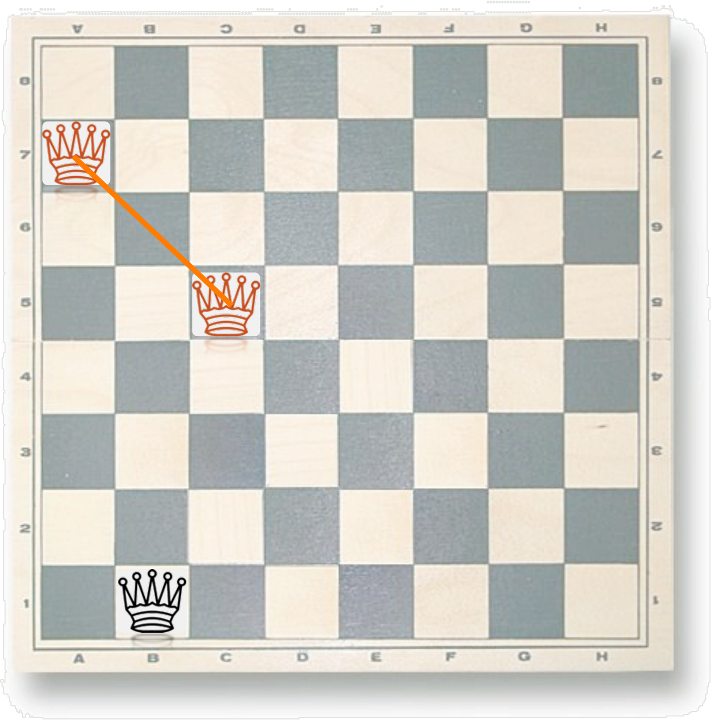
Partial Queries



✓

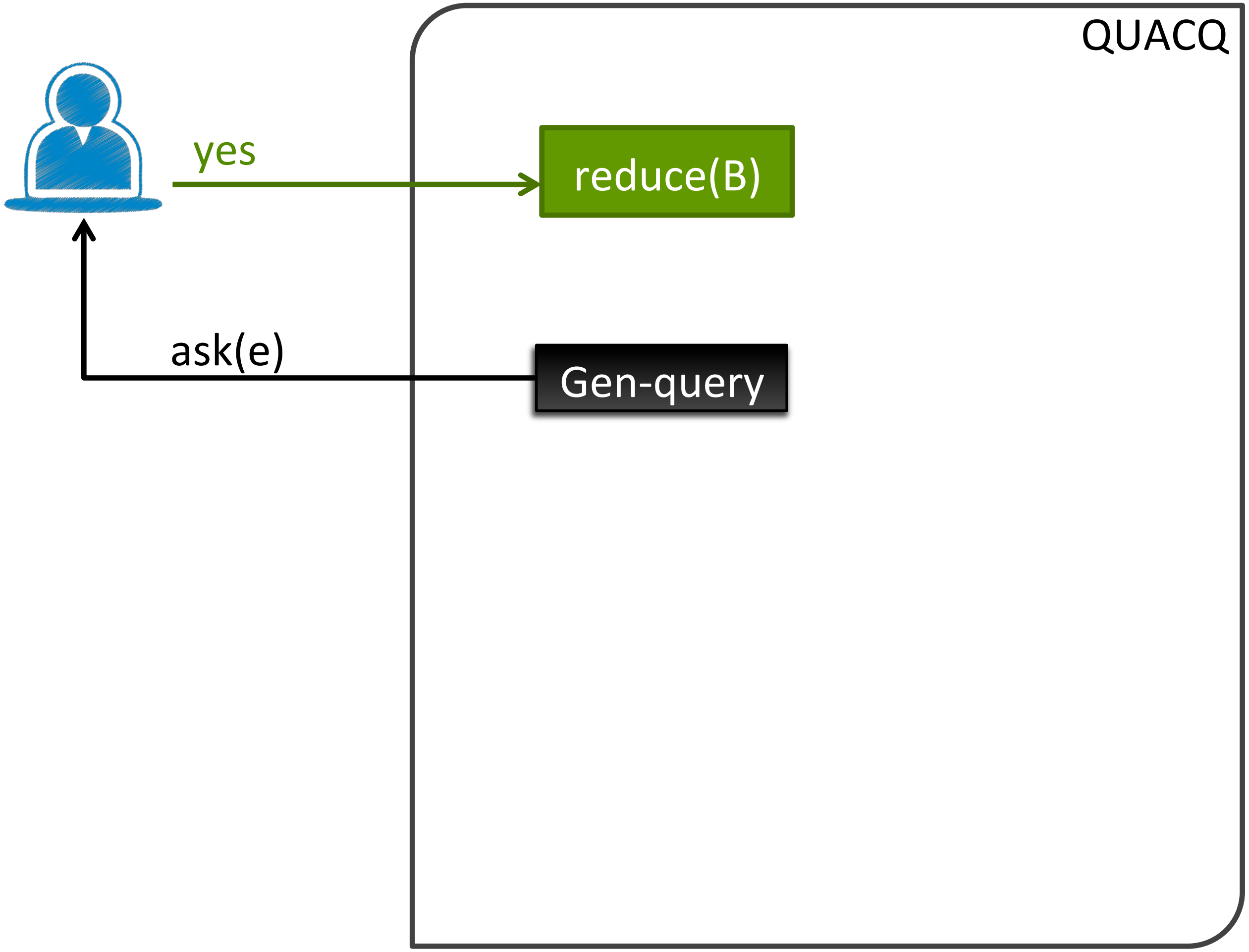
ask(2, 8, -, -, -, -, -, -) = Yes

Partial Queries



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

QUACQ: Quick Acquisition



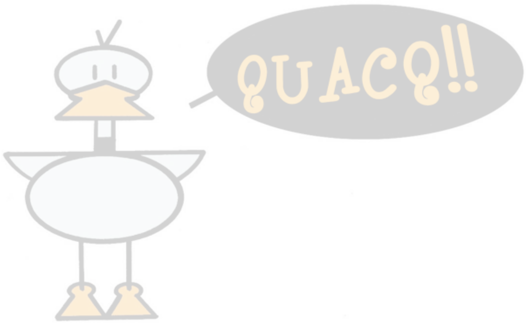
QUACQ

yes

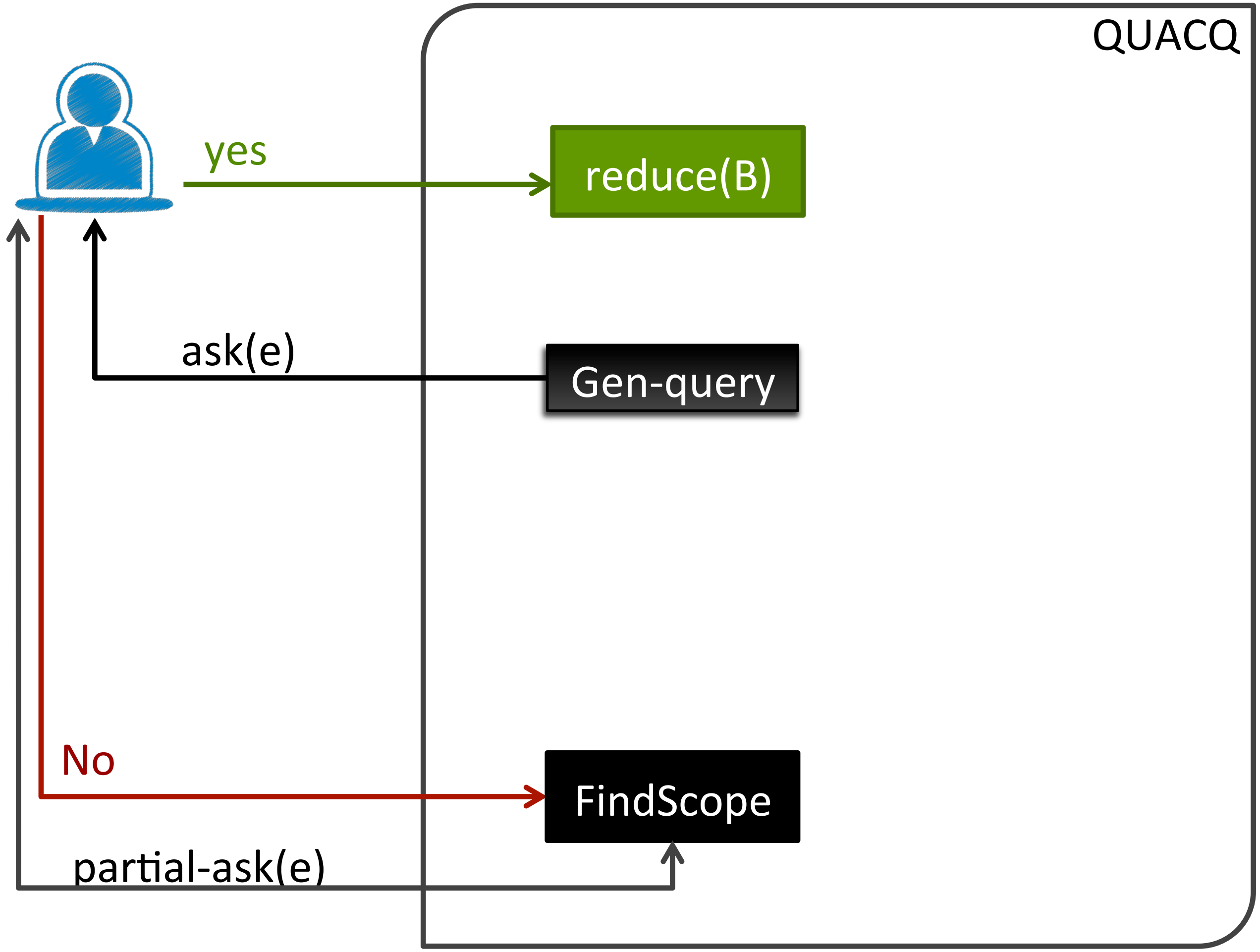
reduce(B)

ask(e)

Gen-query



QUACQ: Quick Acquisition



QUACQ

yes

reduce(B)

ask(e)

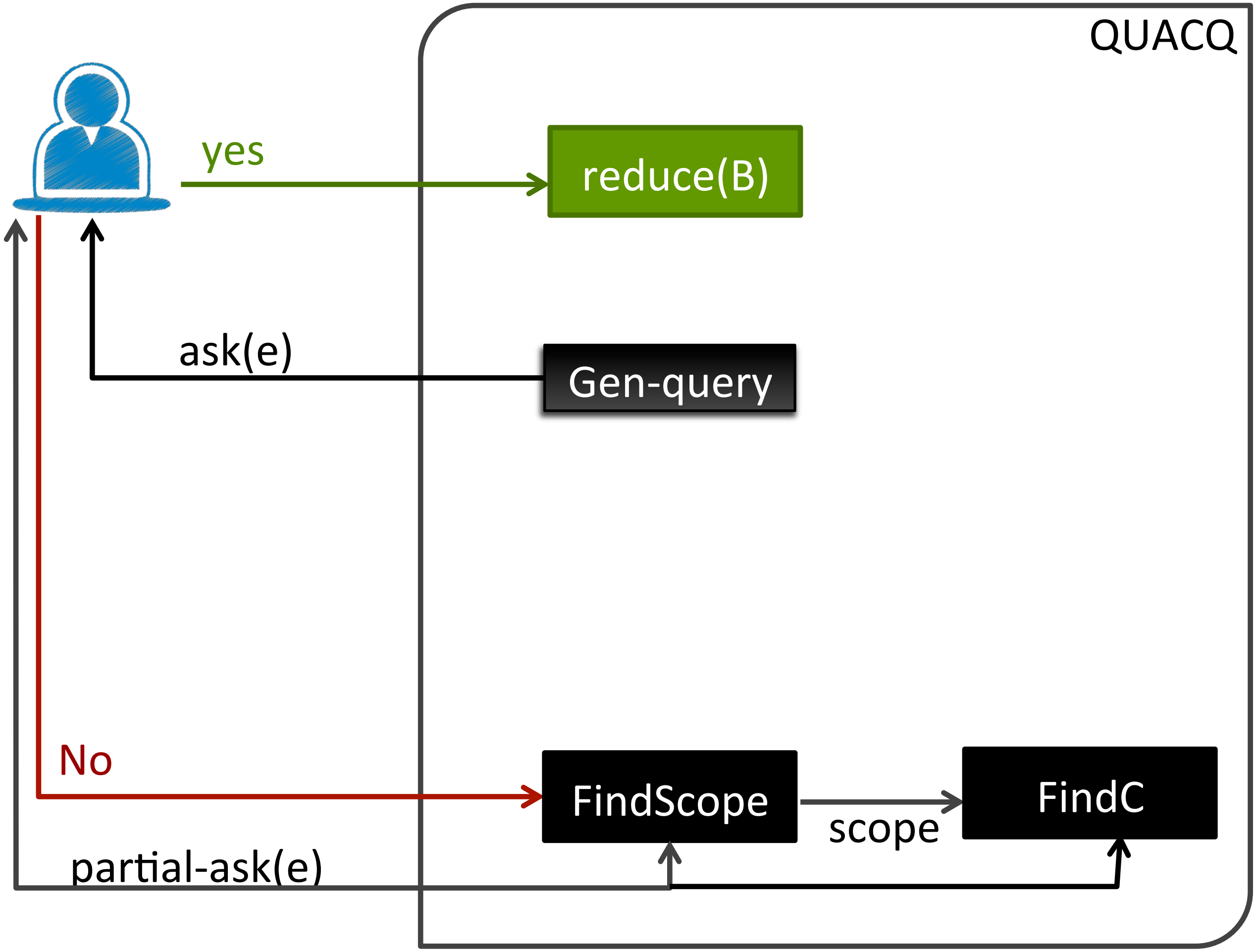
Gen-query

No

FindScope

partial-ask(e)

QUACQ: Quick Acquisition



QUACQ

yes

reduce(B)

ask(e)

Gen-query

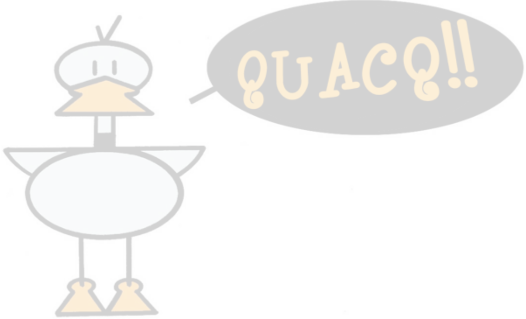
No

FindScope

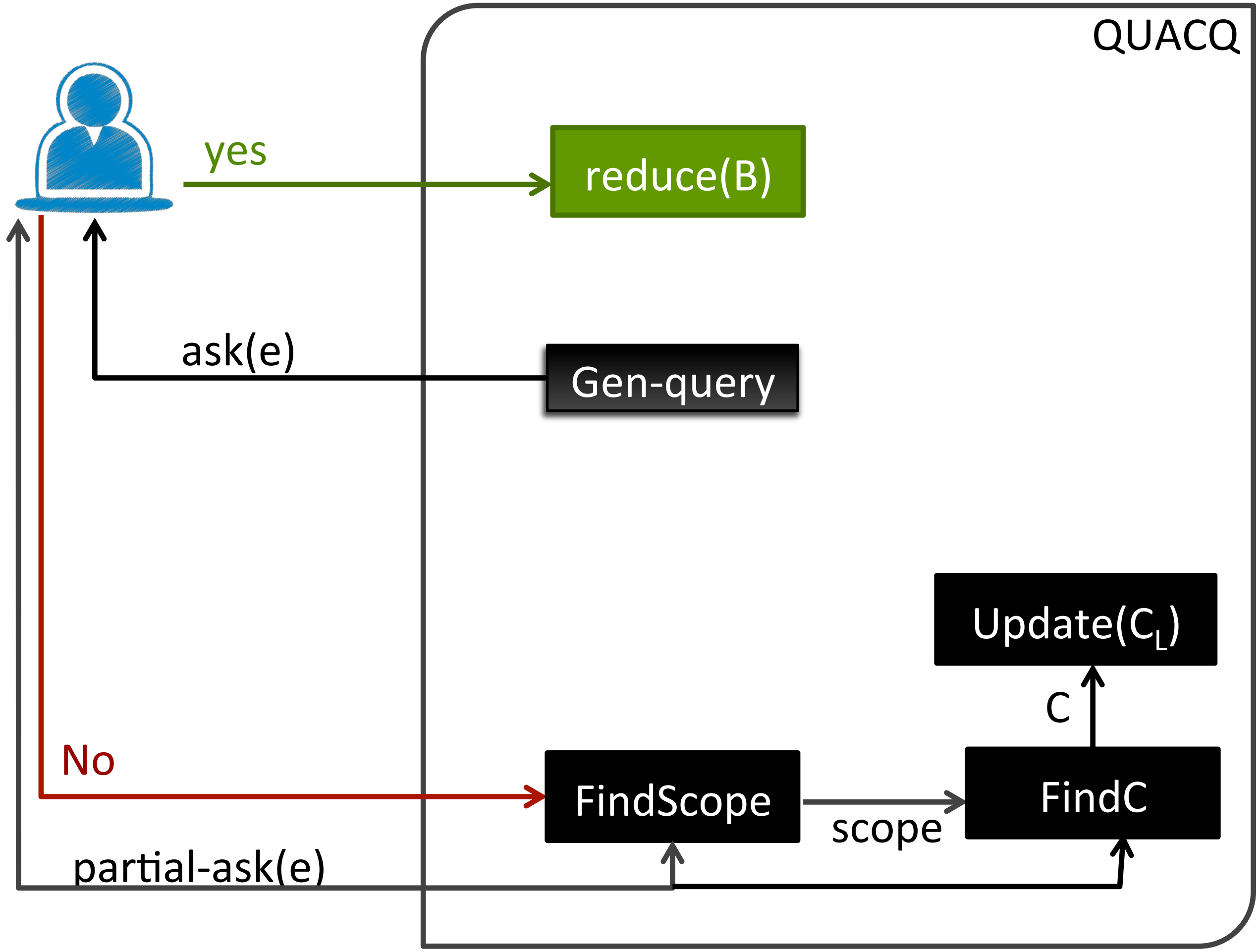
scope

FindC

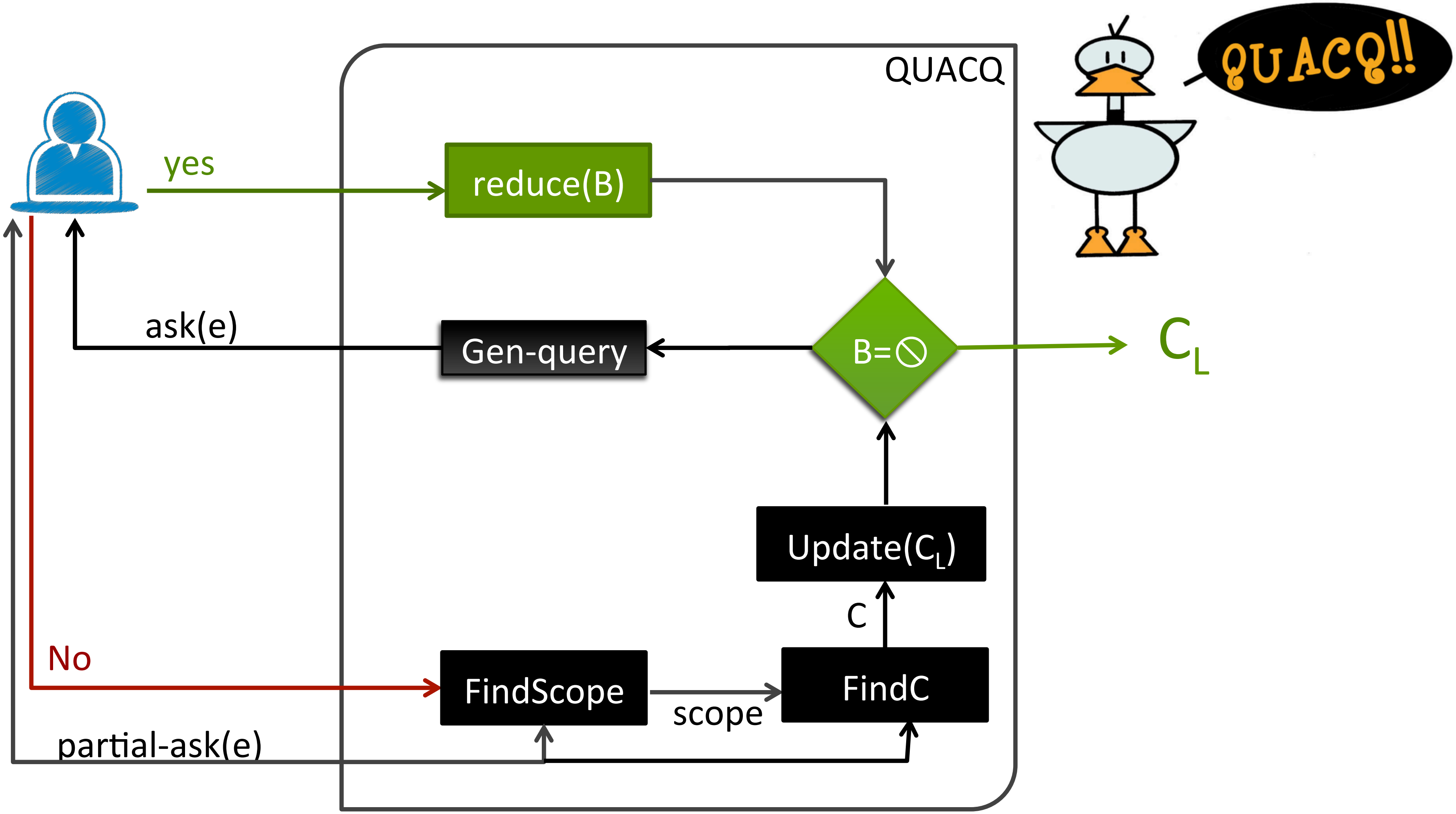
partial-ask(e)



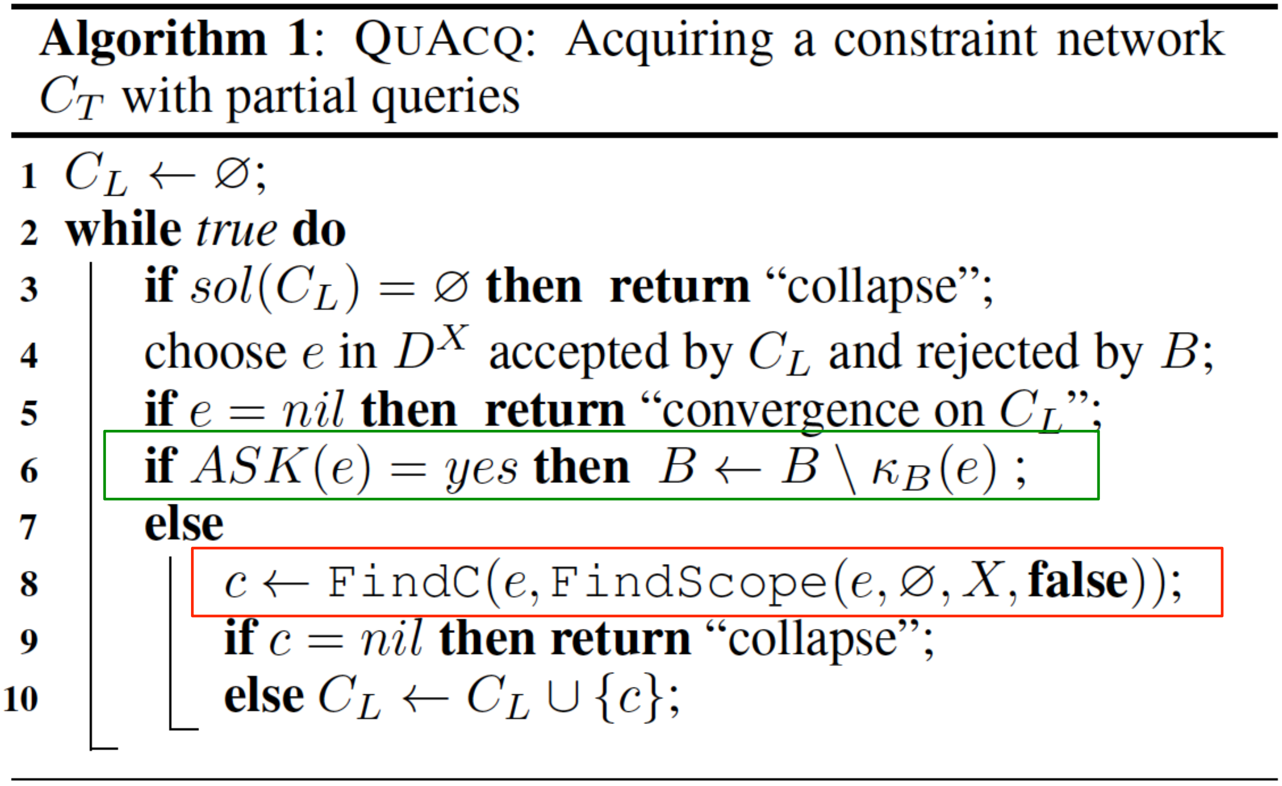
QUACQ: Quick Acquisition



QUACQ: Quick Acquisition



QUACQ: Quick Acquisition



Algorithm 1: QUACQ: Acquiring a constraint network C_T with partial queries

```
$C_L \leftarrow \varnothing$;
while *true* do
    if $sol(C_L) = \varnothing$ then return "collapse";
    choose $e$ in $D^X$ accepted by $C_L$ and rejected by $B$;
    if $e = nil$ then return "convergence on $C_L$";
    if $ASK(e) = yes$ then $B \leftarrow B \setminus \kappa_B(e)$ ;
    else
        $c \leftarrow$ FindC($e$, FindScope($e, \varnothing, X$, false));
        if $c = nil$ then return "collapse";
        else $C_L \leftarrow C_L \cup \{c\}$;
```

Complexity of QUACQ

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

$$O(|C_T| \cdot (\log |X| + |\Gamma|))$$

- The number of queries required to converge is in:

$$O(|B|)$$

E^-

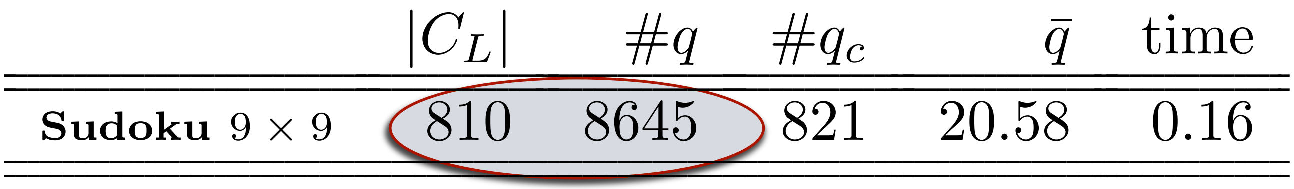
E^+

Some Results

- Sudoku

A target network on 81 variables with 810 constraints

Bias of 19440 binary constraints

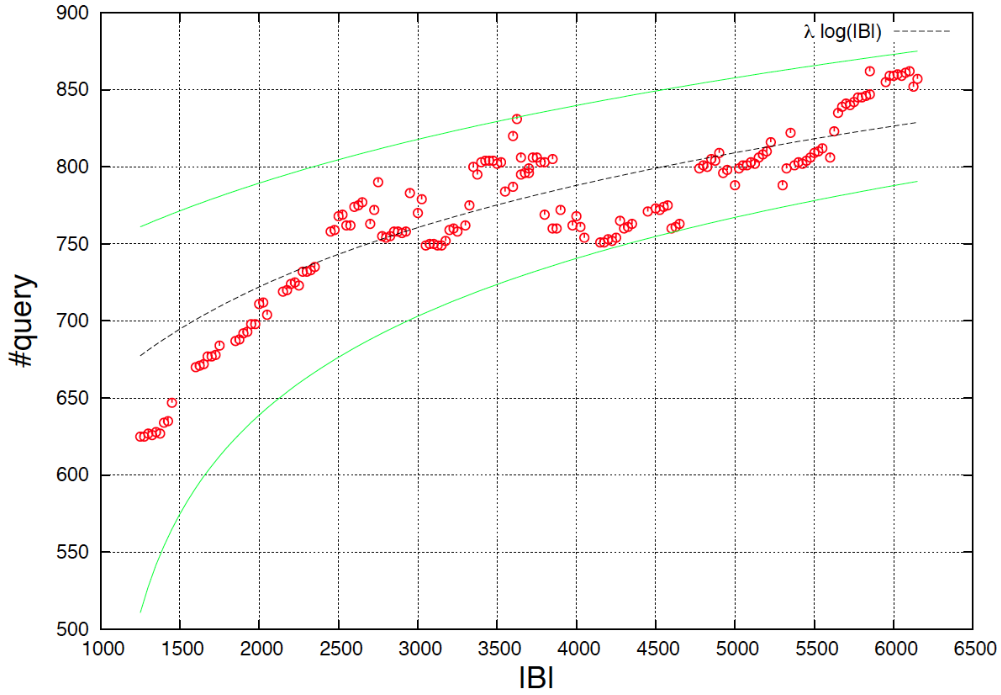


	$\lvert C_L \rvert$	$\#q$	$\#q_c$	$\bar{q}$	time
Sudoku 9×9	810	8645	821	20.58	0.16

Intel Xeon E5462 @ 2.80GHz with 16 Gb of RAM.

Experiments

- Zebra puzzle
  - QUACQ behavior on different bias sizes



Intel Xeon E5462 @ 2.80GHz with 16 Gb of RAM.

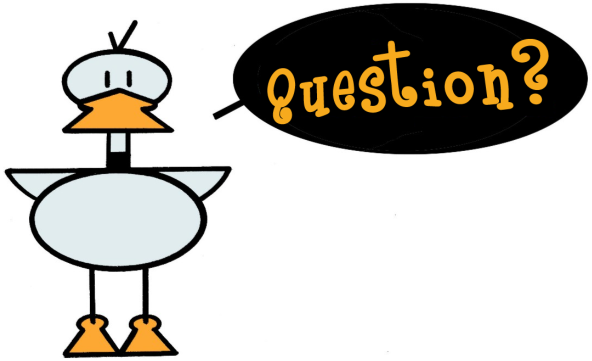
Conclusions

- QUACQ: new constraint acquisition approach based on partial queries
  - Active learning approach
  - Learning a constraint in a log scale of #queries
  - Queries are often much shorter than membership ones
  - Can follow a top-down search to learn a constraint network

Time left?

Constraint Acquisition

Nadjib Lazaar



U. Montpellier, France
LIRMM - COCONUT team

24-11-17
Jussieu

In practice?

Limitation:

- Hard to put in practice:
  - QUACQ needs more than **8000** queries to learn the Sudoku model

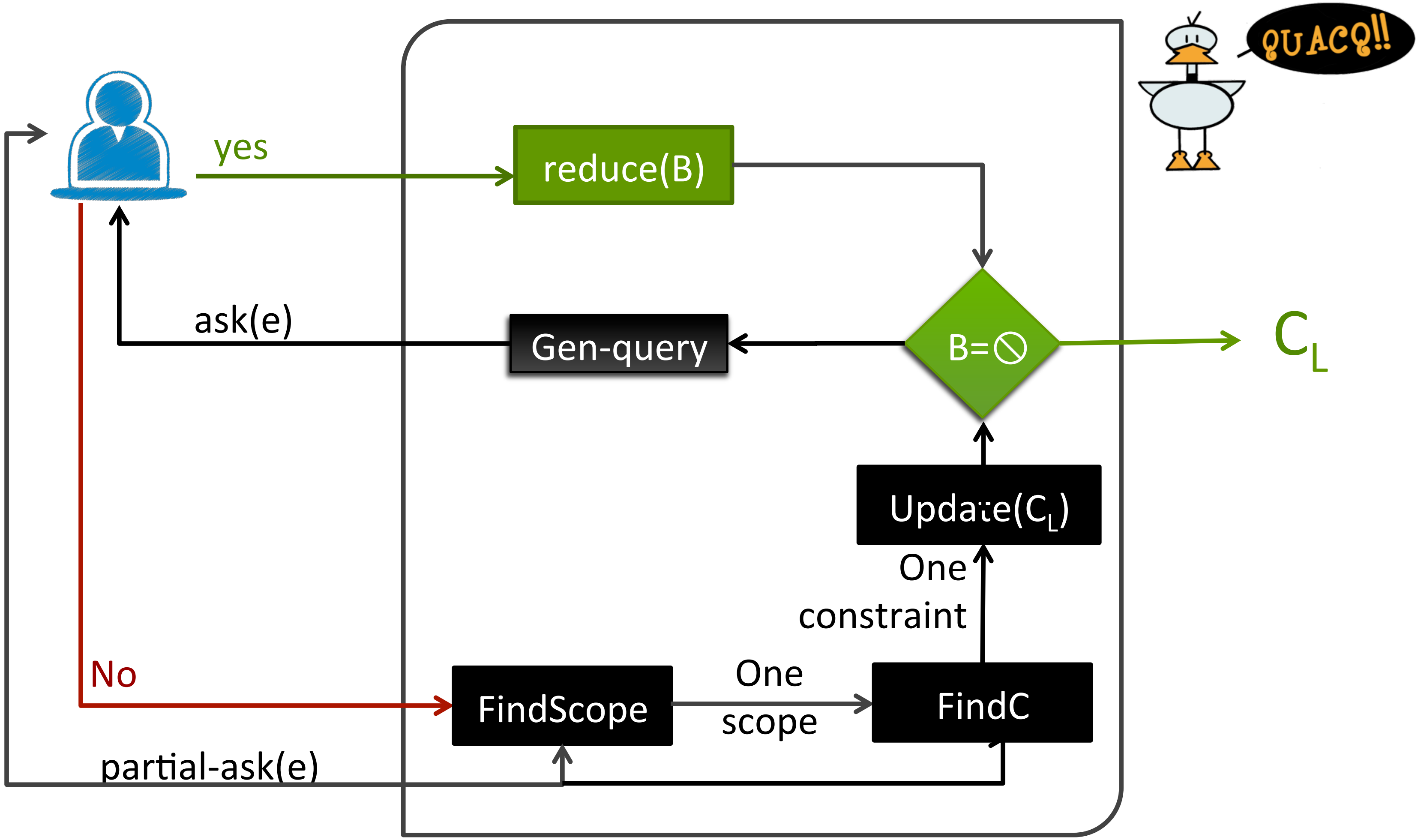
Need:

- Reduce the dialogue with the user to make constraint acquisition more efficient in practice

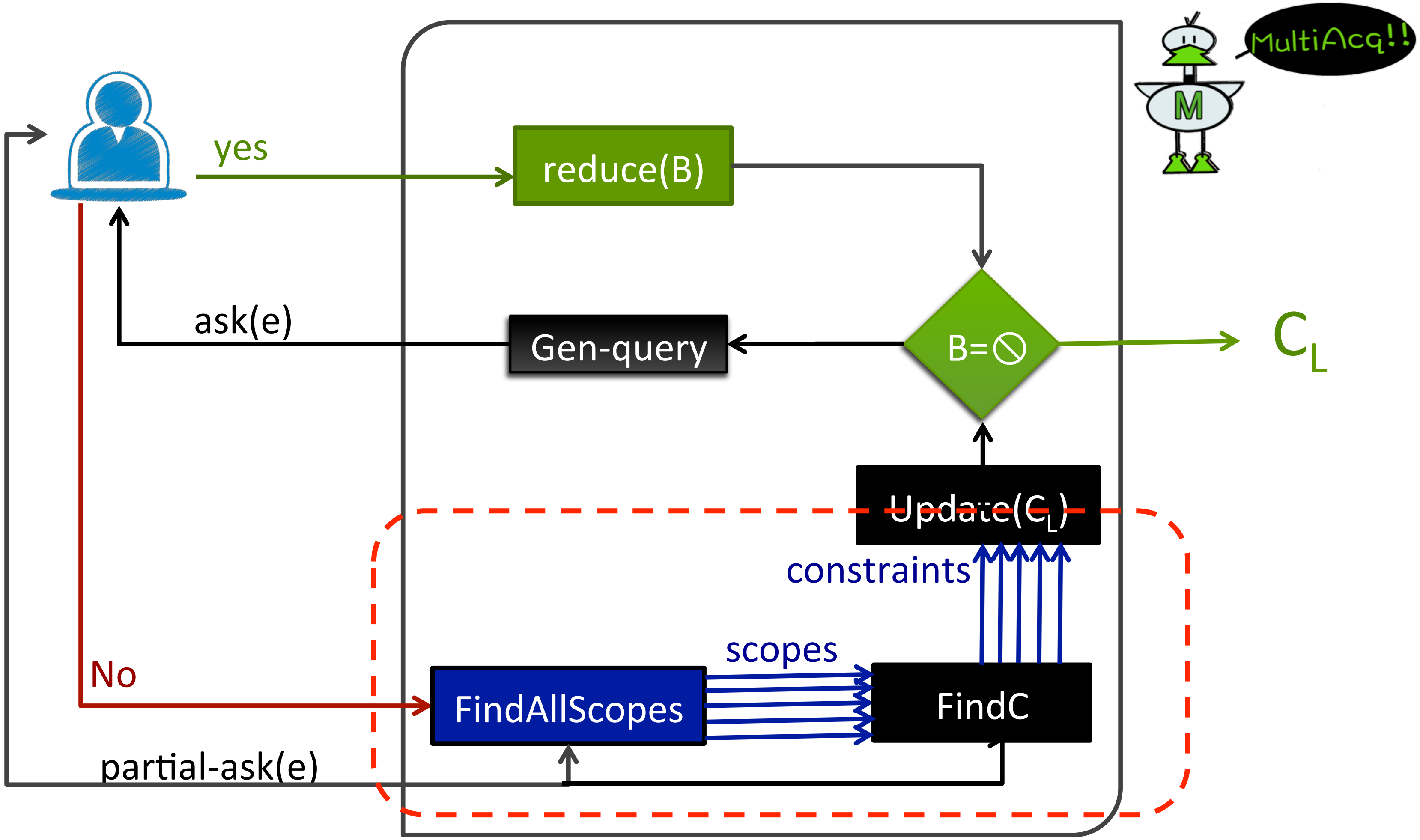
How:

- Eliciting more information on why a complete instantiation is classified as negative by the user

QUACQ: Quick Acquisition

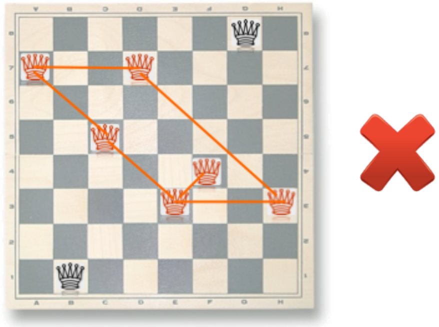


MULTIACQ: Multiple Acquisition [IJCAI-W15]



MULTIACQ: Multiple Acquisition

e: Membership query



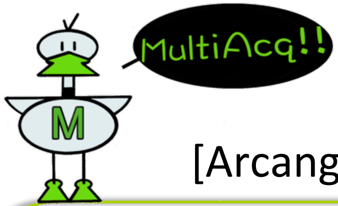
Q: Why the user said No?

[Bessiere et al., IJCAI13]

- **FindScope** function
- QuickXplain like function [Junker 04]
- Returns **one** scope (explanation)

➔ FindScope(e)=(X3,x5)

➔ #learned_constraint = **1**



[Arcangioli et al., IJCAI16]

- **FindAllScopes** function
- CAMUS like function [Liffiton et al. 07]
- Returns **all** Minimal No Scopes (MUS in SAT)

➔ FindAllScope(e)={(X1,x3), (X1,x4), (X3,x5), (X5,x6), (X5,x8),(X4,x8)}

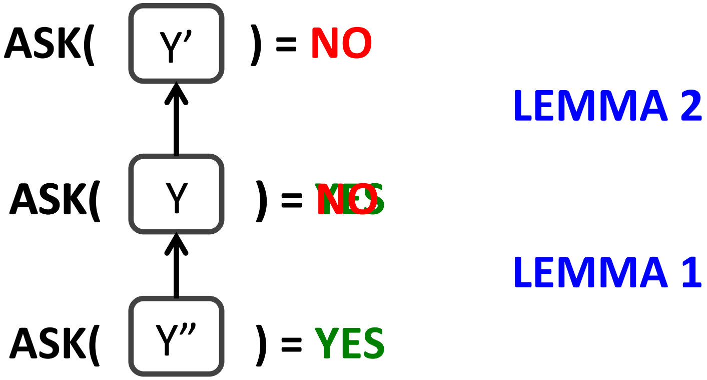
➔ #learned_constraint = **6**

MNS: Minimal No Scope

- Given a negative example e, an MNS is a subset of variables $U \subseteq X$ such that:

$$ASK(e_U) = no \text{ and } \forall x_i \in U : ASK(e_{U \setminus x_i}) = yes$$

- Lemmas:



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = (X1, X2) , (X1, X3), (X2, X3, X4)

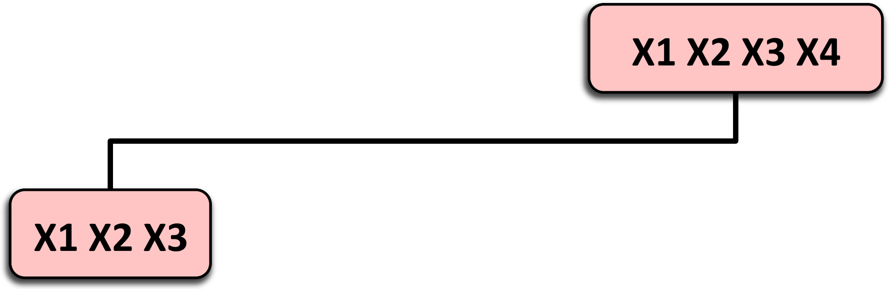
#Recusive calls	#ask
1	1

X1 X2 X3 X4

FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = (X1, X2) , (X1, X3), (X2, X3, X4)

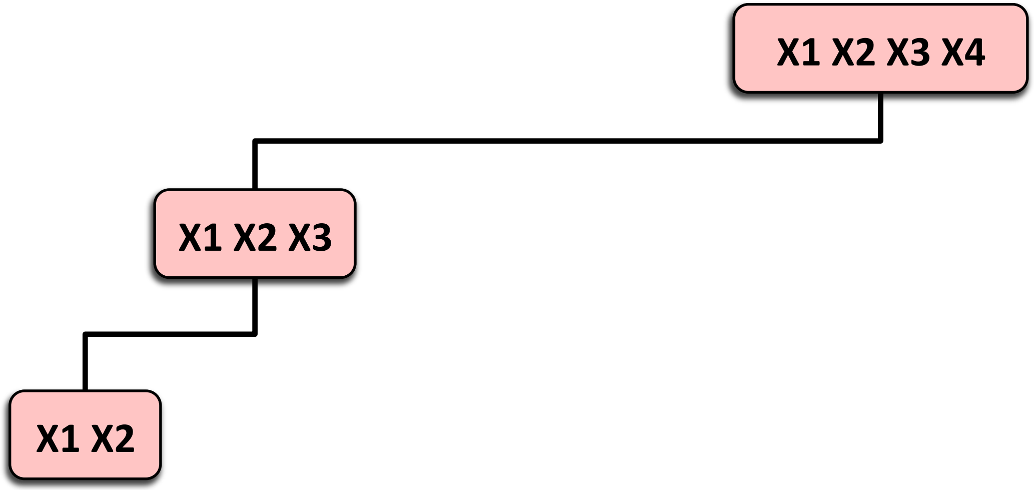
#Recusive calls	#ask
2	2



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = (X1, X2) , (X1, X3), (X2, X3, X4)

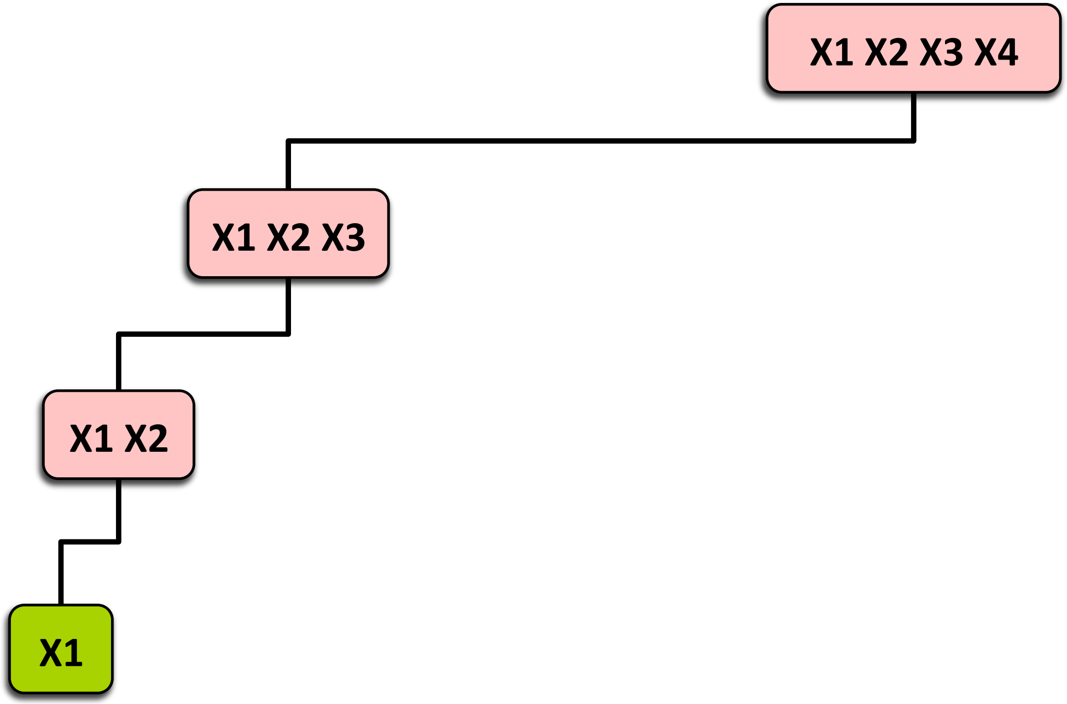
#Recusive calls	#ask
3	3



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = (X1, X2) , (X1, X3), (X2, X3, X4)

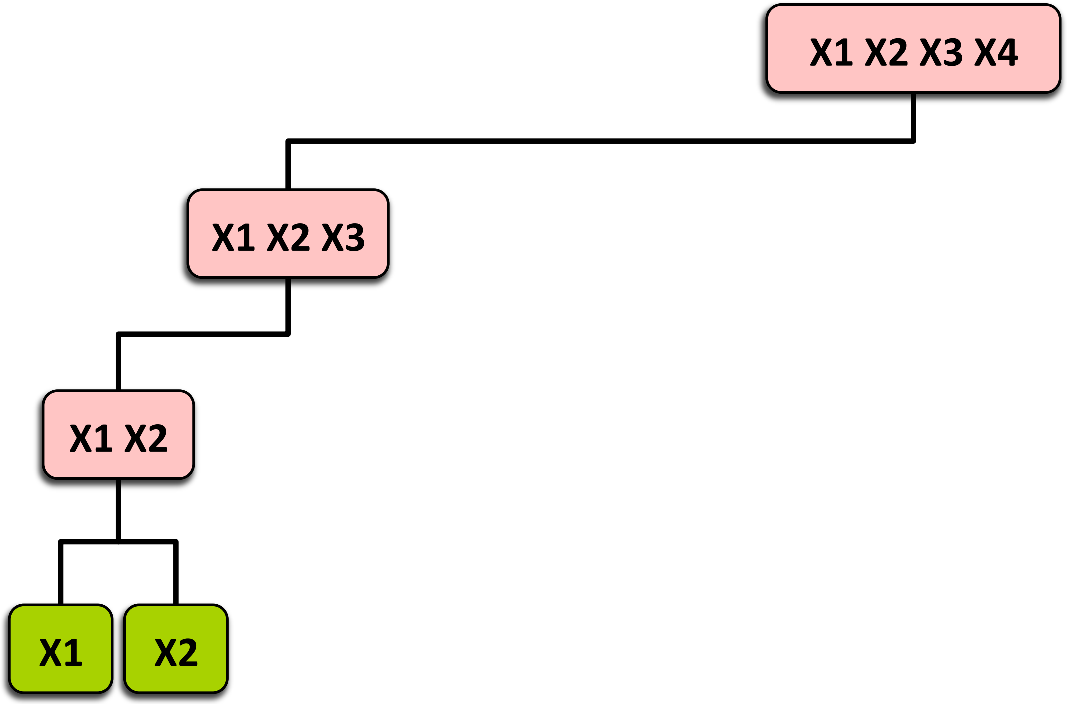
#Recusive calls	#ask
4	4



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = (X1, X2) , (X1, X3), (X2, X3, X4)

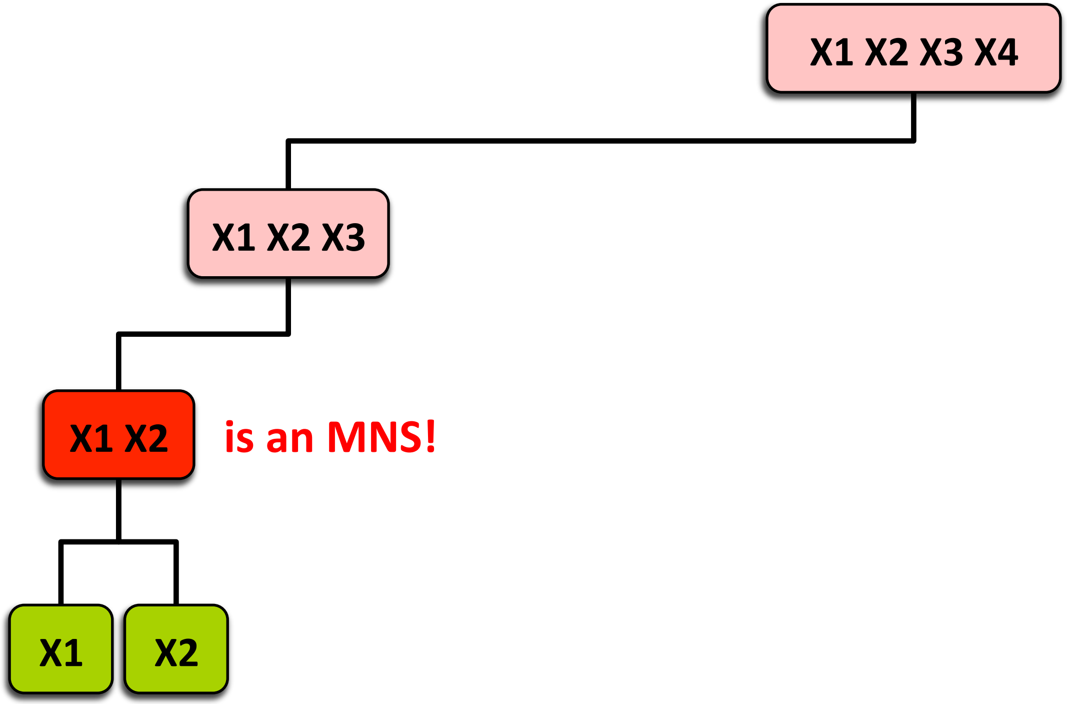
#Recusive calls	#ask
5	5



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = **(X1, X2)** , (X1, X3), (X2, X3, X4)

#Recusive calls	#ask
5	5

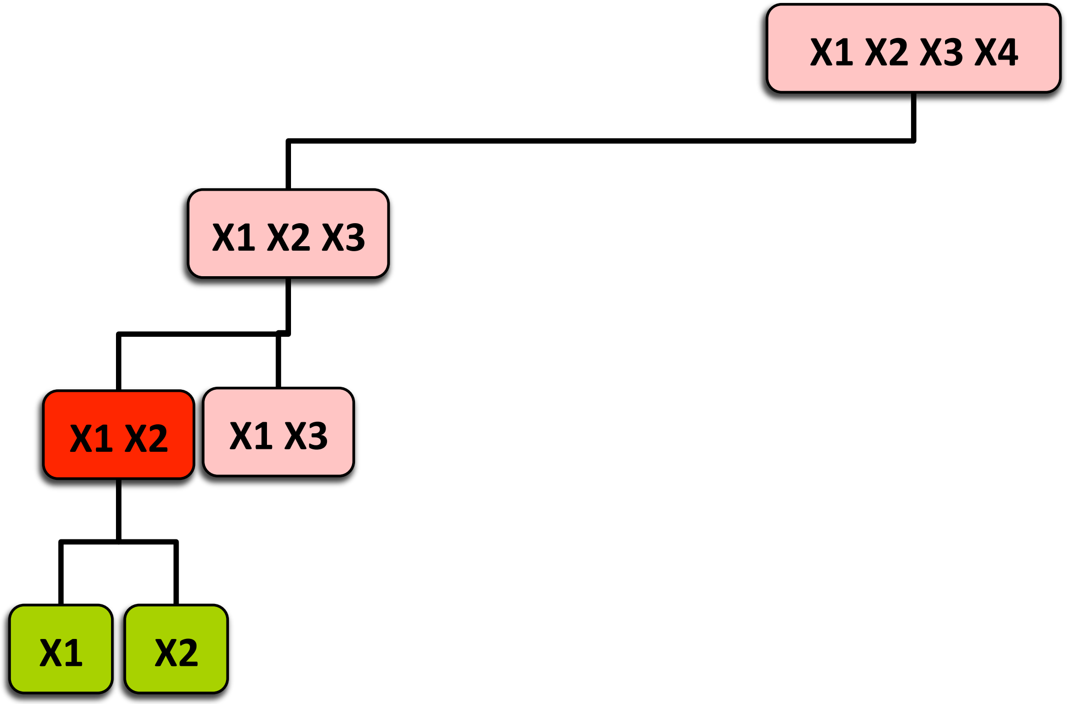


is an MNS!

FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = **(X1, X2)** , (X1, X3), (X2, X3, X4)

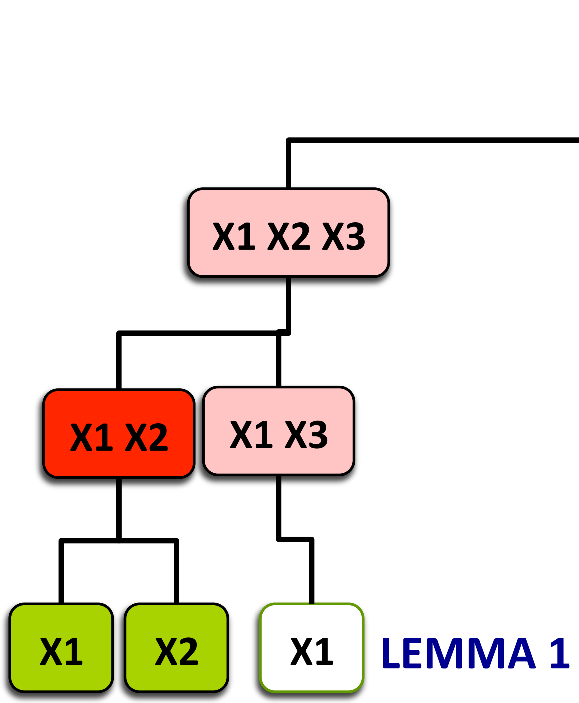
#Recusive calls	#ask
6	6



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = **(X1, X2)** , (X1, X3), (X2, X3, X4)

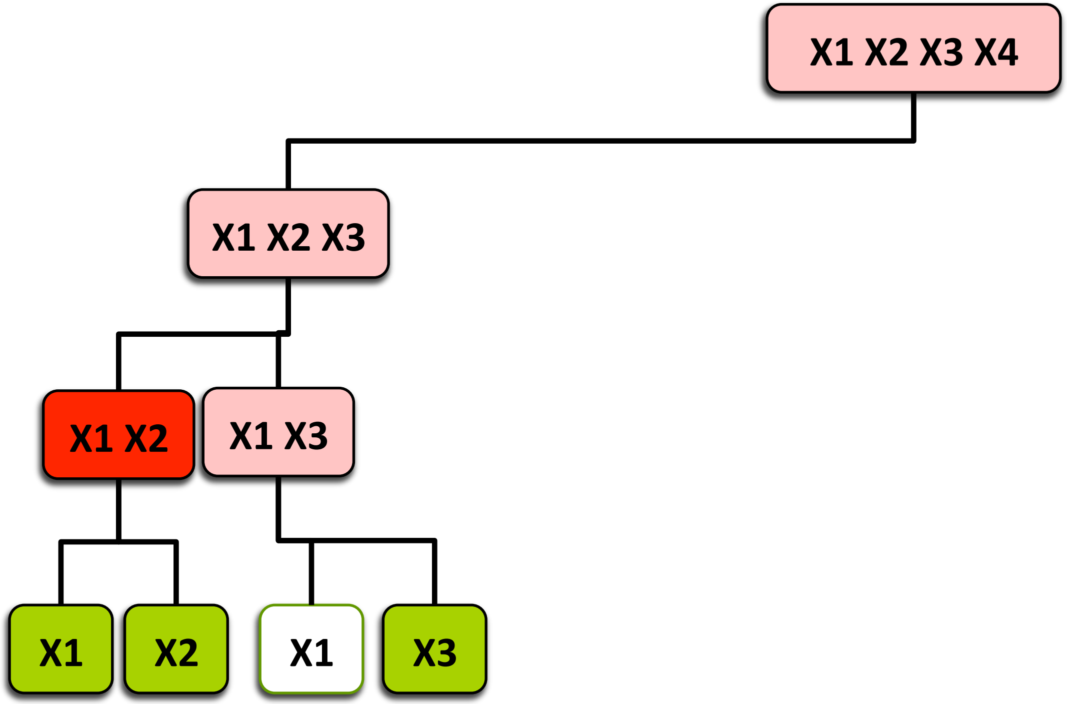
#Recusive calls	#ask
7	6



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = **(X1, X2)** , (X1, X3), (X2, X3, X4)

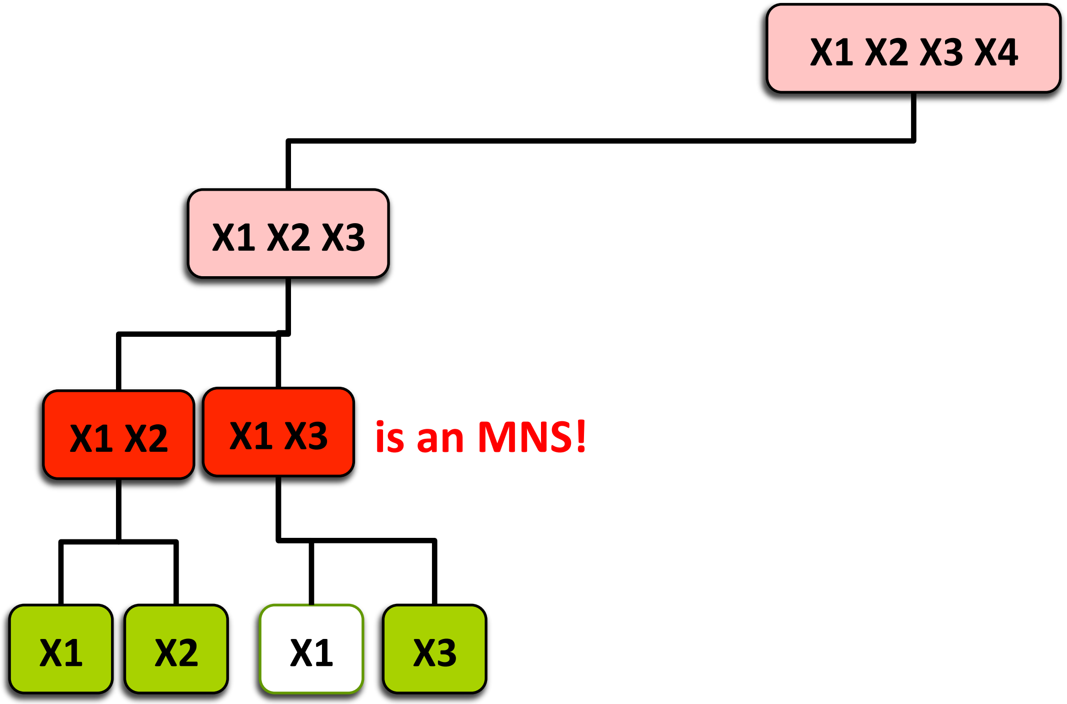
#Recusive calls	#ask
8	7



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = **(X1, X2)** , **(X1, X3)**, (X2, X3, X4)

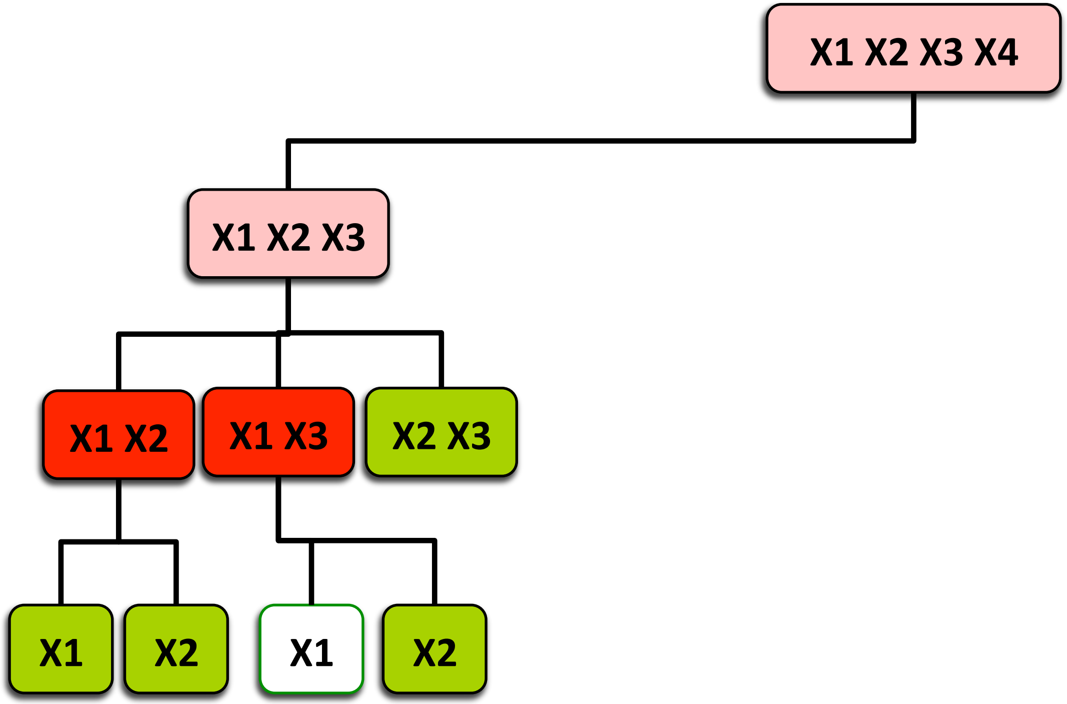
#Recusive calls	#ask
8	**7**



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = **(X1, X2)** , **(X1, X3)**, (X2, X3, X4)

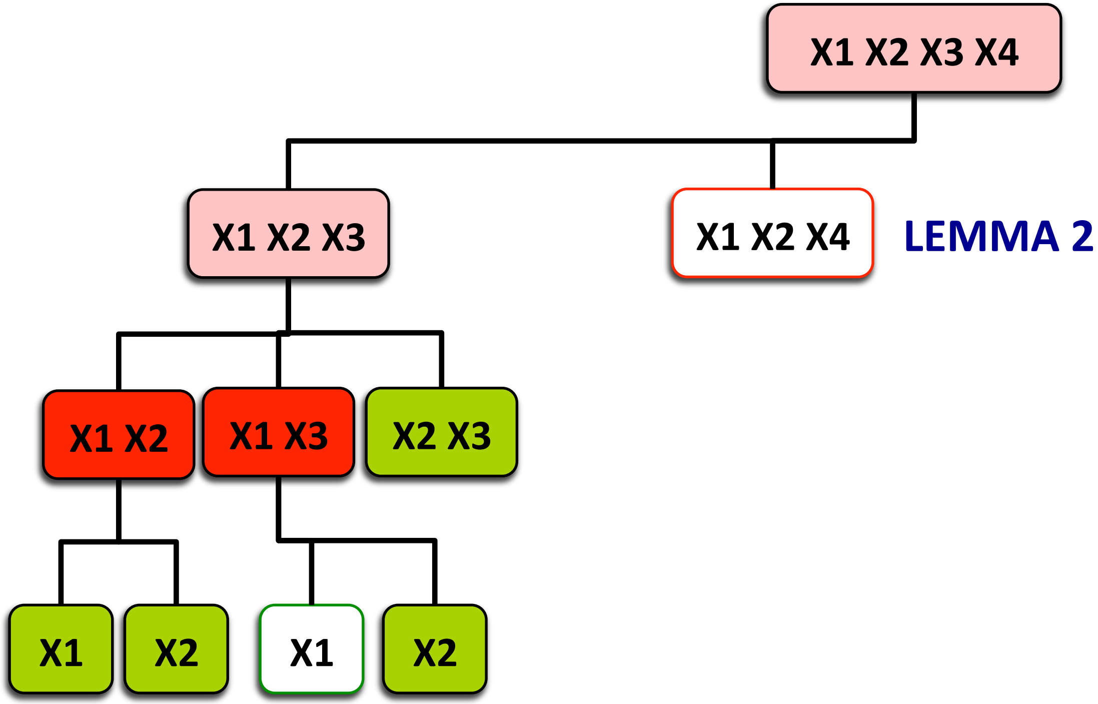
#Recusive calls	#ask
9	8



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = **(X1, X2)** , **(X1, X3)**, (X2, X3, X4)

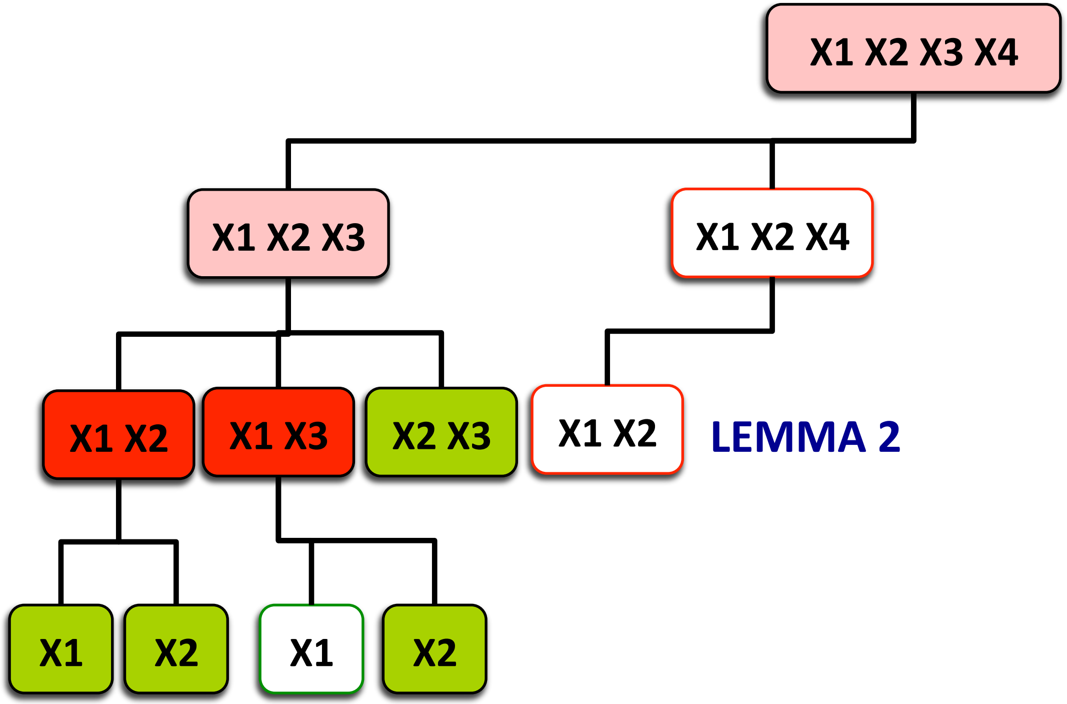
#Recusive calls	#ask
10	8



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = **(X1, X2)** , **(X1, X3)**, (X2, X3, X4)

#Recusive calls	#ask
11	8

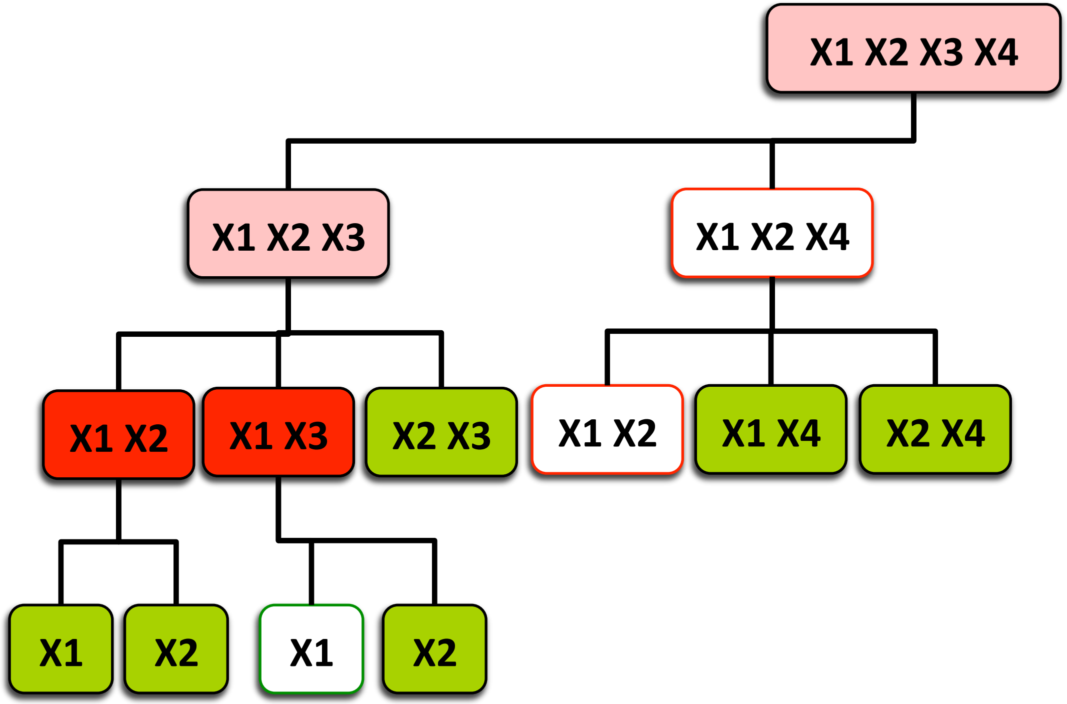


LEMMA 2

FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = **(X1, X2)** , **(X1, X3)**, (X2, X3, X4)

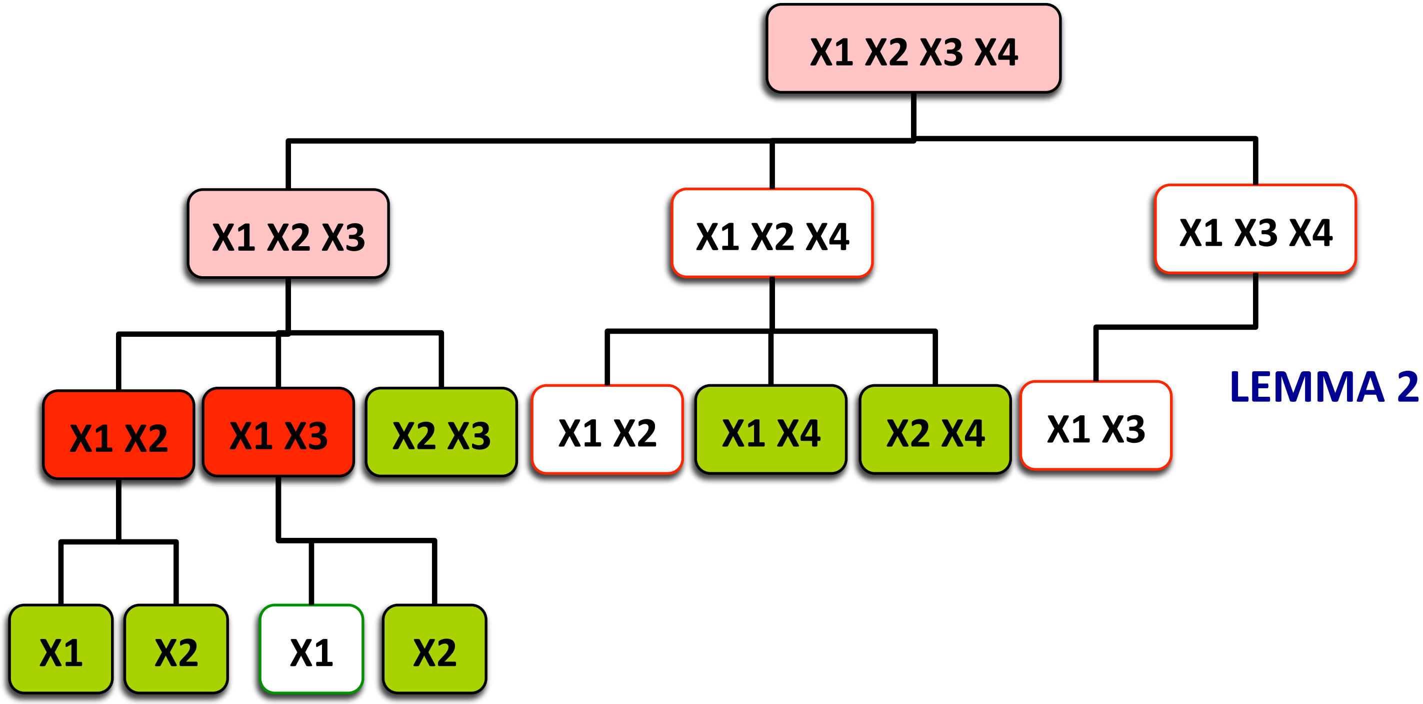
#Recusive calls	#ask
13	10



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = **(X1, X2)** , **(X1, X3)**, (X2, X3, X4)

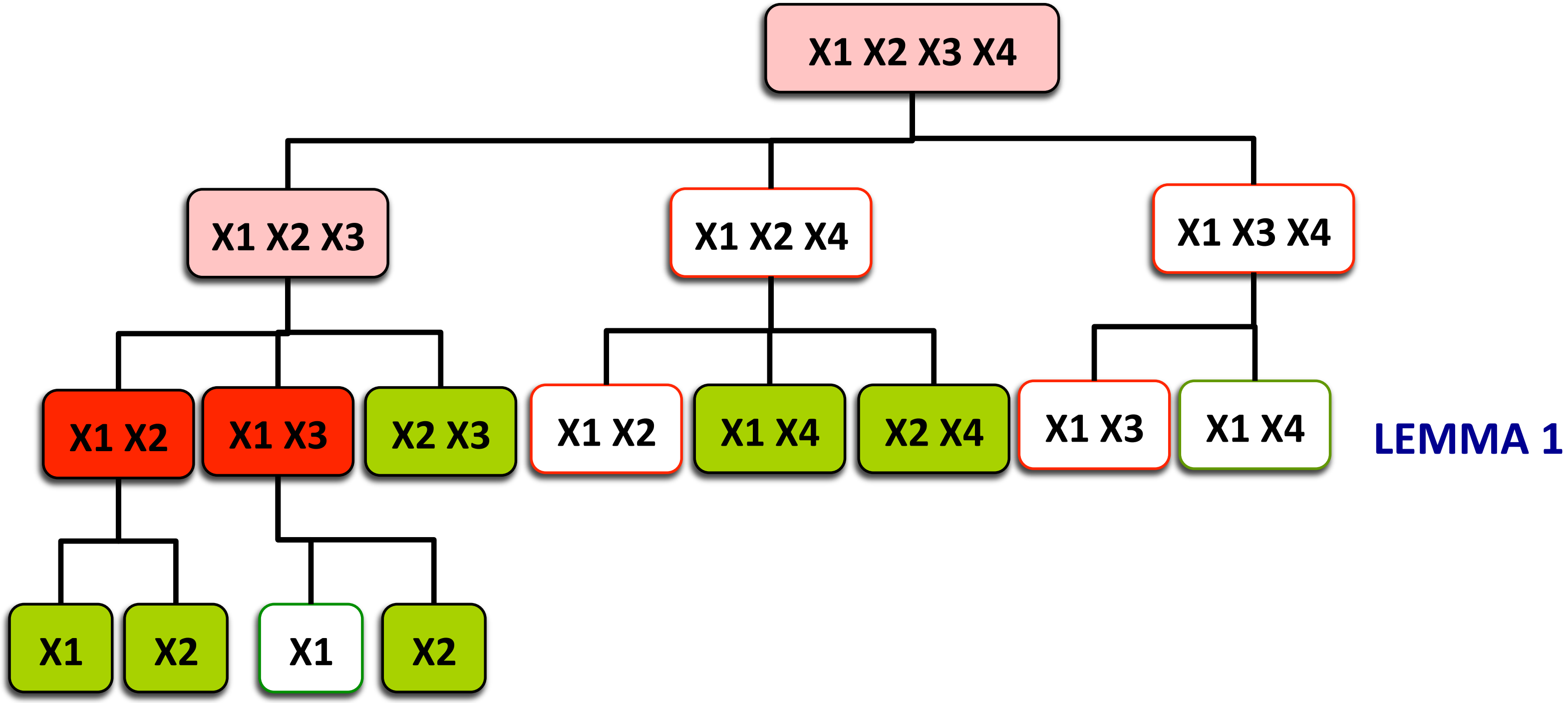
#Recusive calls	#ask
15	10



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = **(X1, X2)** , **(X1, X3)**, (X2, X3, X4)

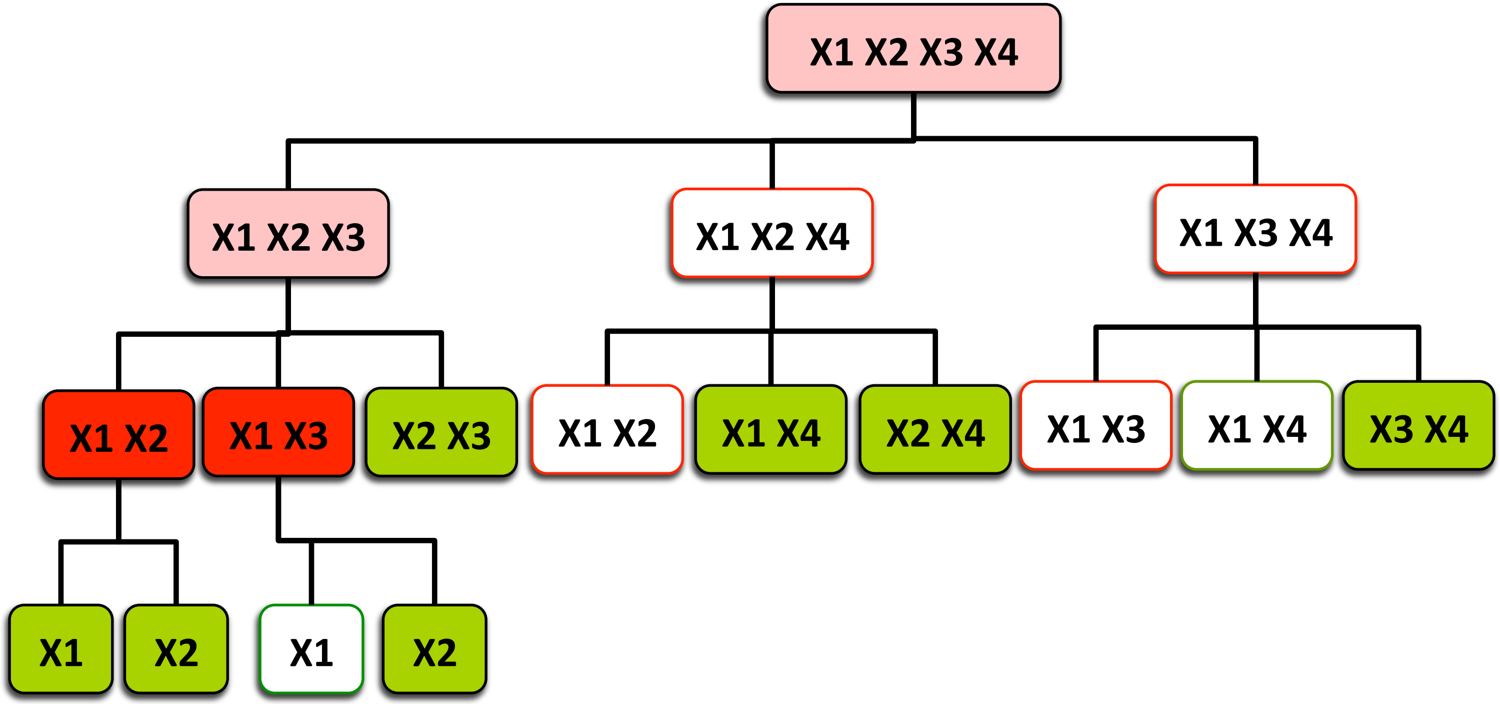
#Recusive calls	#ask
16	10



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = **(X1, X2)** , **(X1, X3)**, (X2, X3, X4)

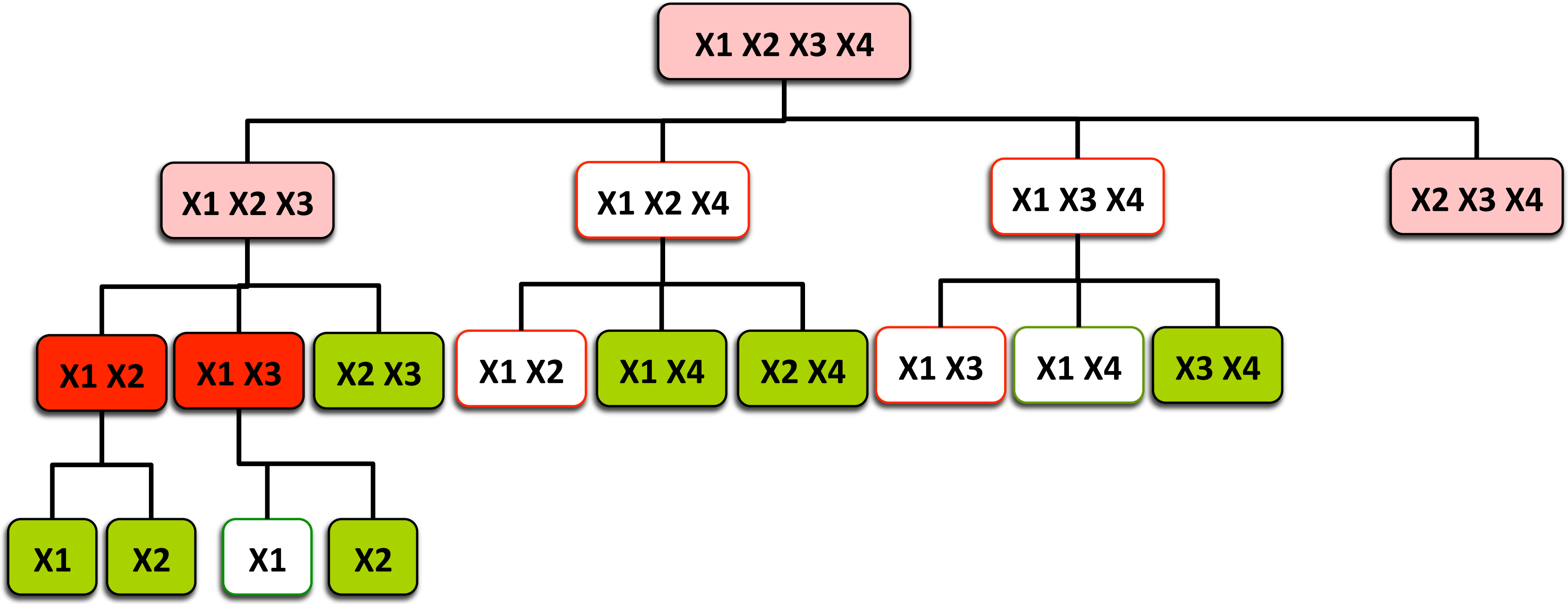
#Recusive calls	#ask
17	11



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = **(X1, X2)** , **(X1, X3)**, (X2, X3, X4)

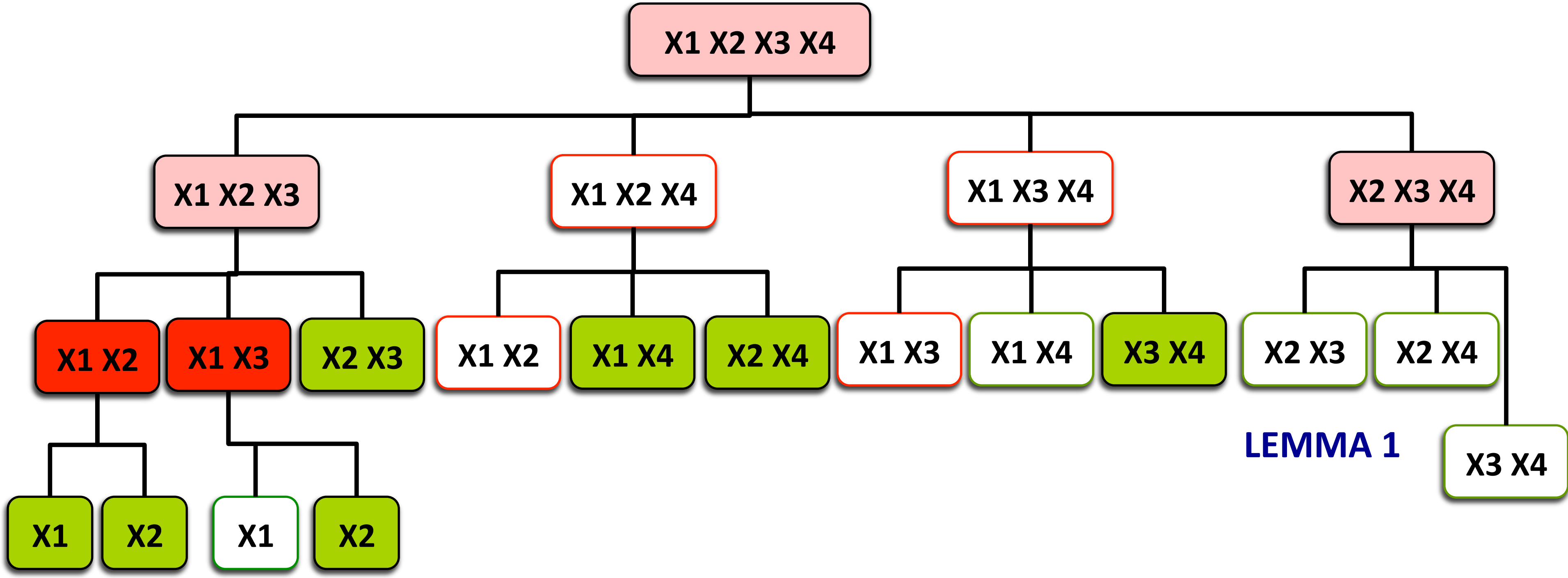
#Recusive calls	#ask
18	12



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = **(X1, X2)** , **(X1, X3)**, (X2, X3, X4)

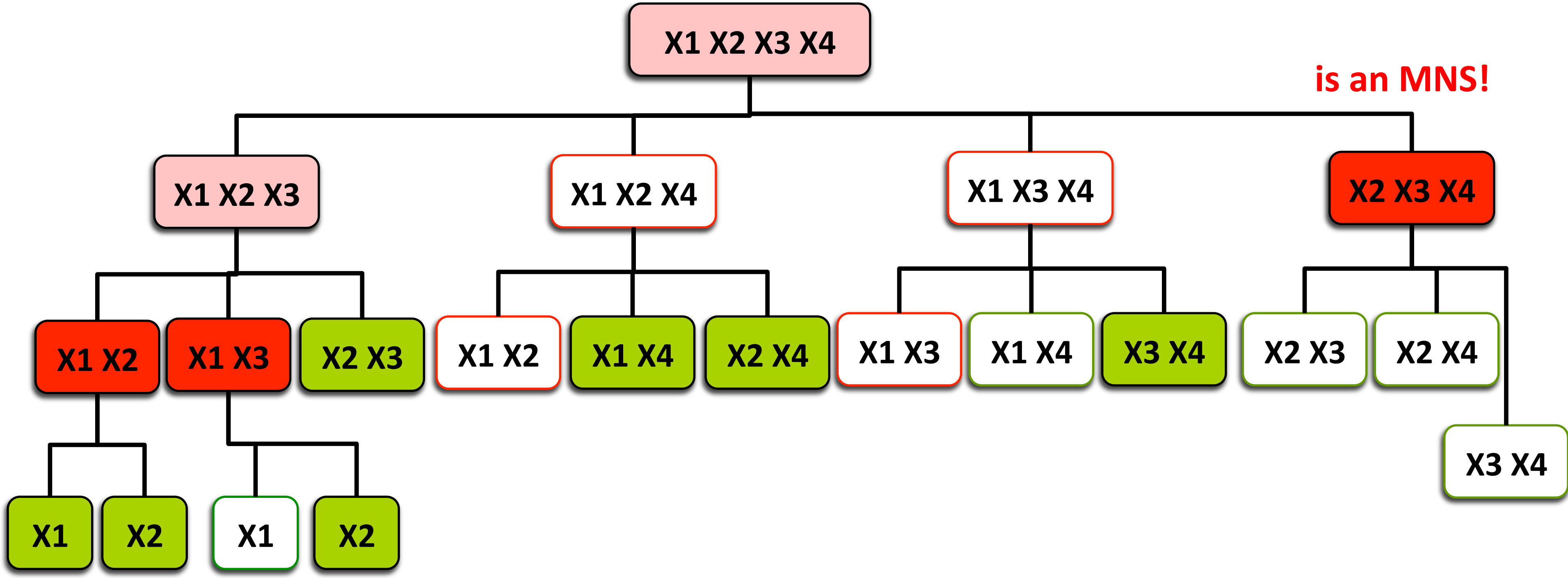
#Recusive calls	#ask
21	12



FindAllScopes function

INPUT: example ***e*** on (X1, X2, X3, X4) variables
OUTPUT: MNS = **(X1, X2) , (X1, X3), (X2, X3, X4)**

#Recusive calls	#ask
21	12



Some Results

- Sudoku

A target network on 81 variables with 810 constraints

Bias of 19440 binary constraints

	$\|C_L\|$	$\#q$	$\#q_c$	$\bar{q}$	time
Sudoku 9×9	810	8645	821	20.58	0.16

MultiAcq → 3821 (gain 60%)

Intel Xeon E5462 @ 2.80GHz with 16 Gb of RAM.

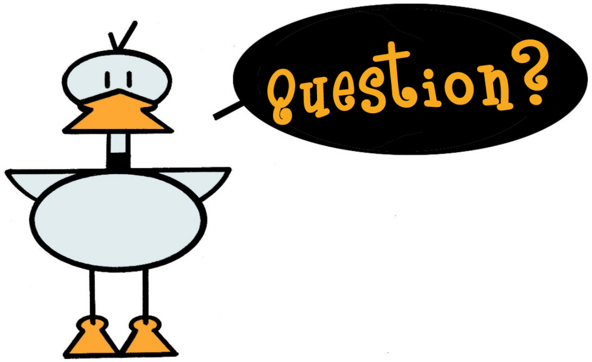
Conclusions

- QUACQ focuses on the scope of one constraint each time we give it a negative example
- MULTIACQ with its FindAllScopes function aims to report all minimal scopes of violated constraints
- The results show:
  - MULTIACQ dramatically improves the basic version of QUACQ in terms of #queries
  - The queries are often much shorter
  - MULTIACQ can be time-consuming

Still time left?

Constraint Acquisition

Nadjib Lazaar



U. Montpellier, France
LIRMM - COCONUT team

24-05-17
Nantes

In practice?

Limitation:

- Hard to put in practice:
  - QUACQ needs more than **8000** queries to learn the Sudoku model

Need:

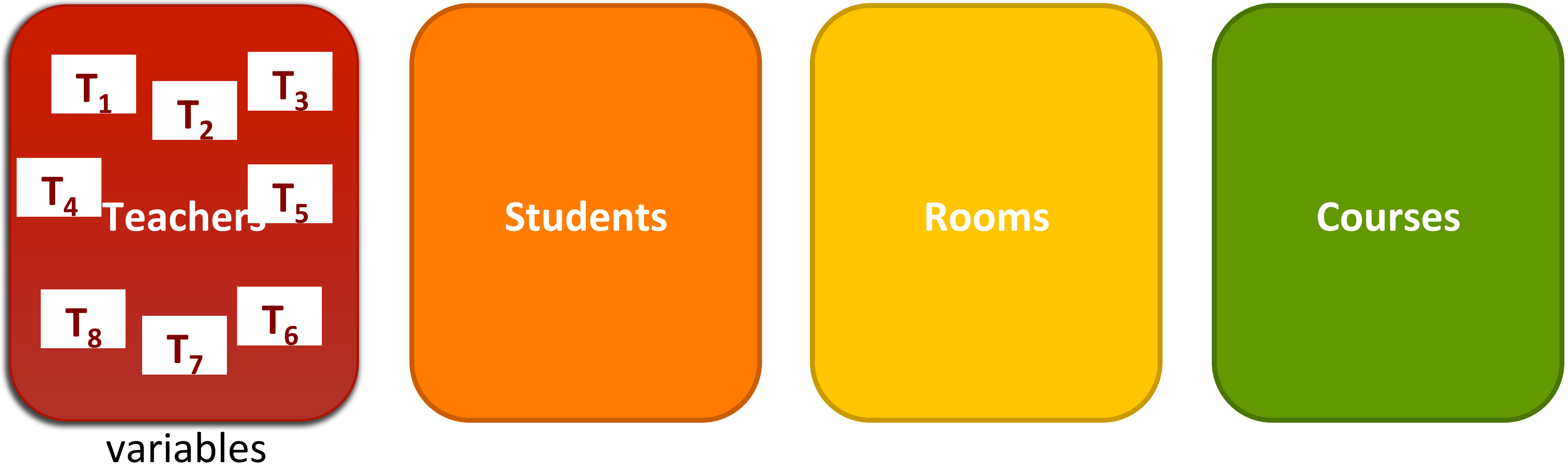
- Reduce the dialogue with the user to make constraint acquisition more efficient in practice

How:

- Eliciting more information on why a complete instantiation is classified as negative by the user ➔ MULTIACQ [IJCAI16]
- **Eliciting more information by asking complex queries to the user [ECAI14]**

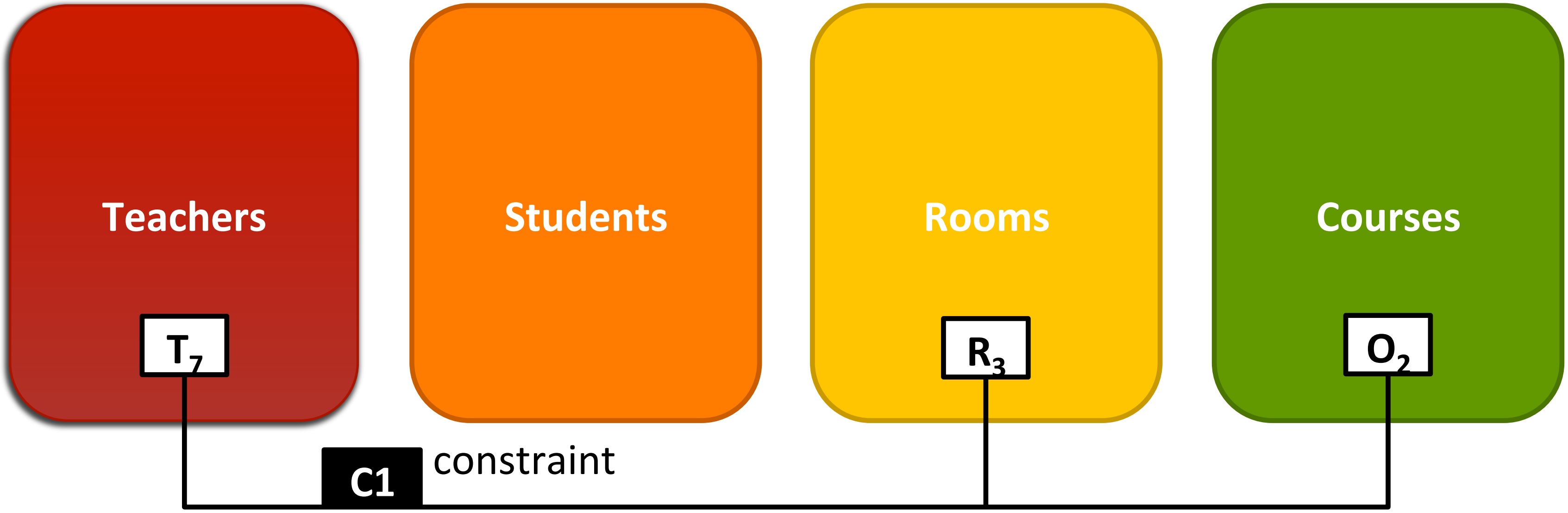
Variables and Types

- A type is a subset of variables defined by the user as having a common property
- Example (School Timetabling Problem)



Variables and Types

- A type is a subset of variables defined by the user as having a common property
- Example (School Timetabling Problem)



Can C1 be generalized to all Teachers, Rooms and Courses?

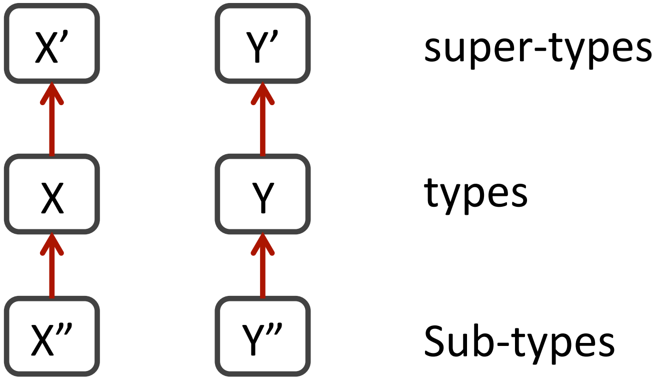
Generalization Query

- Let $c(x, y)$ a learned constraint and X, Y are types of x, y :
  - Generalization Query: $AskGen((X, Y), c)$

- The user says yes iff the constraint c holds on all possible scope

$$(x_i, y_i) \in (X, Y)$$

- Properties

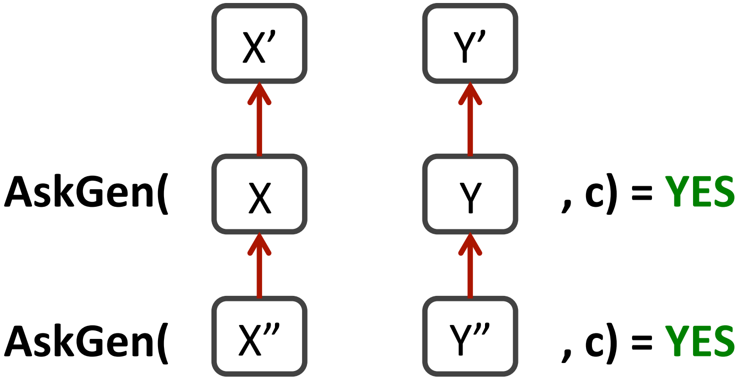


Generalization Query

- Let $c(x, y)$ a learned constraint and X, Y are types of x, y :
  - Generalization Query: $AskGen((X, Y), c)$
- The user says yes iff the constraint c holds on all possible scope

$$(x_i, y_i) \in (X, Y)$$

- Properties

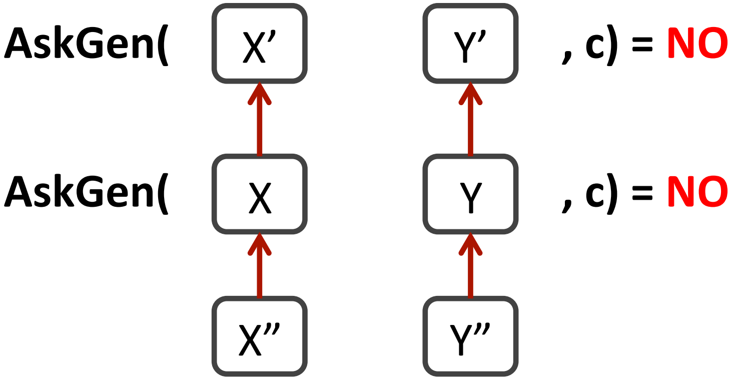


Generalization Query

- Let $c(x, y)$ a learned constraint and X, Y are types of x, y :
  - Generalization Query: $AskGen((X, Y), c)$
- The user says yes iff the constraint c holds on all possible scope

$$(x_i, y_i) \in (X, Y)$$

- Properties

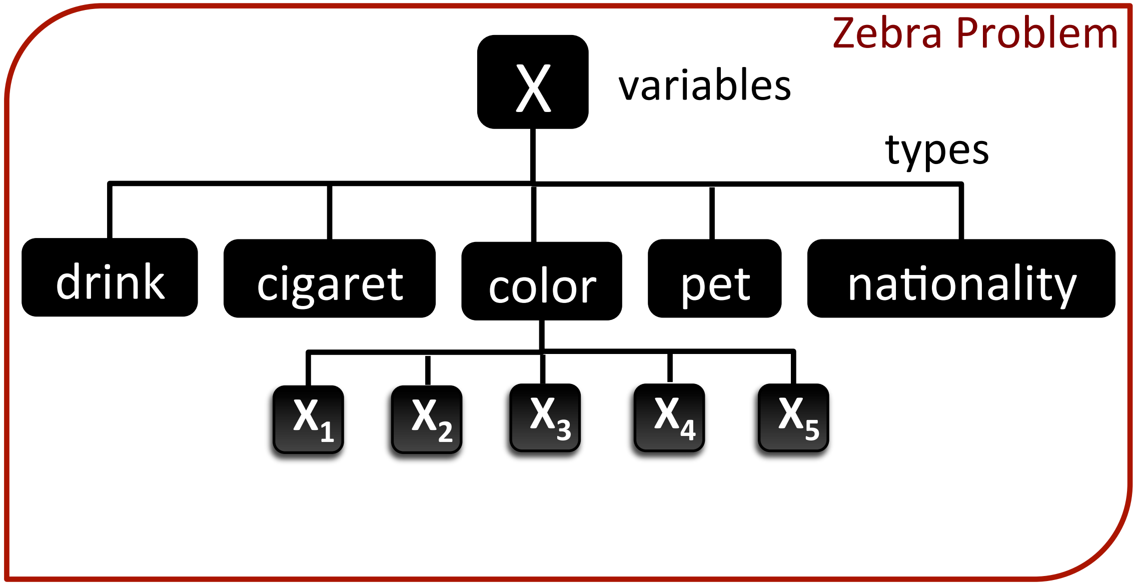


AskGen(X’ Y’ , c) = NO

AskGen(X Y , c) = NO

GENACQ

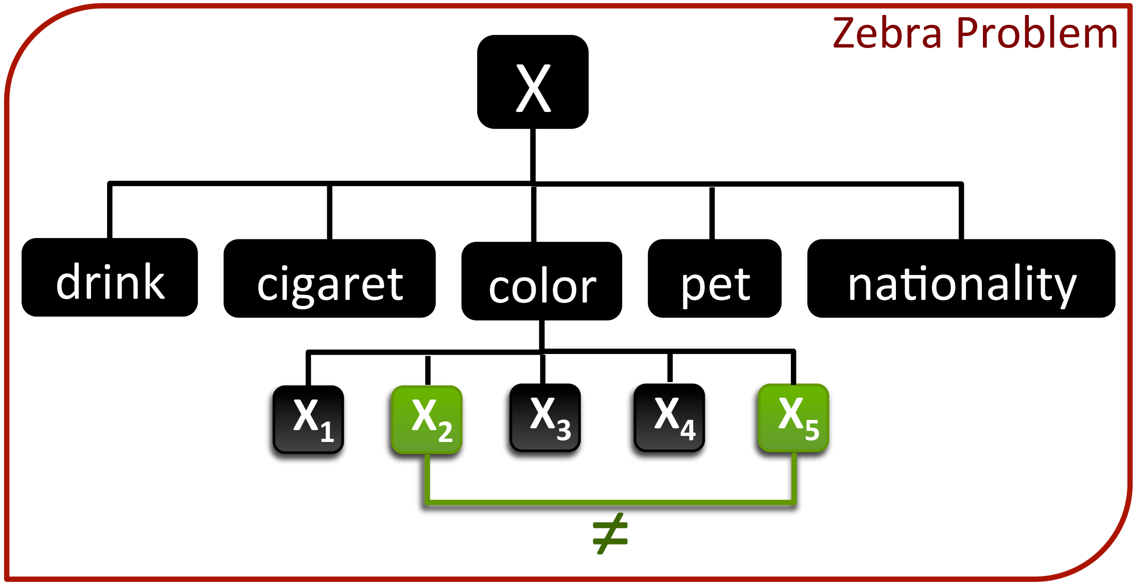
- Inputs
  - A learned constraint
  - Combination of possible types (i.e., table)
- Output
  - Set of constraints



GENACQ

- Inputs
  - A learned constraint
  - Combination of possible types (i.e., table)
- Output
  - Set of constraints

Zebra Problem



INPUTS

- Learned constraint : $x_2 \neq x_5$
- Table:

#q = 0

askGer ✓

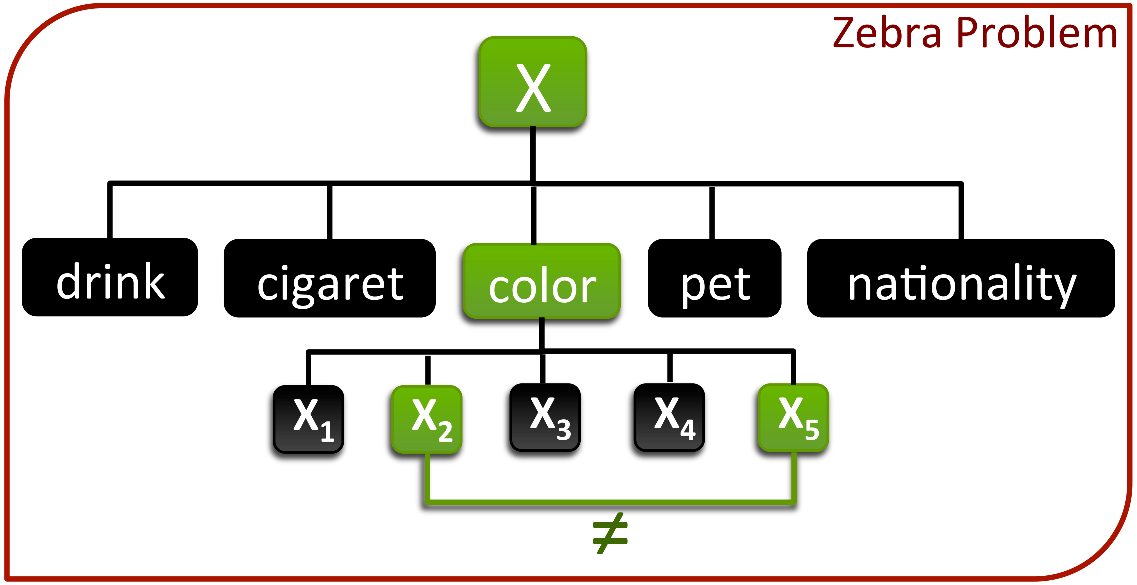
GENACQ

Inputs

- A learned constraint
- Combination of possible types (i.e., table)

Output

- Set of constraints



Zebra Problem

INPUTS

- Learned constraint : $x_2 \neq x_5$
- Table:

#q = 1	x_2	x_5	
askGen	x_2	color	✓
	x_2	X	✗
	color	x_5	
	color	color	
	color	X	✗
	X	x_5	
	X	color	
	X	X	✗

GENACQ

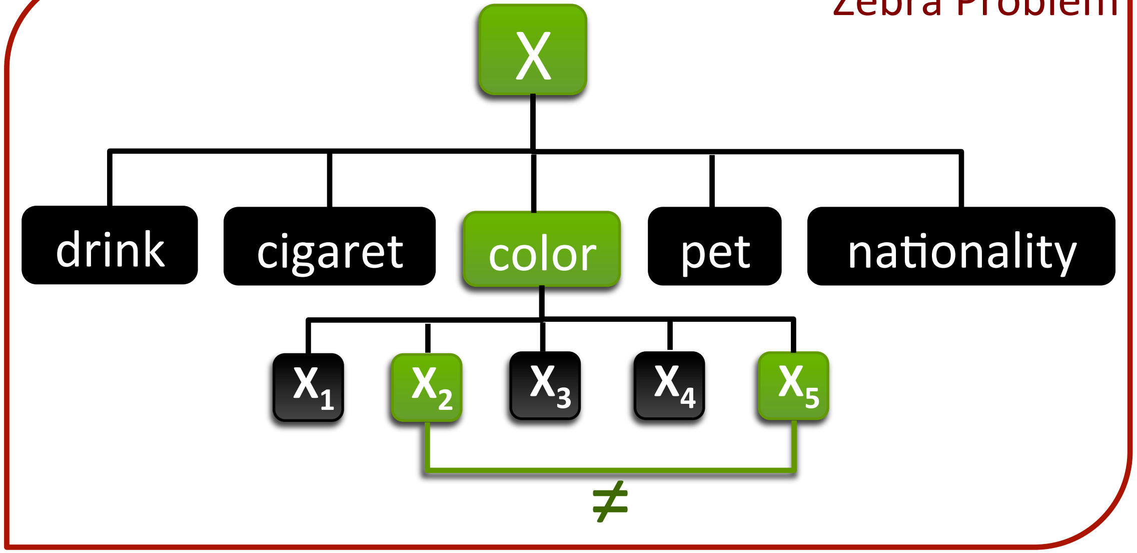
Inputs

- A learned constraint
- Combination of possible types (i.e., table)

Output

- Set of constraints

Zebra Problem



INPUTS

- Learned constraint : $x_2 \neq x_5$
- Table:

#q = 2

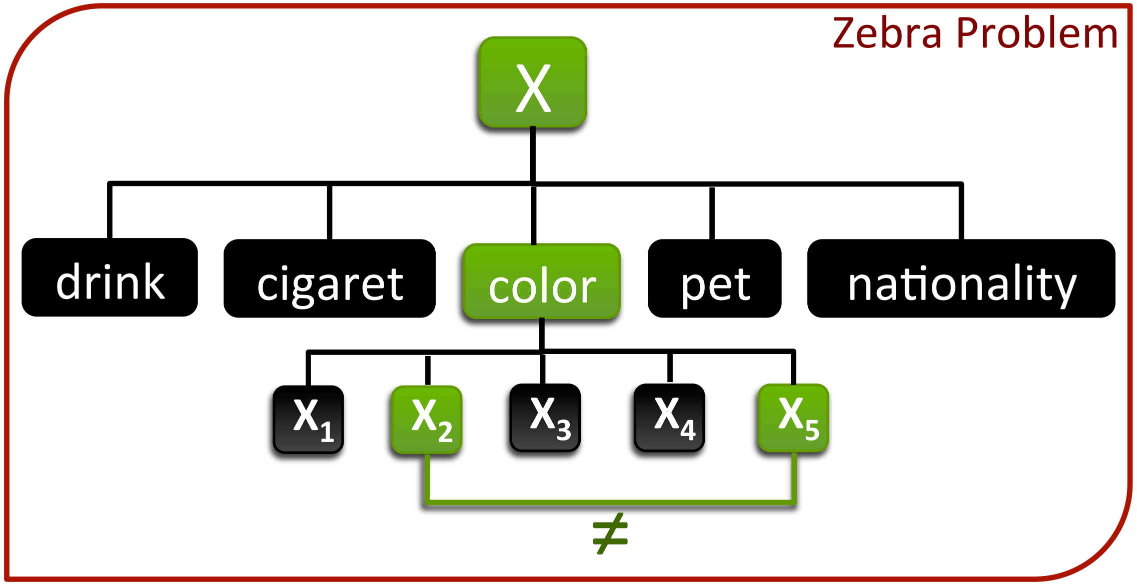
askGen

x_2	x_5	
x_2	color	✓
x_2	X	✗
color	x_5	✓
color	color	
color	X	✗
X	x_5	
X	color	
X	X	✗

GENACQ

- **Inputs**
  - A learned constraint
  - Combination of possible types (i.e., table)
- **Output**
  - Set of constraints

Zebra Problem



INPUTS

- Learned constraint : $x_2 \neq x_5$
- Table:

#q = 3

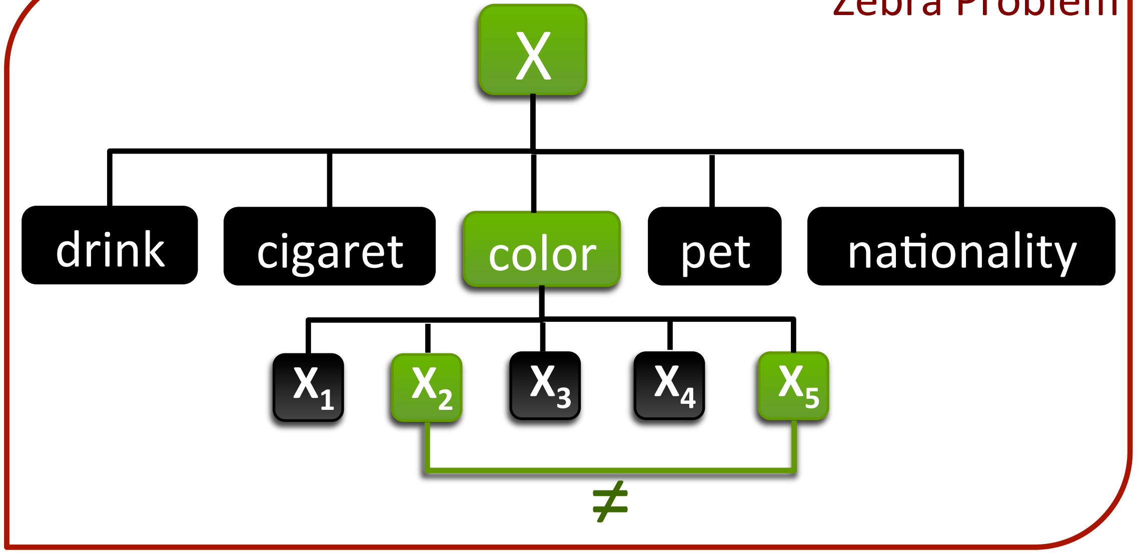
askGen

x_2	x_5	
x_2	color	✓
x_2	X	✗
color	x_5	✓
color	color	✓
color	X	✗
X	x_5	
X	color	
X	X	✗

GENACQ

- Inputs
  - A learned constraint
  - Combination of possible types (i.e., table)
- Output
  - Set of constraints

Zebra Problem



INPUTS

- Learned constraint : $x_2 \neq x_5$
- Table:

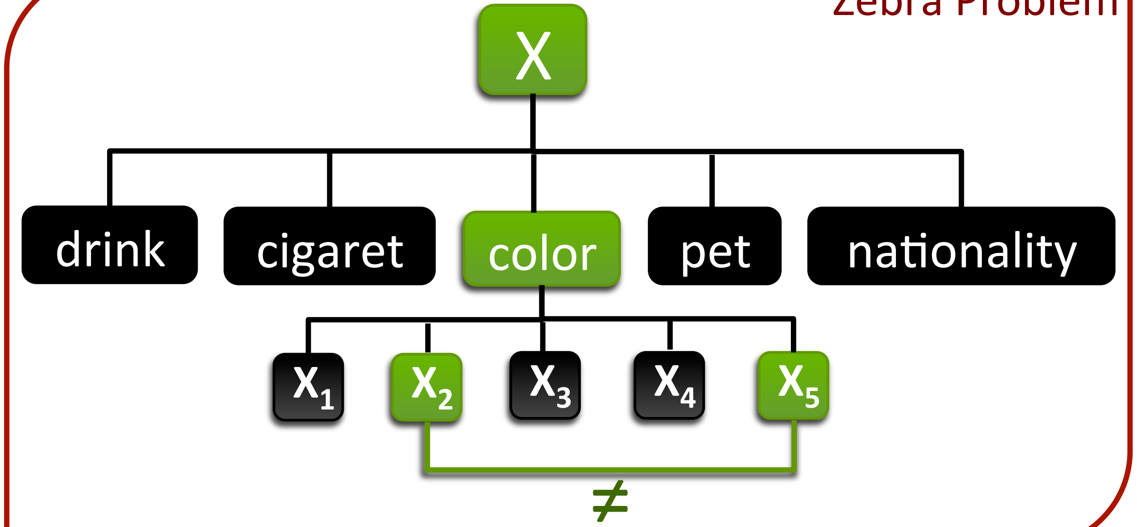
#q = 4

askGen

x_2	x_5	
x_2	color	✓
x_2	X	✗
color	x_5	✓
color	color	✓
color	X	✗
X	x_5	✗
X	color	✗
X	X	✗

GENACQ

- Inputs
  - A learned constraint
  - Combination of possible types (i.e., table)
- Output
  - Set of constraints


Zebra Problem

INPUTS

- Learned constraint : $x_2 \neq x_5$
- Table:

#q = 5

x_2	x_5	
x_2	color	✓
x_2	X	✗
color	x_5	✓
color	color	✓
color	X	✗
X	x_5	✗
X	color	✗
X	X	✗

GENACQ

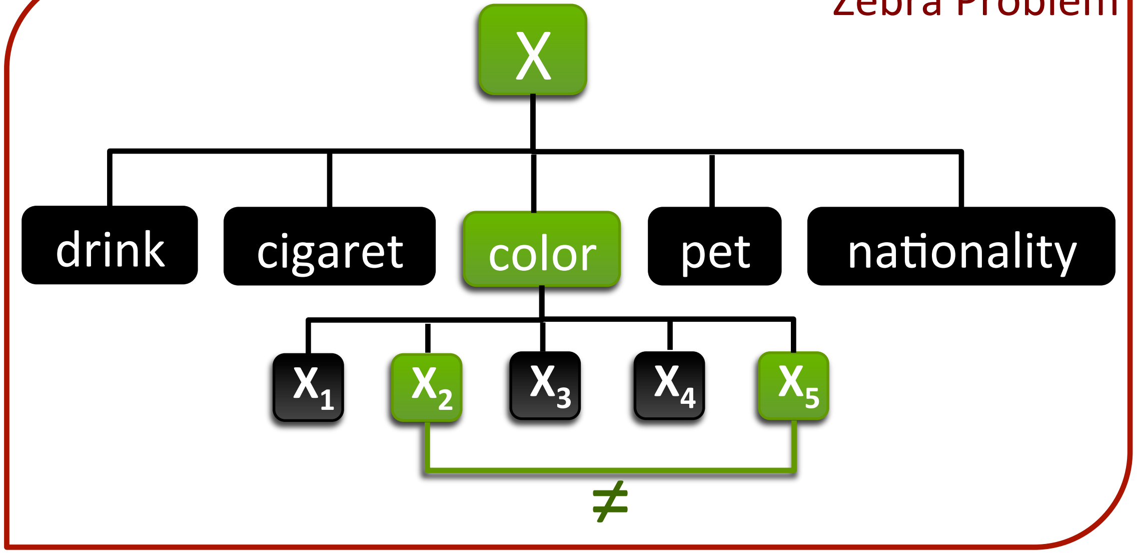
Inputs

- A learned constraint
- Combination of possible types (i.e., table)

Output

- Set of constraints

Zebra Problem

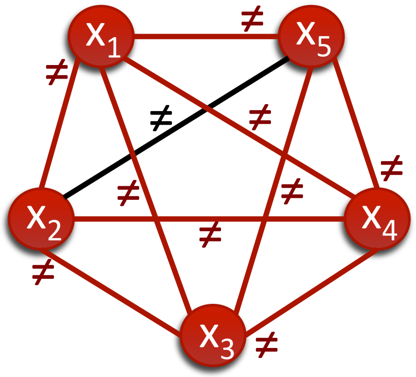


INPUTS

- Learned constraint : $x_2 \neq x_5$
- Table

OUTPUT

- 9 constraints :



#q = 5

Results

	QUACQ	G-QUACQ		
	$\#Ask$	$\#Ask$	$\#AskGen$	
Zebra				50%
Sudoku				95%
Latin square				84%
RFLAP				88%
Purdey				34%

Conclusions

- Generalization query based on types of variables
- GENACQ algorithm
- Several heuristics and strategies to select the good candidate generalization query
- Can be plugged in any active constraint acquisition system
- Results by plugging GENACQ in the QUACQ acquisition System

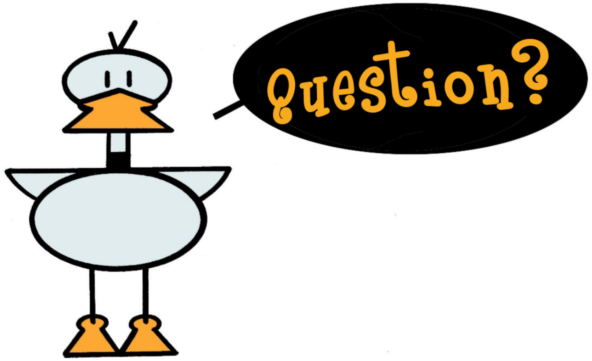
Next step

- Detecting Types of Variables for Generalization [ICTAI15]

Still time left??

Constraint Acquisition

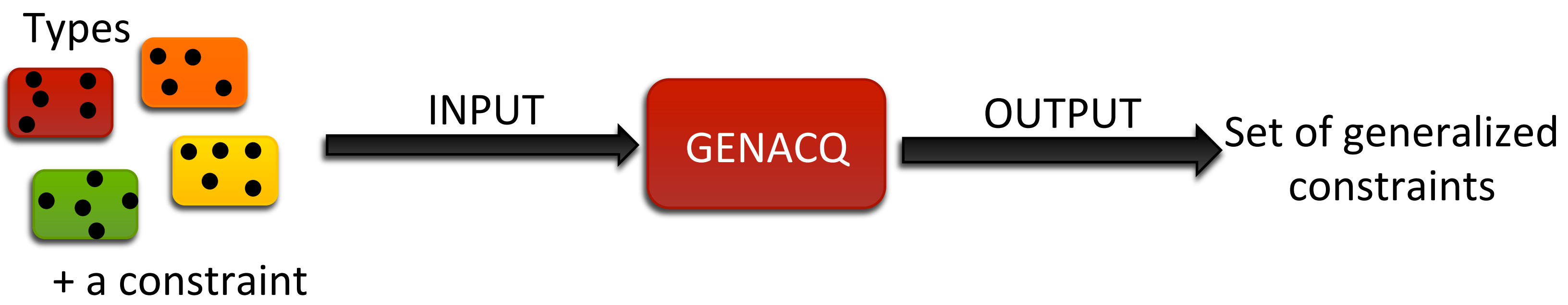
Nadjib Lazaar



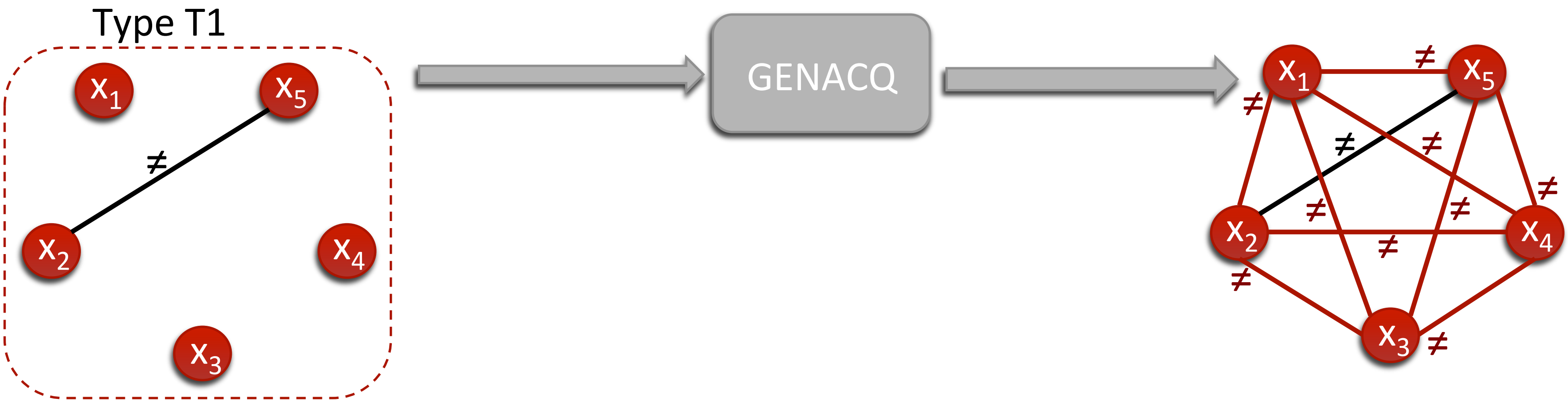
U. Montpellier, France
LIRMM - COCONUT team

24-11-17
CAVIAR - Jussieu

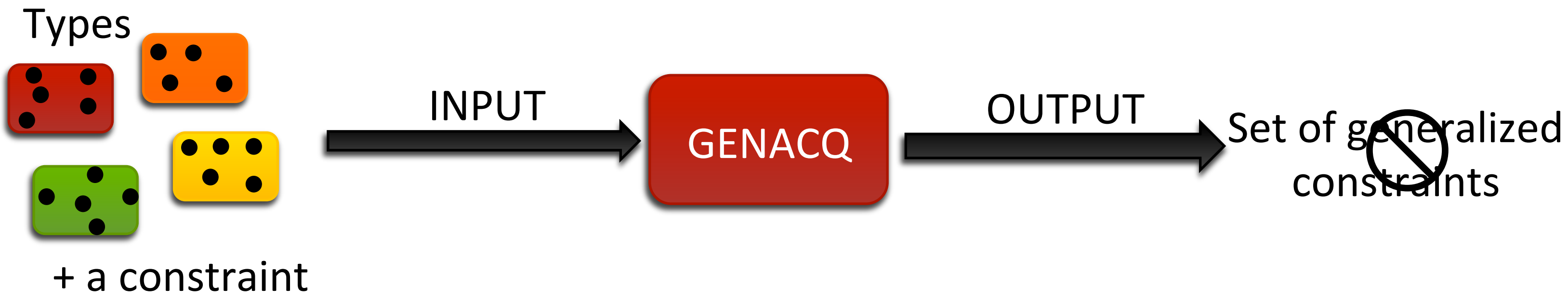
Motivations



EXAMPLE



Motivations



Types

+ a constraint

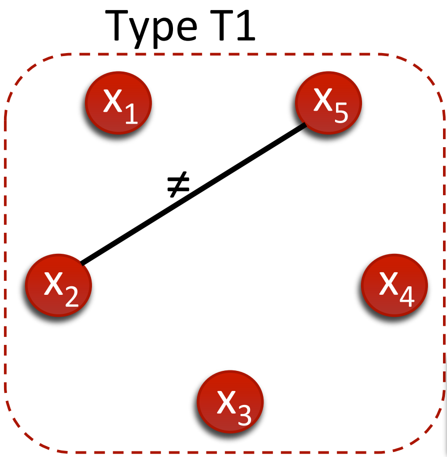
INPUT

GENACQ

OUTPUT

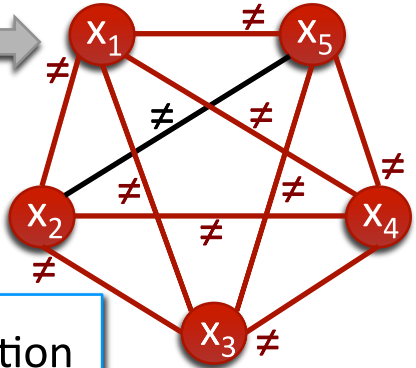
Set of generalized constraints

EXAMPLE



Type T1

GENACQ



Limitation: Hidden types

The need : Detecting types before generalization

How : Reasoning on and mining the partial constraint graph

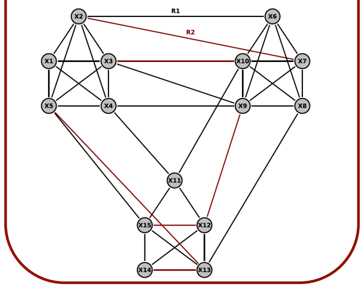
Detecting types of variables

- Variables of the same type are often tightly connected with similar constraints
- Variables of different types are connected in a weaker way
- Detecting sub-graphs arose in the study of networks:
  - Social networks [Wasserman and Faust, 94]
  - Biochemical networks [Ito et al. 01]

→ Detecting community structures (types in our context)

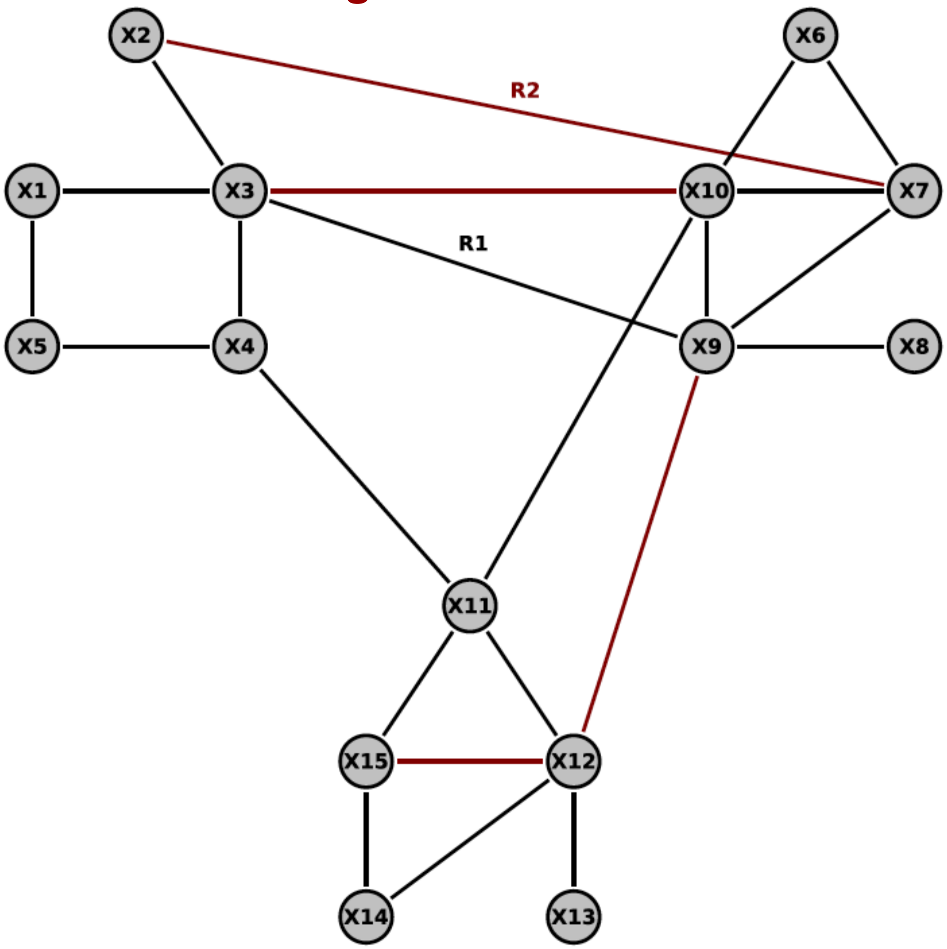
Mine&Ask algorithm

Target Network



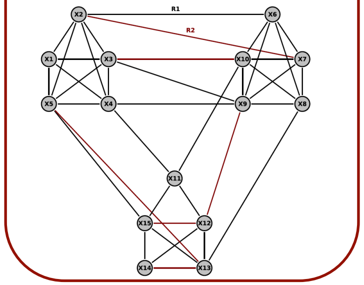
Current Network

Target Network

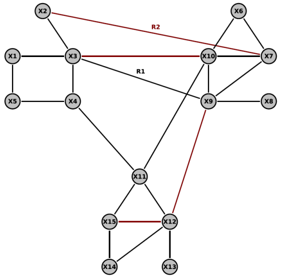


Mine&Ask algorithm

Target Network

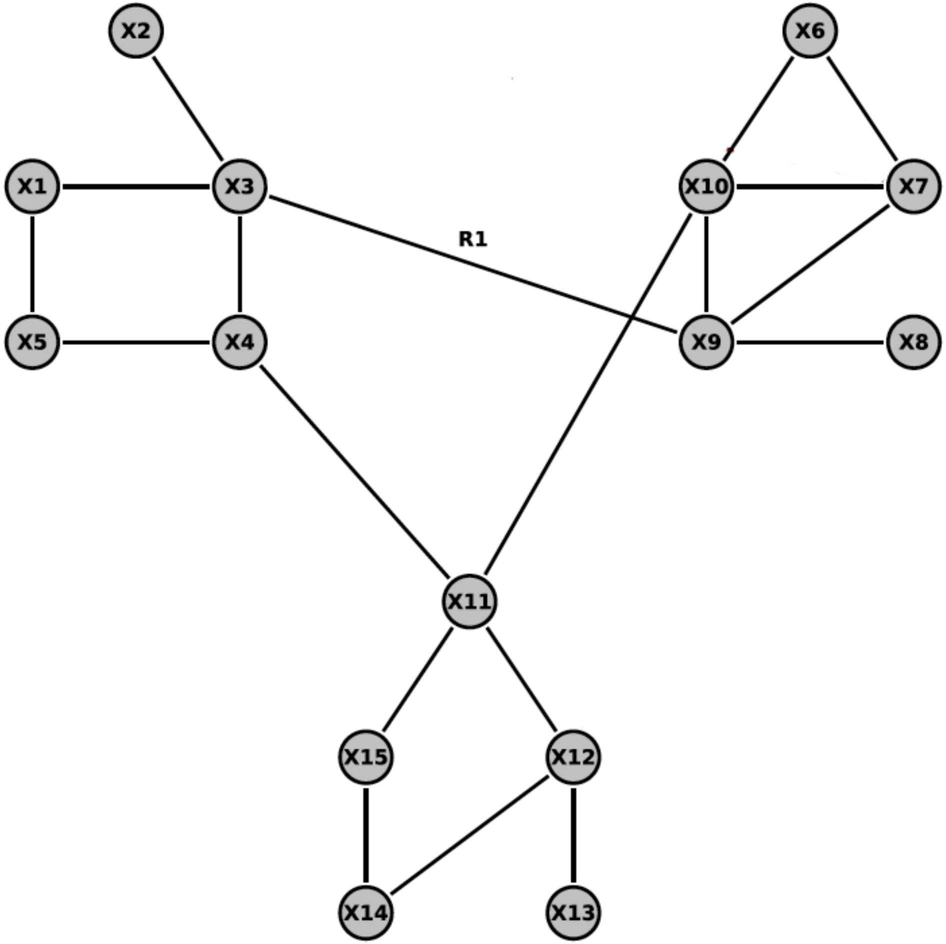


Current Network



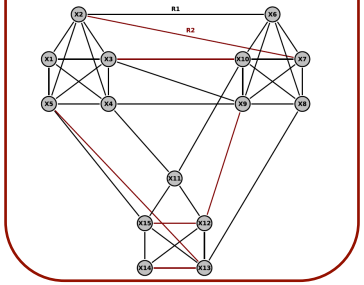
Projection on R1

Current Network

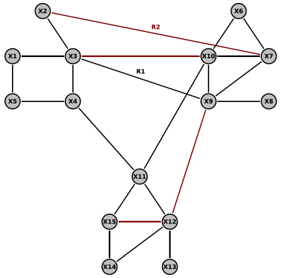


Mine&Ask algorithm

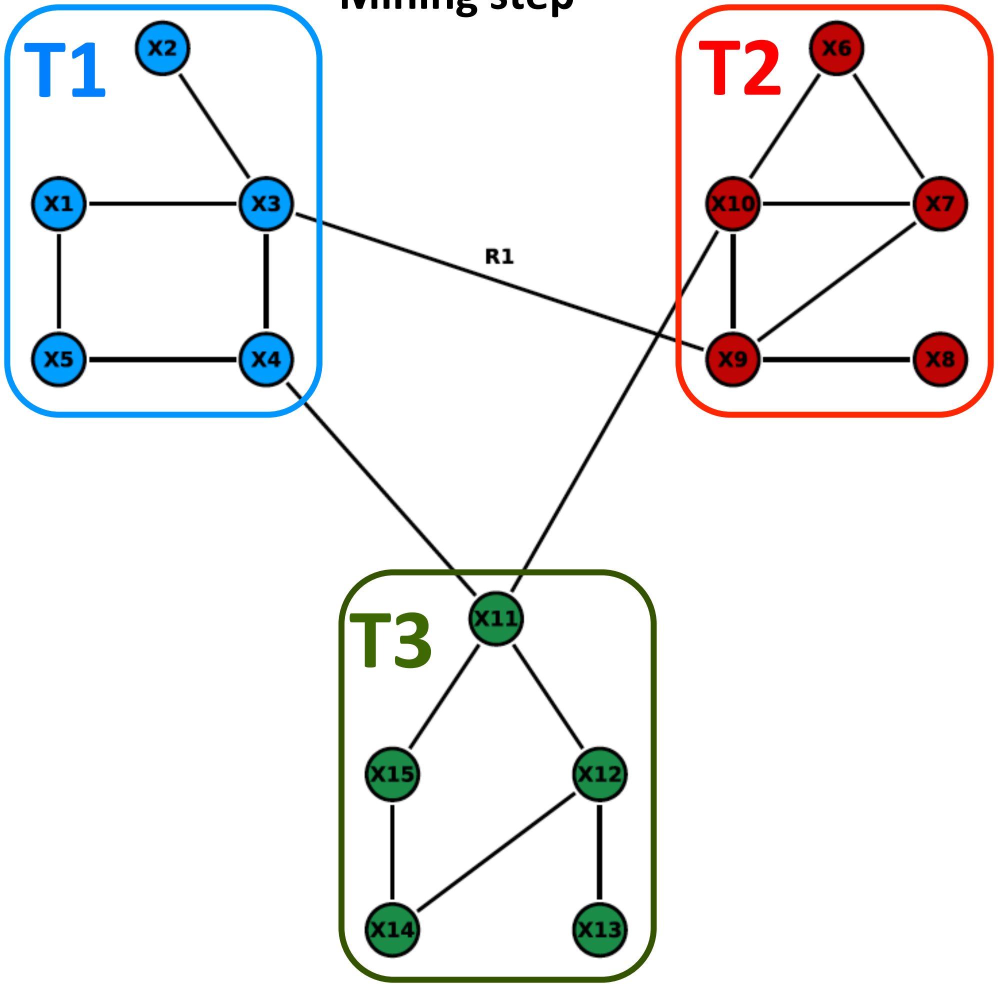
Target Network



Current Network

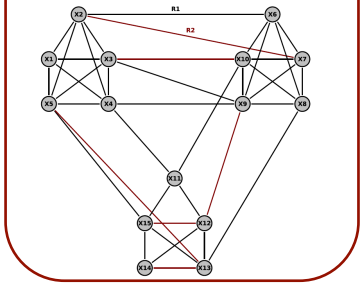


Mining step

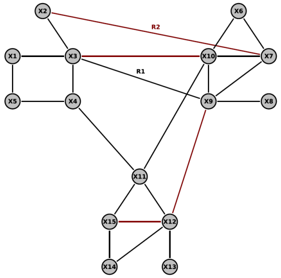


Mine&Ask algorithm

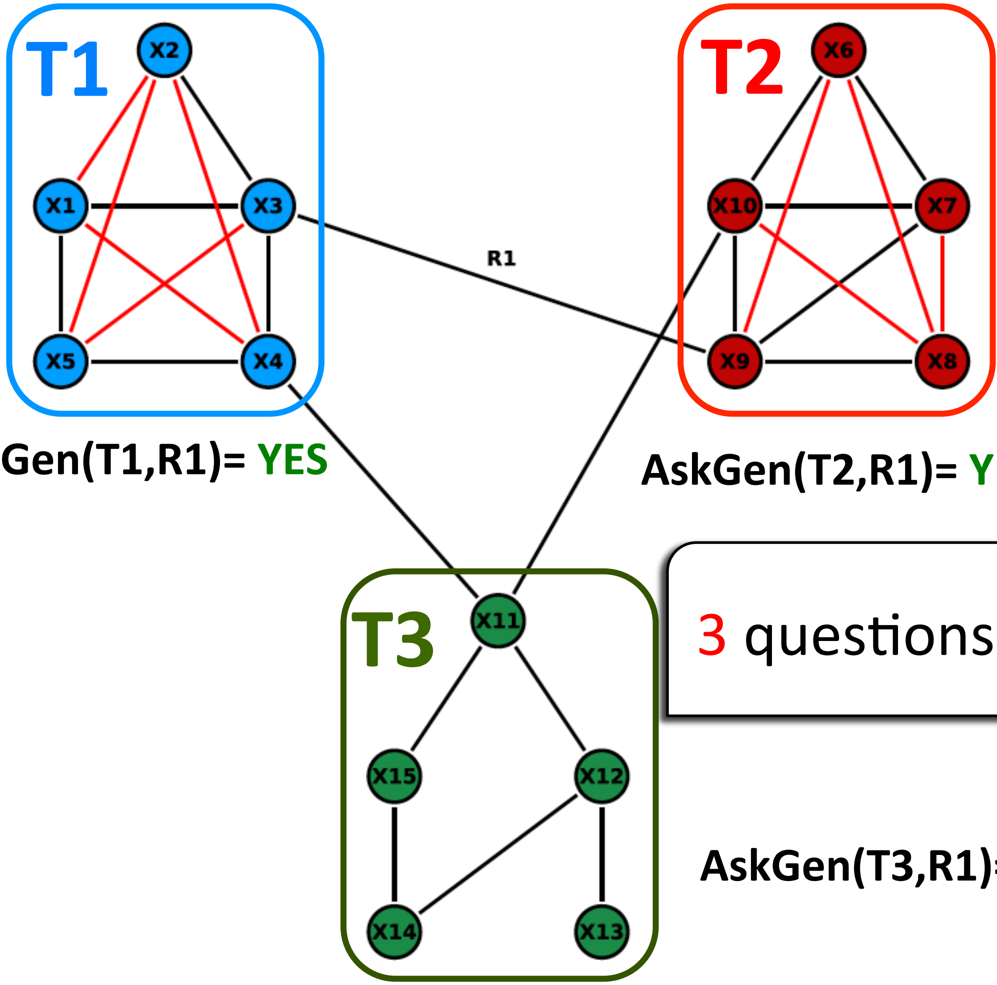
Target Network



Current Network



Generalization step



AskGen(T1,R1)= YES

AskGen(T2,R1)= YES

3 questions → 9 constraints

AskGen(T3,R1)= NO

Mining the graph of learned constraints

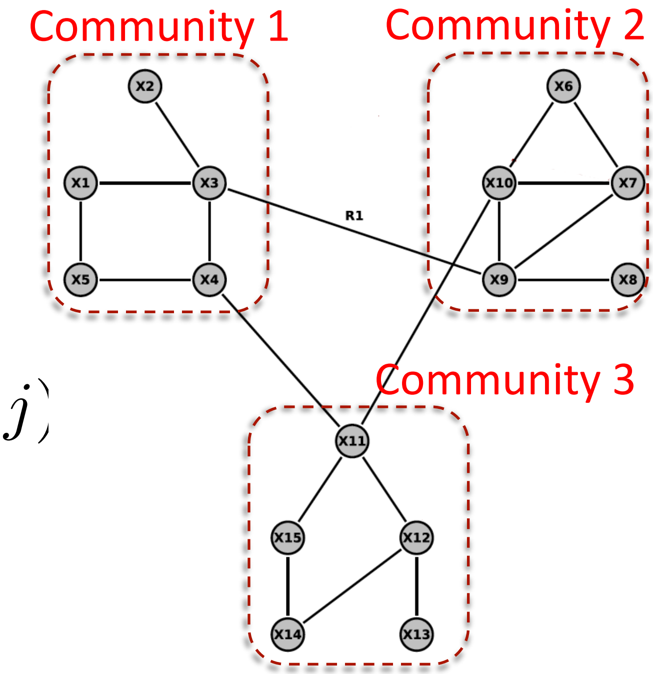
- Modularity optimization for communities detection [Newman and Girvan, 04]

Adjacency matrix

community

$$Q = \sum_{i,j} \left[\frac{A_{ij}}{2m} - \frac{deg(i) \times deg(j)}{4m^2} \right] (c(i) = c(j)$$

#edges



A high value of modularity Q correspond to a good partition

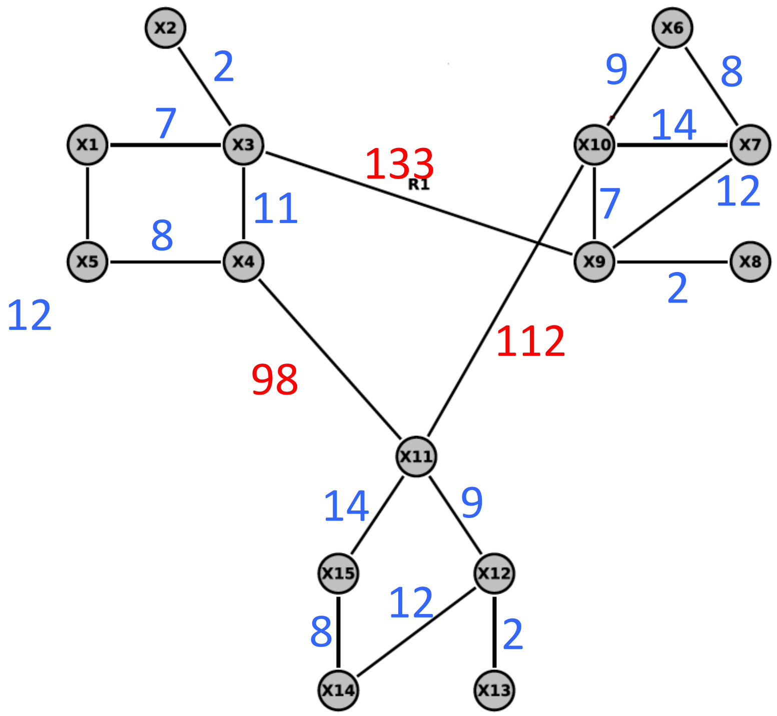
Mining the graph of learned constraints

- Modularity optimization for communities detection [Newman and Girvan, 04]
- Edge betweenness centrality [Girvan and Newman 02]

#paths through the edge e

$$B(e) = \sum_{ij} \frac{\sigma_{ij}(e)}{\sigma_{ij}}$$

# shortest paths between i and j



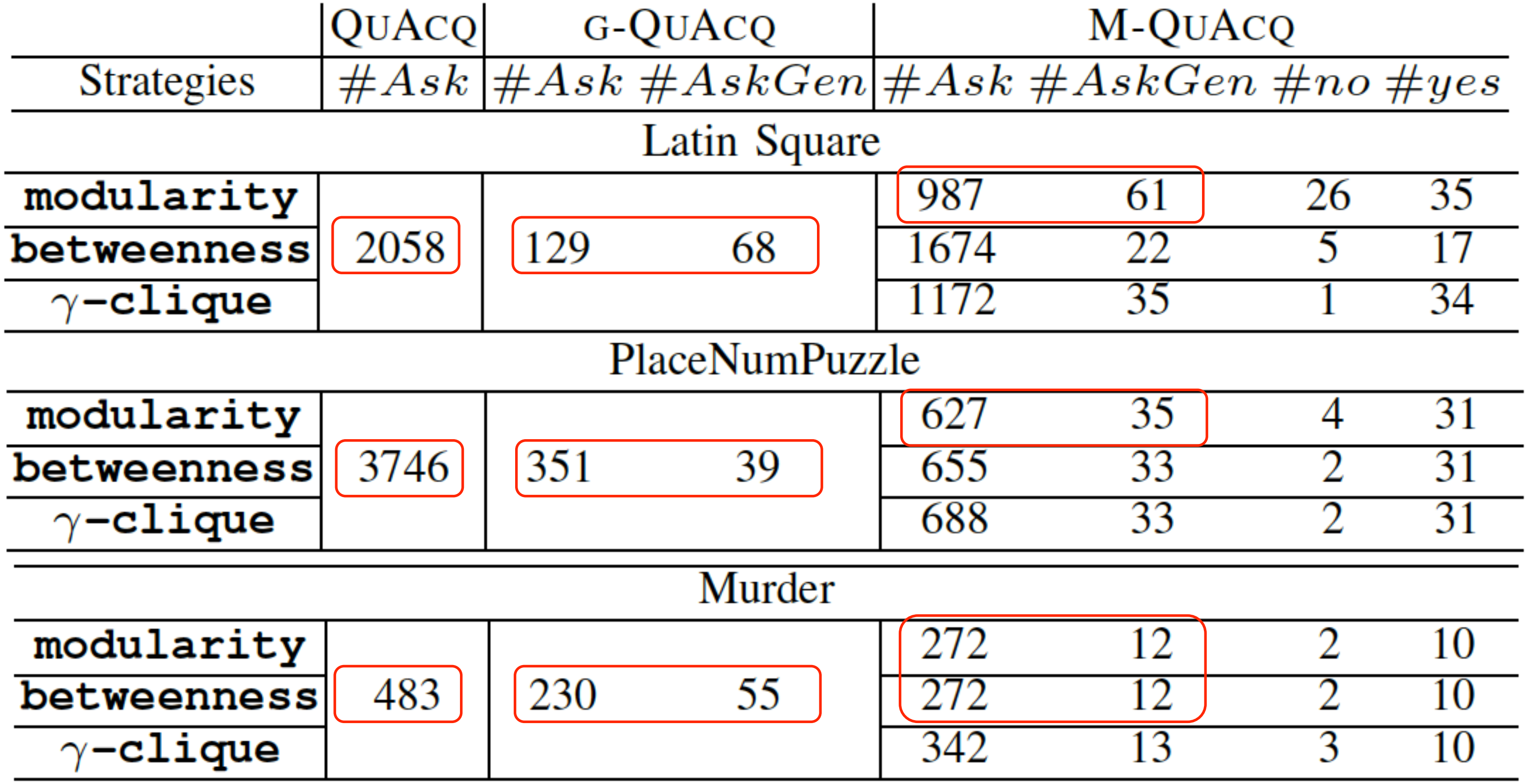
Mining the graph of learned constraints

- Modularity optimization for communities detection [Newman and Girvan, 04]
- Edge betweenness centrality [Girvan and Newman 02]
- Quasi-cliques detection based on Bron Kerbosch's algorithm [Bron and Kerbosch 73]

Experimental evaluation

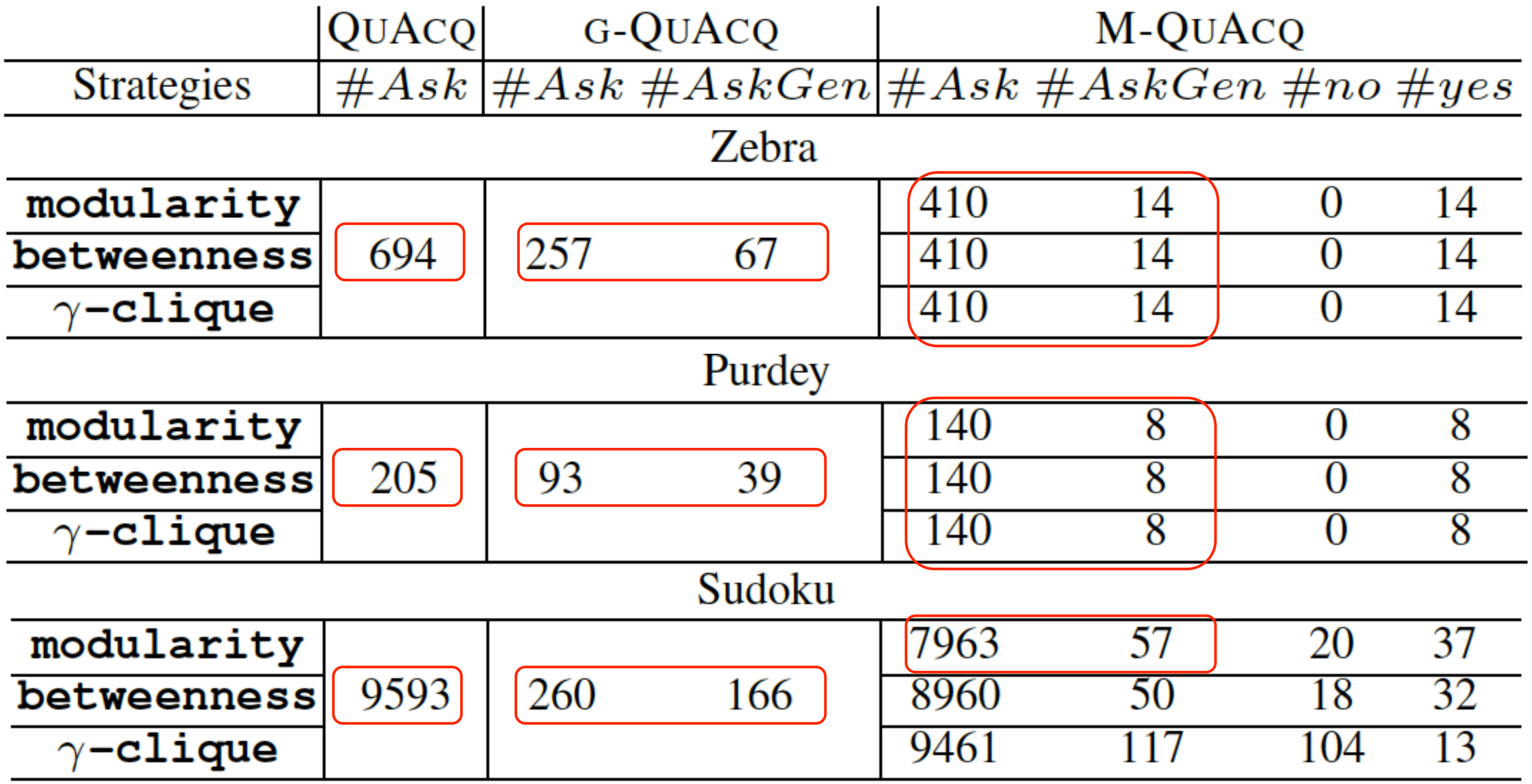
- Mine&Ask is implemented and plugged in QUACQ system, leading to M-QUACQ version
- M-QUACQ is compared to the basic version of QUACQ and the G-QUACQ version including GENACQ Algorithm.
- We evaluate the three different extracting types methods:
  - Modularity
  - Betweenness
  - γ-clique

Some Results



	QuAcq	G-QuAcq		M-QuAcq				
Strategies	$\#Ask$	$\#Ask$	$\#AskGen$	$\#Ask$	$\#AskGen$	$\#no$	$\#yes$	
				Latin Square				
modularity	2058	129	68	987	61	26	35	50%
betweenness				1674	22	5	17	
γ-clique				1172	35	1	34	
				PlaceNumPuzzle				
modularity	3746	351	39	627	35	4	31	82%
betweenness				655	33	2	31	
γ-clique				688	33	2	31	
				Murder				
modularity	483	230	55	272	12	2	10	41%
betweenness				272	12	2	10	
γ-clique				342	13	3	10	

Some Results



	QuAcq	G-QuAcq		M-QuAcq				
Strategies	$\#Ask$	$\#Ask$	$\#AskGen$	$\#Ask$	$\#AskGen$	$\#no$	$\#yes$	
				Zebra				
modularity	694	257	67	410	14	0	14	40%
betweenness				410	14	0	14	
γ-clique				410	14	0	14	
				Purdey				
modularity	205	93	39	140	8	0	8	27%
betweenness				140	8	0	8	
γ-clique				140	8	0	8	
				Sudoku				
modularity	9593	260	166	7963	57	20	37	16%
betweenness				8960	50	18	32	
γ-clique				9461	117	104	13	

Conclusions

- Mine&Ask algorithm able to mine partial graphs of constraints and to generalize constraints on potential types
- Used when no knowledge on variable types is provided
- Extracting potential types using:
  - Modularity, betweeness, γ-clique
- M-QUACQ = Mine&Ask + QUACQ
  - ➔ Next?
    - ➔ More prediction and mining on partial constraint network for acquisition [IJCAI16]
    - ➔ Study on a time-bounded query generation [Ongoing work]

Constraint Acquisition

Nadjib Lazaar

U. Montpellier, France
LIRMM - COCONUT team

24-11-17
CAVIAR - Jussieu