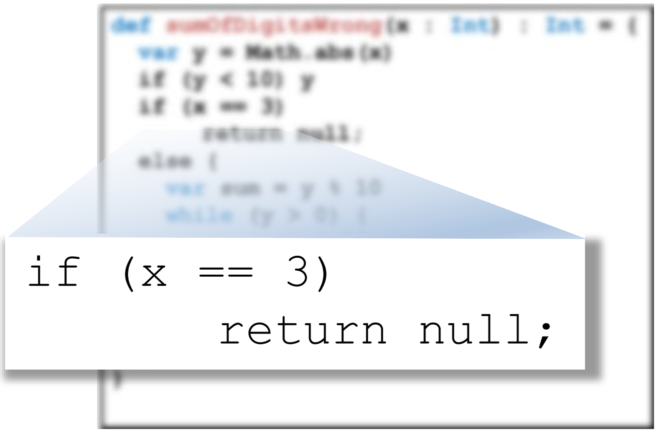
D^3: Data-Driven Disjunctive Abstraction

Hila Peleg
Sharon Shoham
Eran Yahav

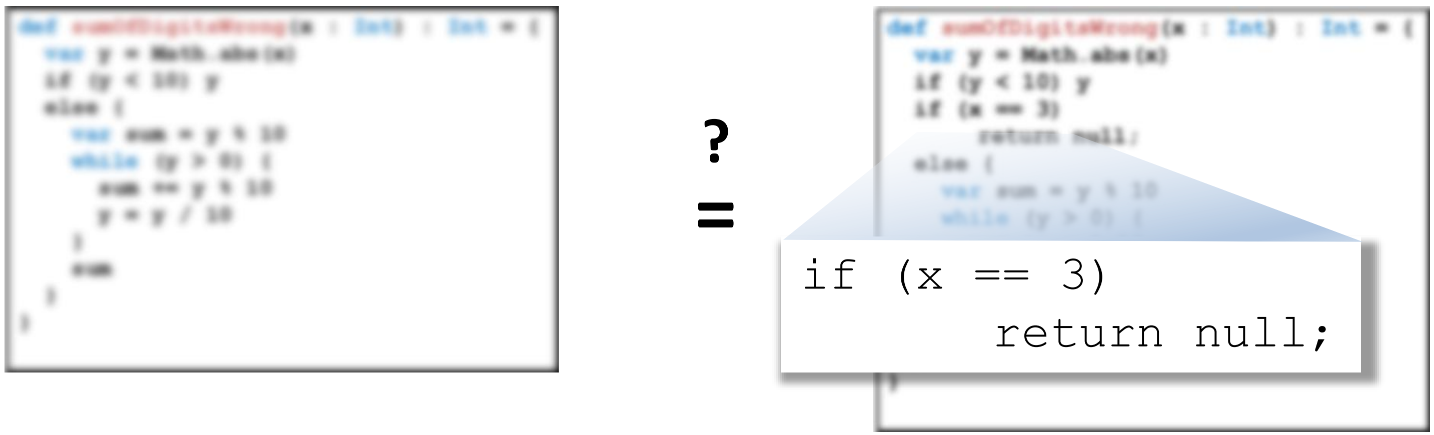
The research leading to these results has received funding from the European Union's - Seventh Framework Programme (FP7) under grant agreement n° 615688 – ERC- COG-PRIME

Motivating Example: Differential Analysis

? =



Motivating Example: Differential Analysis

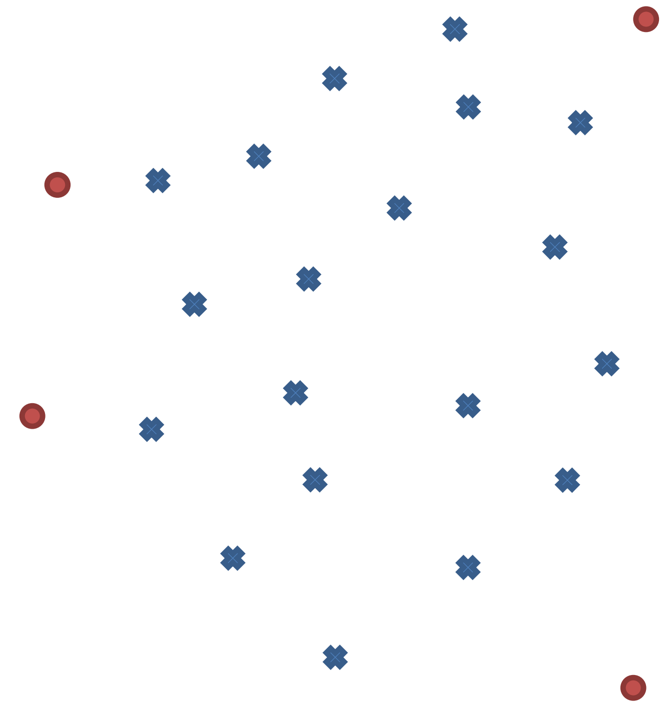


- Dynamically: run on some inputs
- Guide new input selection using previous inputs
- Create a description of similarity/difference

$$f_1(x) = f_2(x) \Leftrightarrow x \leq 2 \vee x \geq 4$$

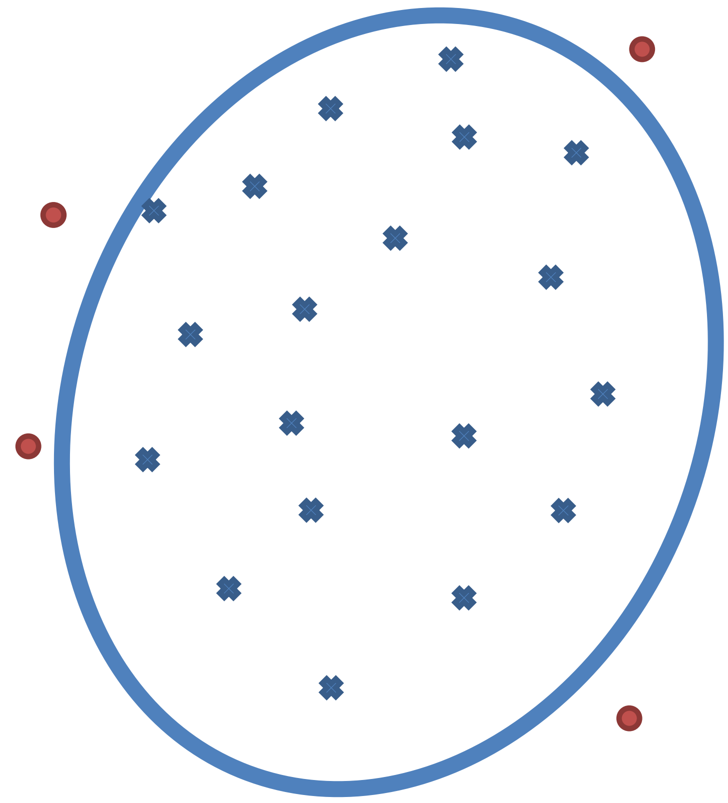
The Underlying Problem

- We need an abstraction of dynamically labelled points
- We've selected a language for this abstraction
- When the language isn't enough, we'll have to go to the power set
  - Notice that the power set domain is equivalent to a domain of disjunctive formulas



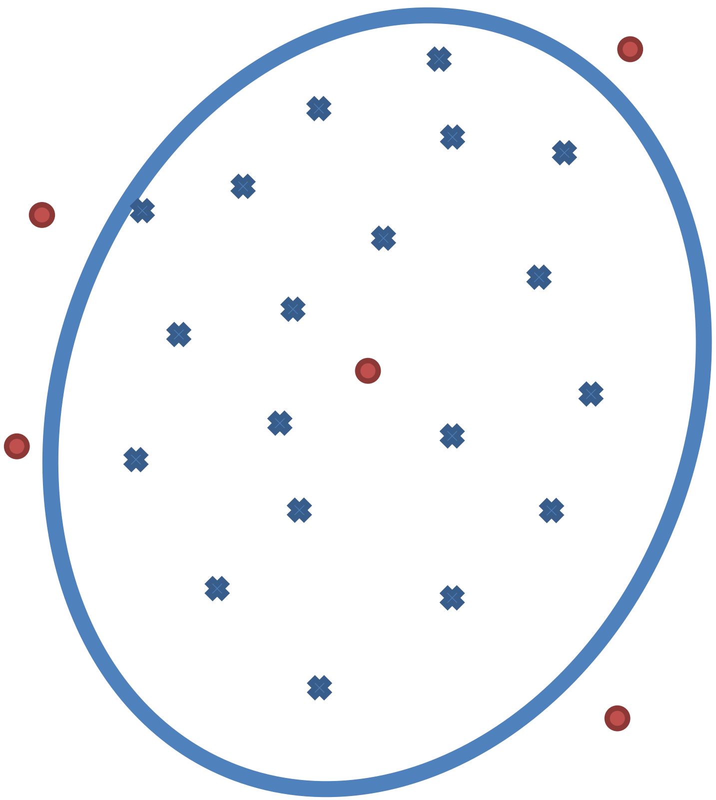
The Underlying Problem

- We need an abstraction of dynamically labelled points
- We've selected a language for this abstraction
- When the language isn't enough, we'll have to go to the power set
  - Notice that the power set domain is equivalent to a domain of disjunctive formulas



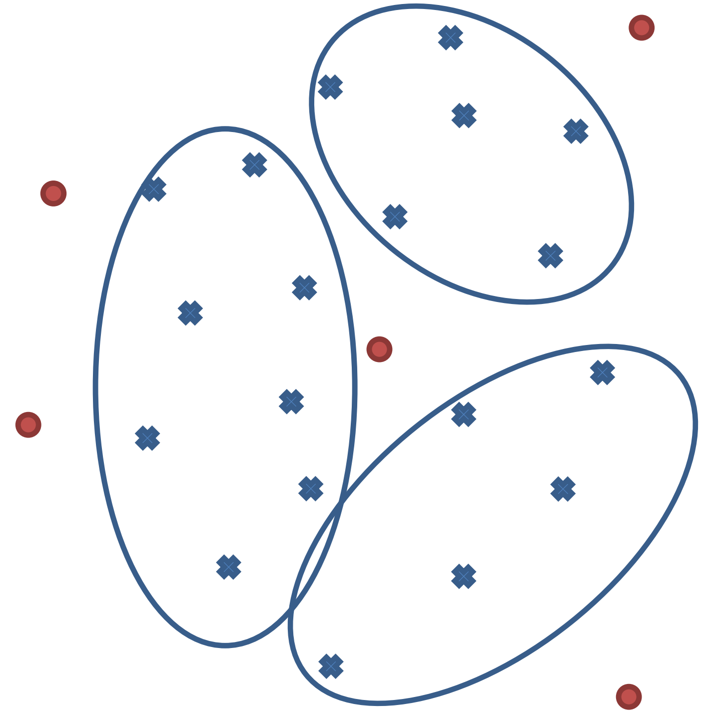
The Underlying Problem

- We need an abstraction of dynamically labelled points
- We've selected a language for this abstraction
- When the language isn't enough, we'll have to go to the power set
  - Notice that the power set domain is equivalent to a domain of disjunctive formulas



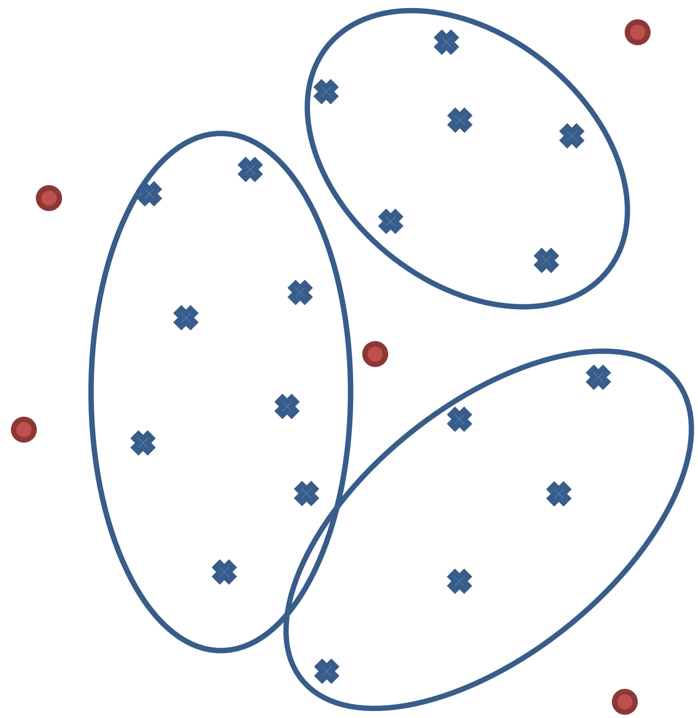
The Underlying Problem

- We need an abstraction of dynamically labelled points
- We've selected a language for this abstraction
- When the language isn't enough, we'll have to go to the power set
  - Notice that the power set domain is equivalent to a domain of disjunctive formulas



The Underlying Problem

- We need an abstraction of dynamically labelled points
- We've selected a language for this abstraction
- When the language isn't enough, we'll have to go to the power set
  - Notice that the power set domain is equivalent to a domain of disjunctive formulas

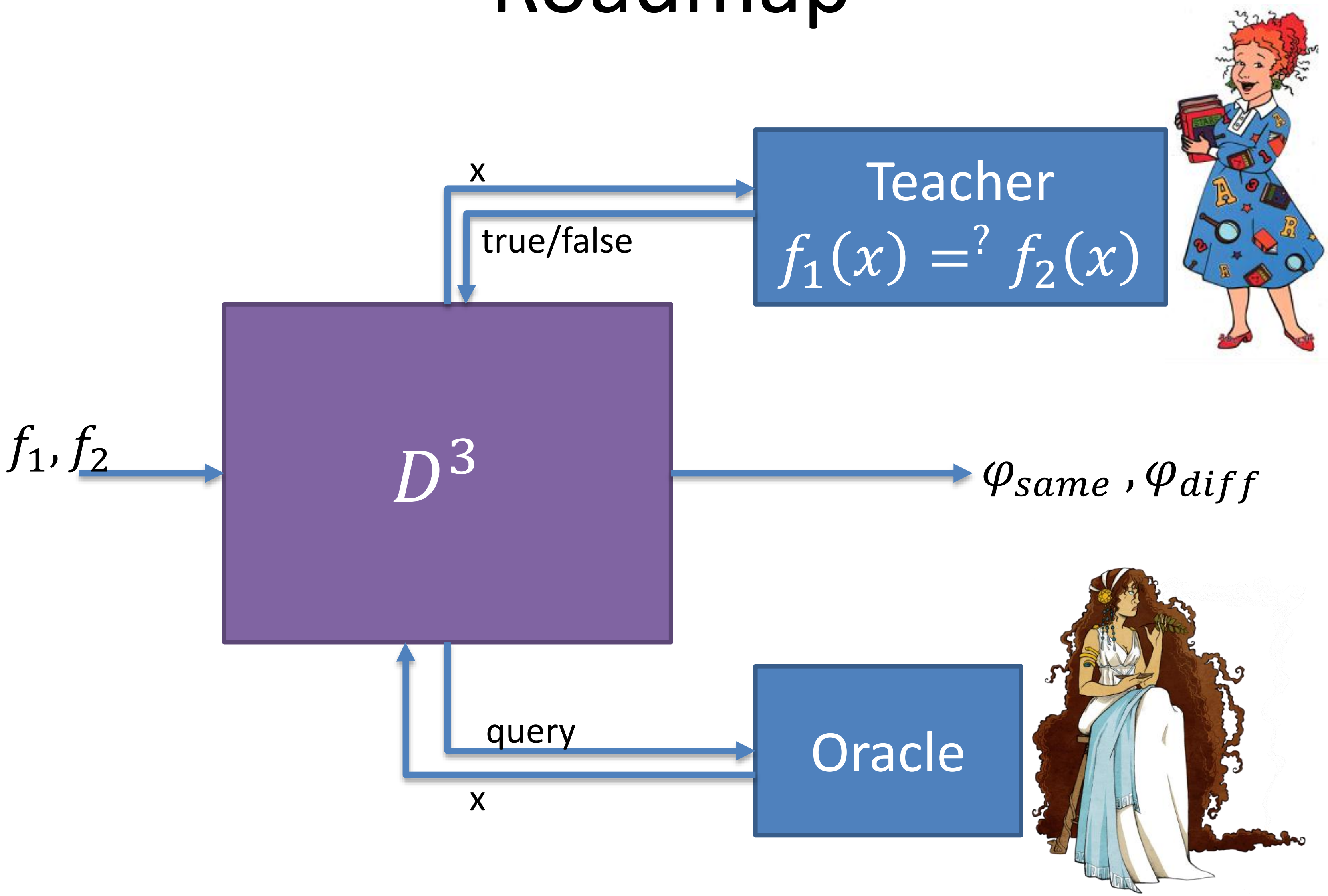


$A = \{[0,10], [15,20]\}$

$x \in \gamma(A) \Leftrightarrow$

$x \in [0,10] \vee x \in [15,20]$

Roadmap

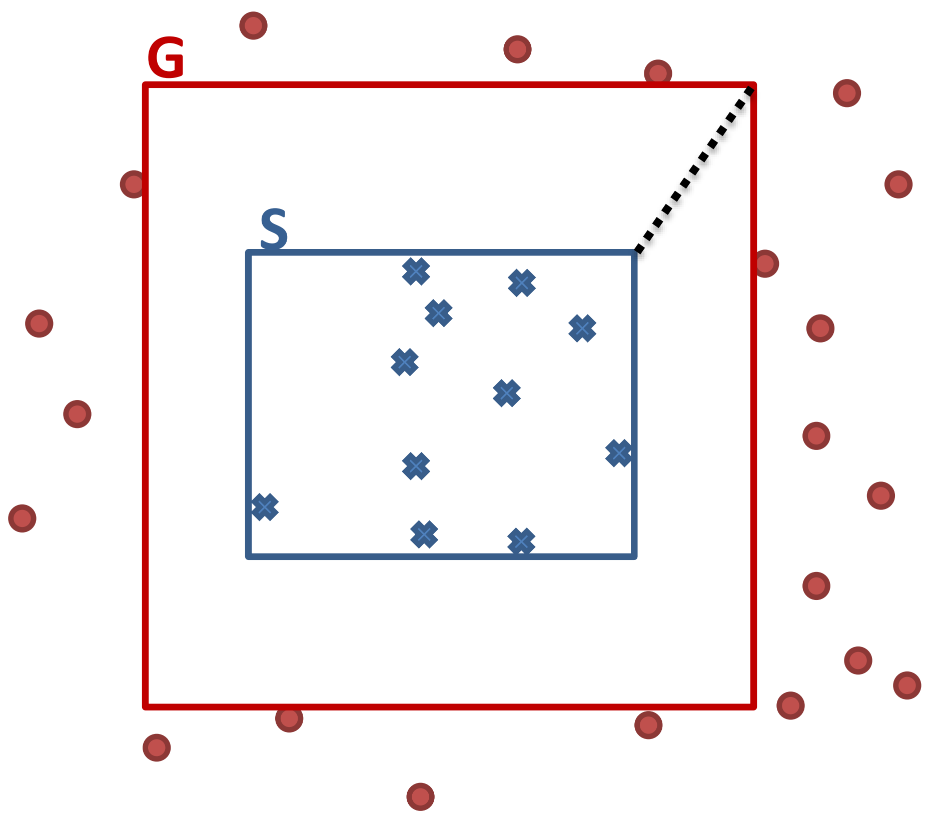


Teacher
$f_1(x) =^? f_2(x)$
x
true/false
f_1, f_2
D^3
$\varphi_{same}, \varphi_{diff}$
query
x
Oracle

The Main Idea: Active Learning with Abstraction

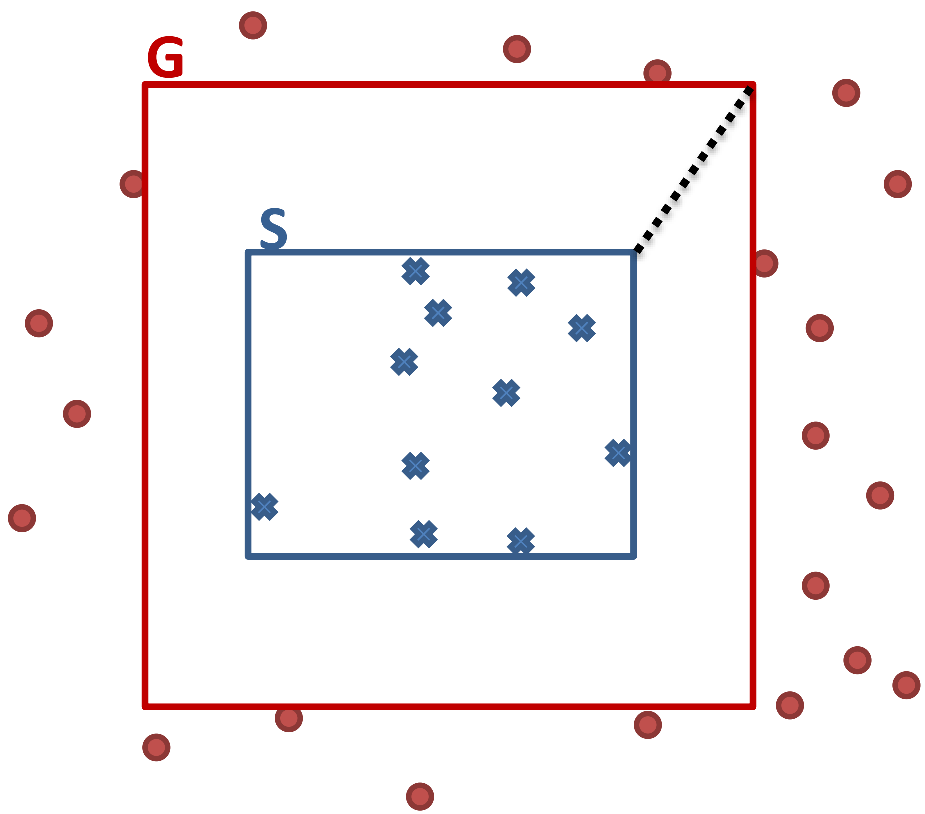
- Run the functions and label points as "positive" (same) or "negative" (different)
- Use Version Spaces with an abstract domain
  - Abstract both the similarity and the difference
  - Use the points where the abstractions disagree to extend/refine both abstractions
  - Stop once there are no more points or the entire domain has been discovered

The Classical Approach: Version Spaces



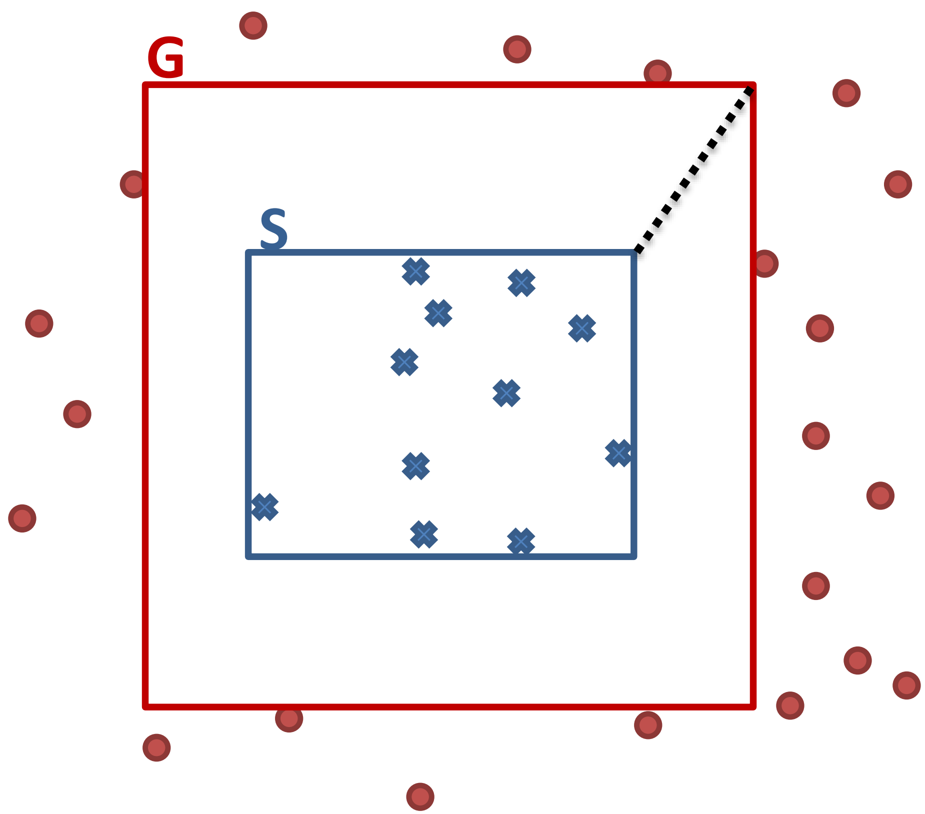
The Classical Approach: Version Spaces

S,G – elements in the box abstract domain
Also boolean functions over concrete domain



The Classical Approach: Version Spaces

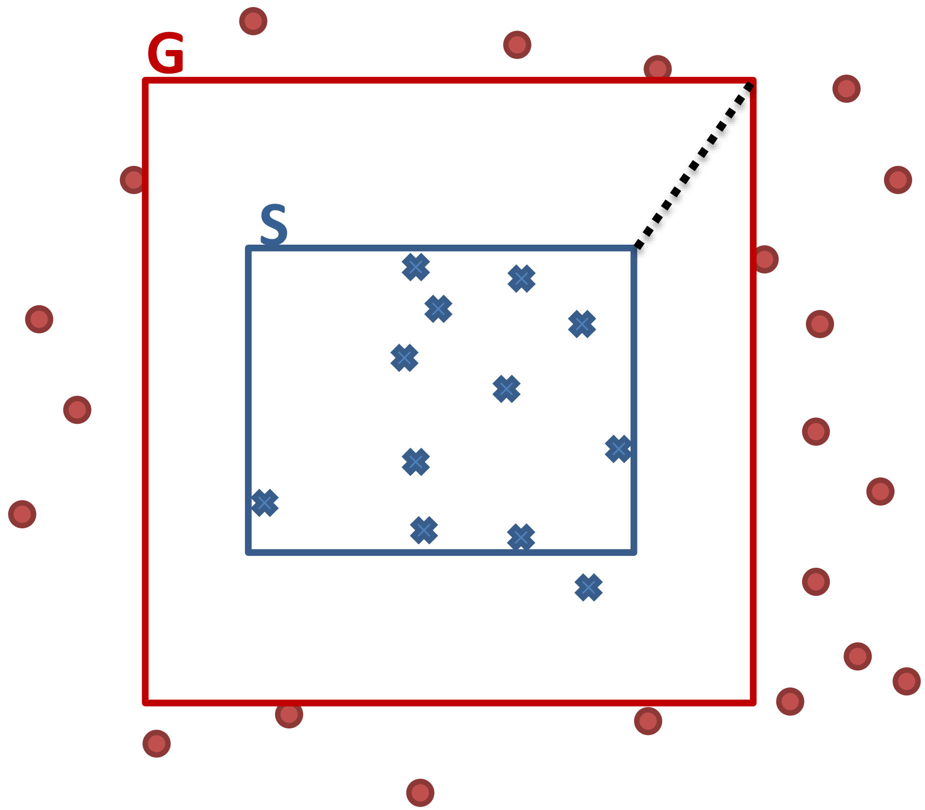
S,G – elements in the box abstract domain
Also boolean functions over concrete domain



φ_S - the formula representing the positive space
φ_G - positive space and unknown
$\varphi_{\neg G}$ - negative space

The Classical Approach: Version Spaces

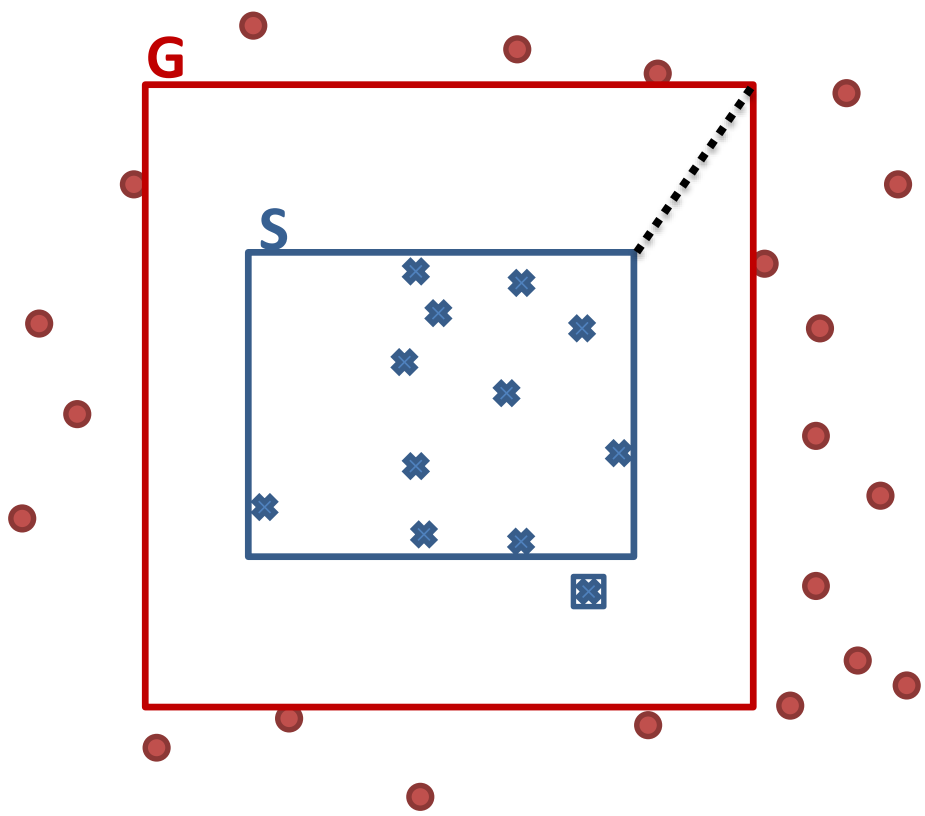
S,G – elements in the box abstract domain
Also boolean functions over concrete domain



φ_S - the formula representing the positive space
φ_G - positive space and unknown
$\varphi_{\neg G}$ - negative space

The Classical Approach: Version Spaces

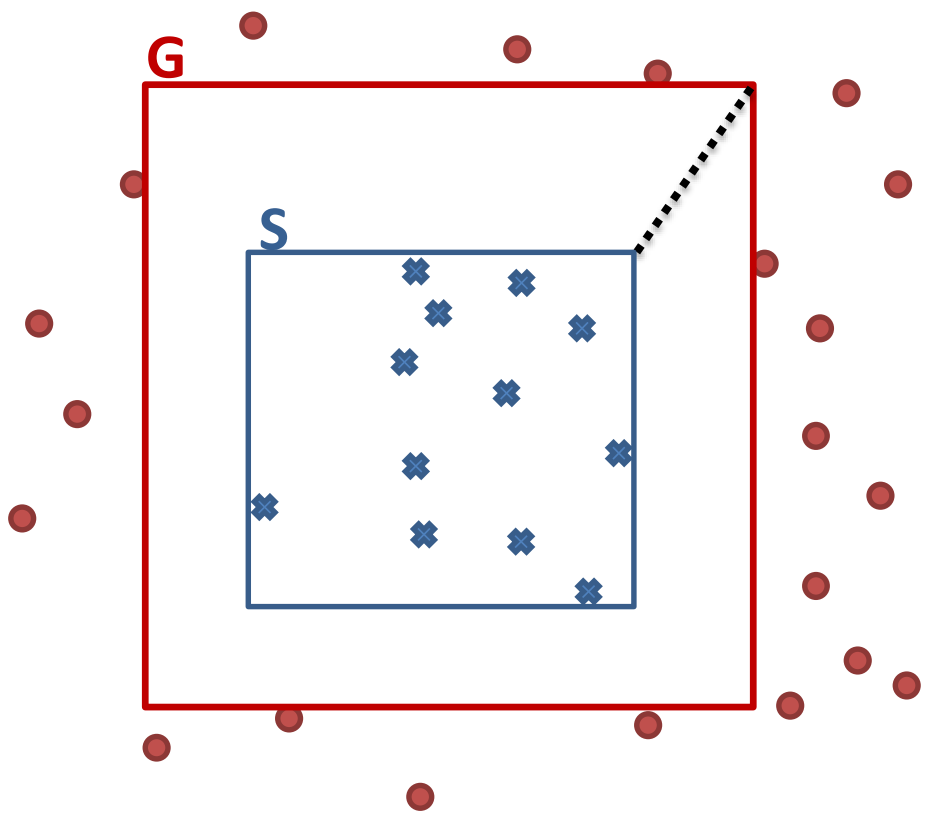
S,G – elements in the box abstract domain
Also boolean functions over concrete domain



φ_S - the formula representing the positive space
φ_G - positive space and unknown
$\varphi_{\neg G}$ - negative space

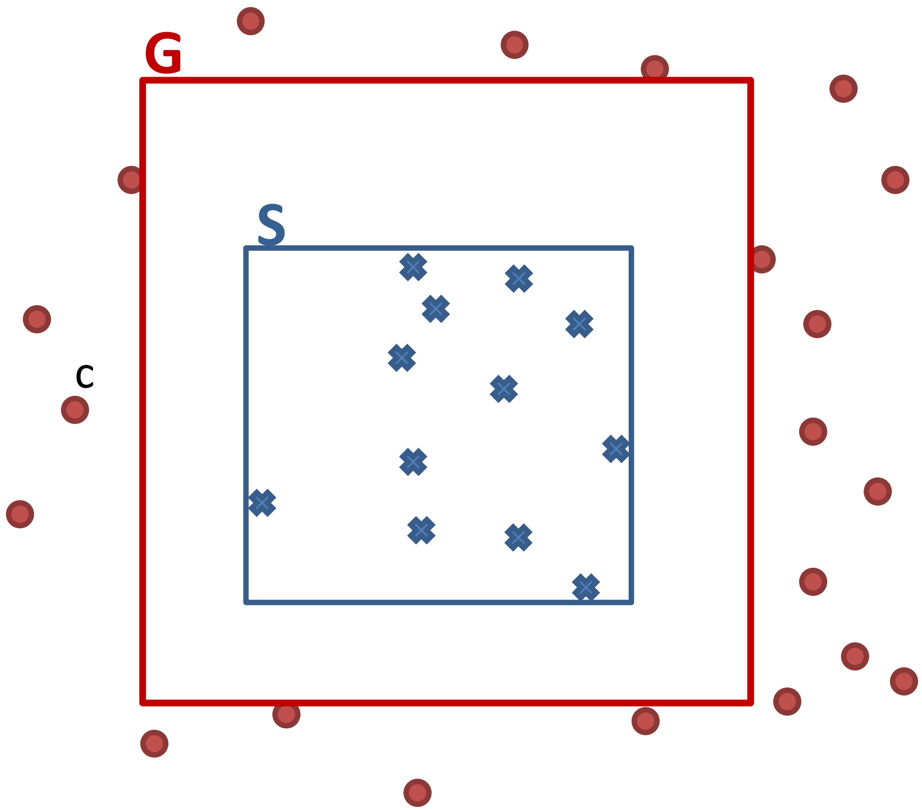
The Classical Approach: Version Spaces

S,G – elements in the box abstract domain
Also boolean functions over concrete domain

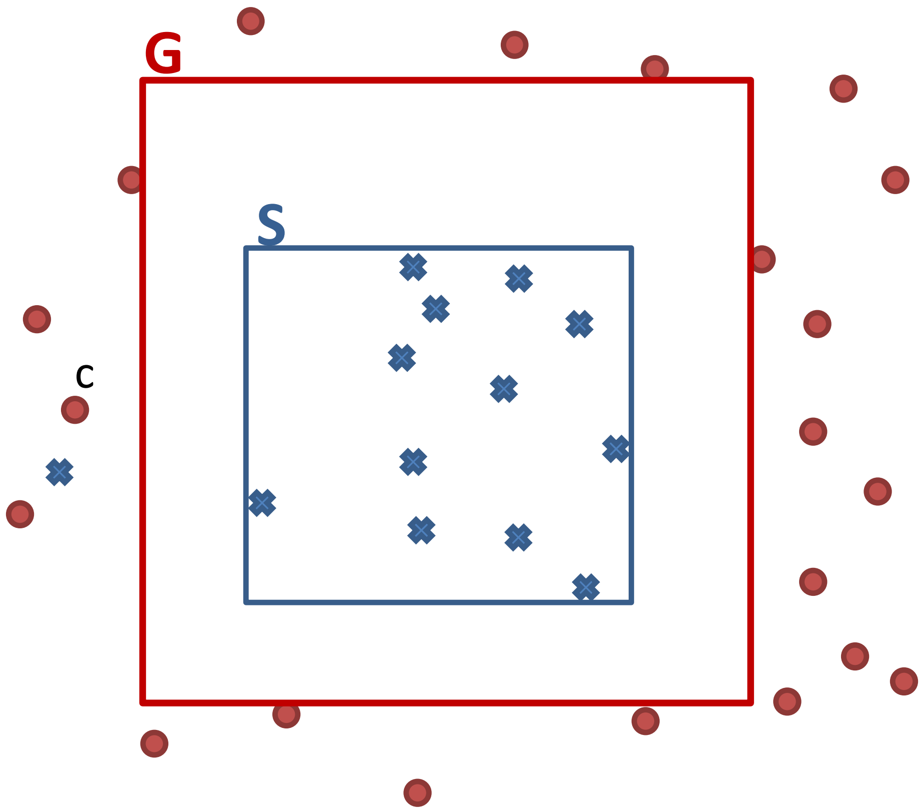


φ_S - the formula representing the positive space
φ_G - positive space and unknown
$\varphi_{\neg G}$ - negative space

Challenge: Representing Disjunction



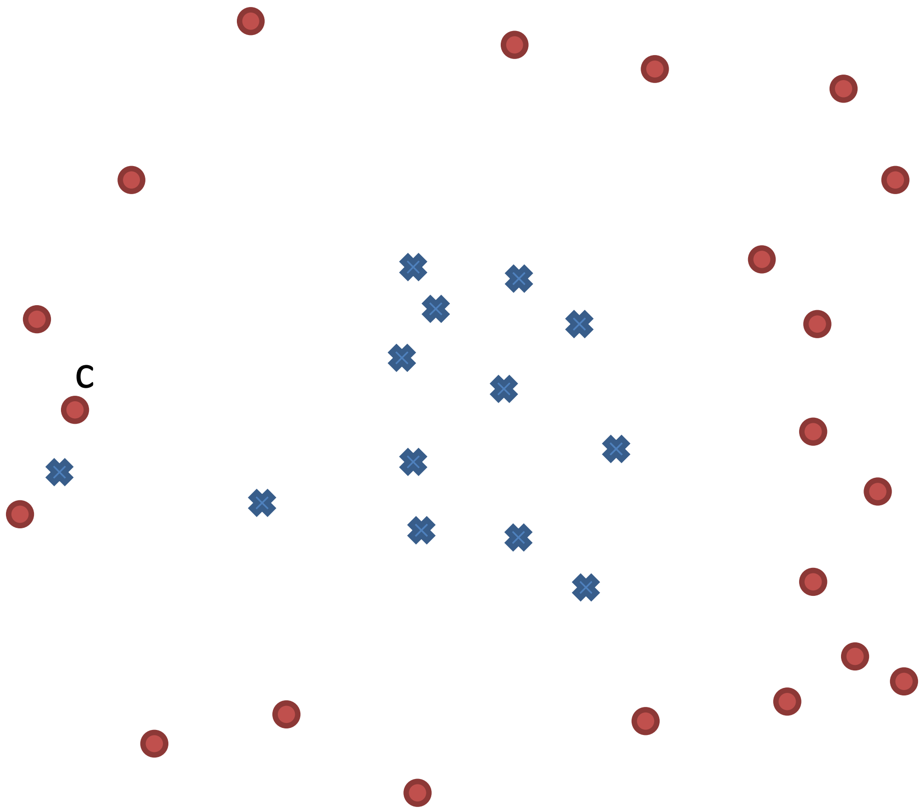
Challenge: Representing Disjunction



Challenge: Representing Disjunction

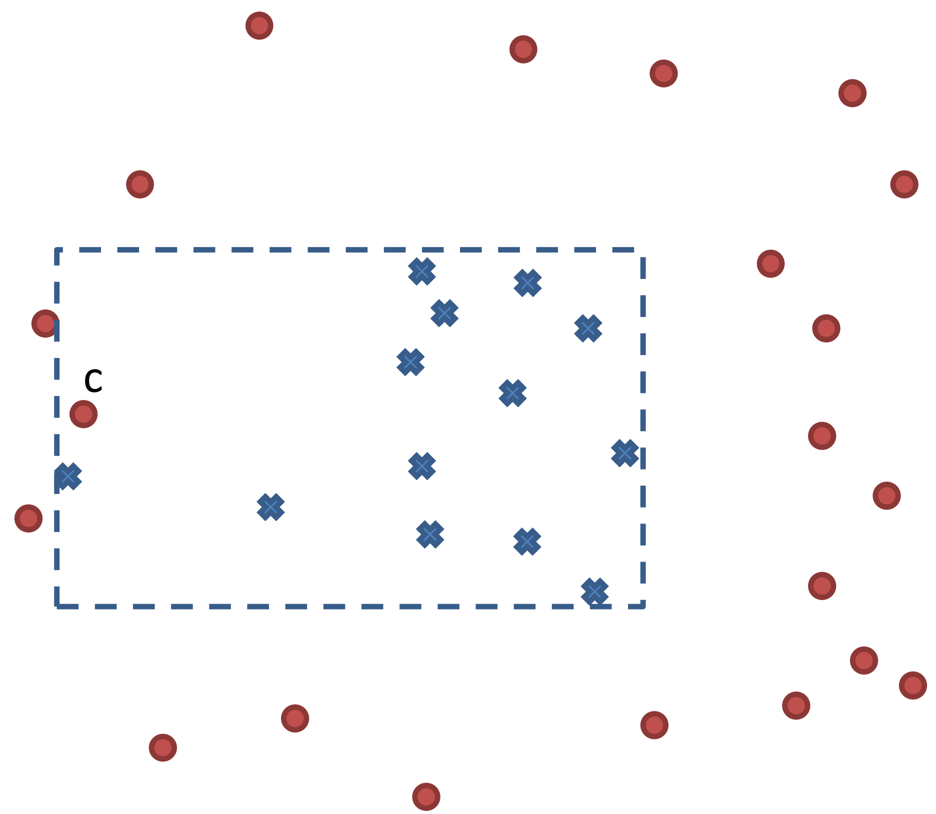
- Option 1: lose *consistency*
  $\varphi_{S'}(c) = true$

- Option 2: lose *abstraction*
  $\varphi_{S'} = \psi_1 \vee \psi_2 \vee \cdots$



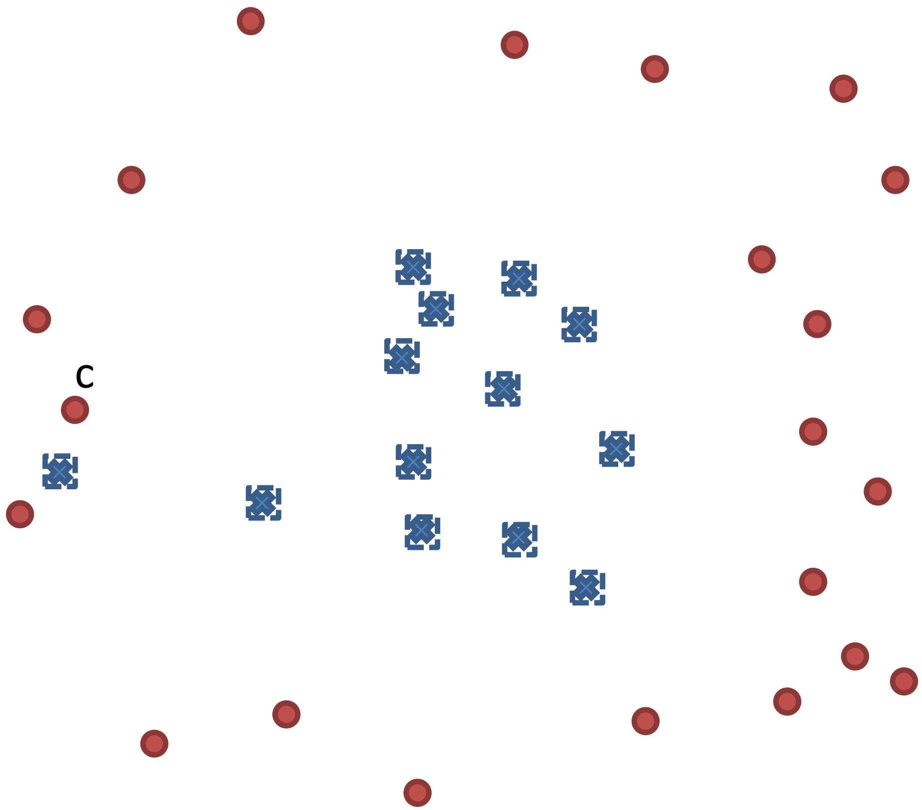
Challenge: Representing Disjunction

- Option 1: lose *consistency* $\varphi_{S'}(c) = true$
- Option 2: lose *abstraction* $\varphi_{S'} = \psi_1 \vee \psi_2 \vee \cdots$



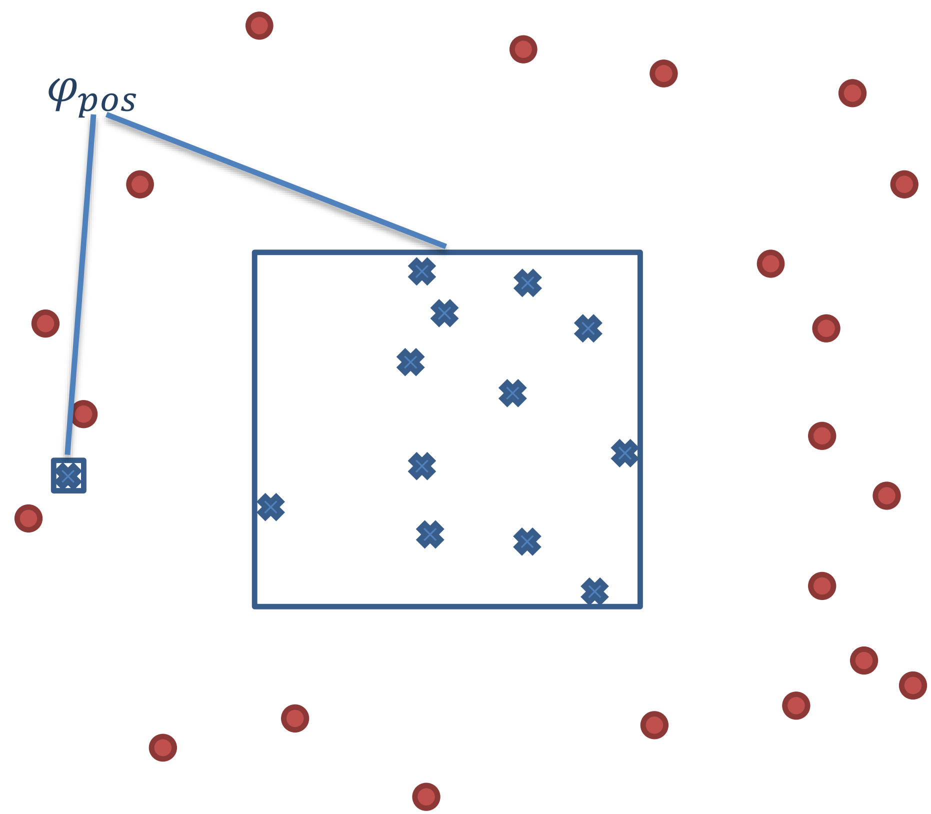
Challenge: Representing Disjunction

- Option 1: lose *consistency* $\varphi_{S'}(c) = true$
- Option 2: lose *abstraction* $\varphi_{S'} = \psi_1 \vee \psi_2 \vee \cdots$



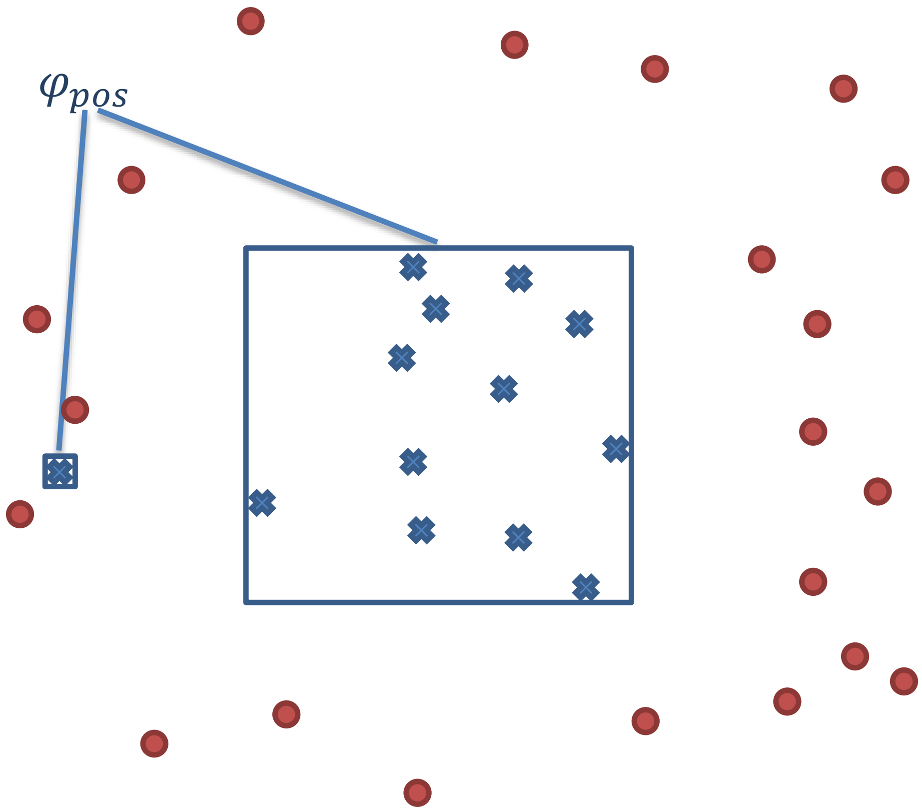
Solution: Disjunction With Abstraction

- Define a new operation: ***Safe Generalization***
- Goal: to abstract as much as possible while consistent



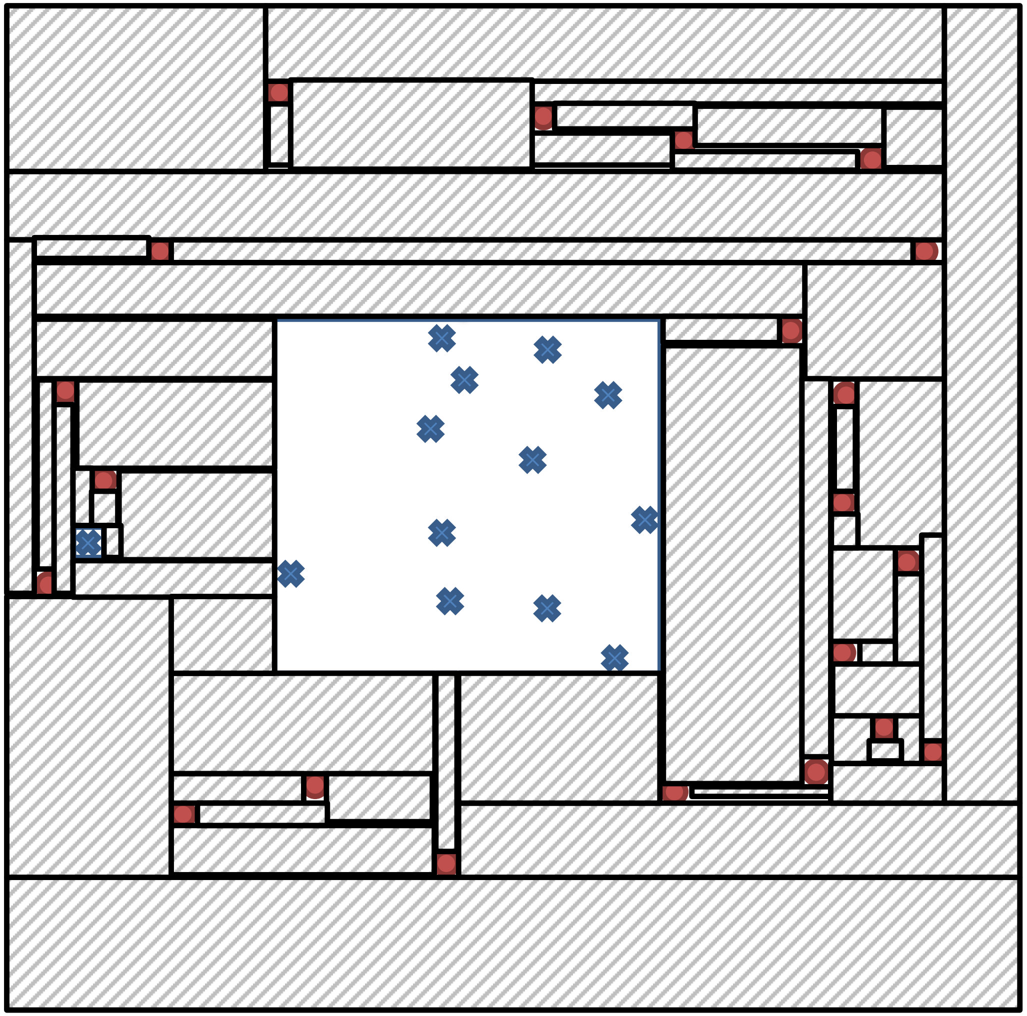
Challenge: Representing the Unknown

- Unknown:
  - Could we abstract it positively?
- A disjunctive abstraction can cover any space between the negative examples



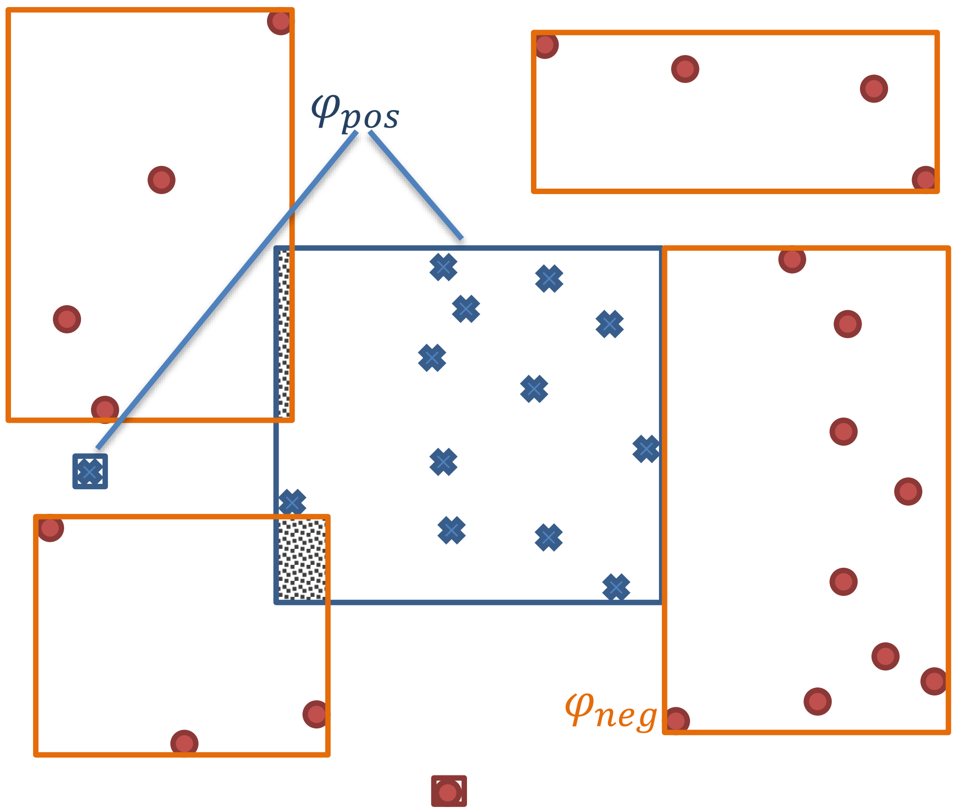
Challenge: Representing the Unknown

- Unknown:
  - Could we abstract it positively?
- A disjunctive abstraction can cover any space between the negative examples

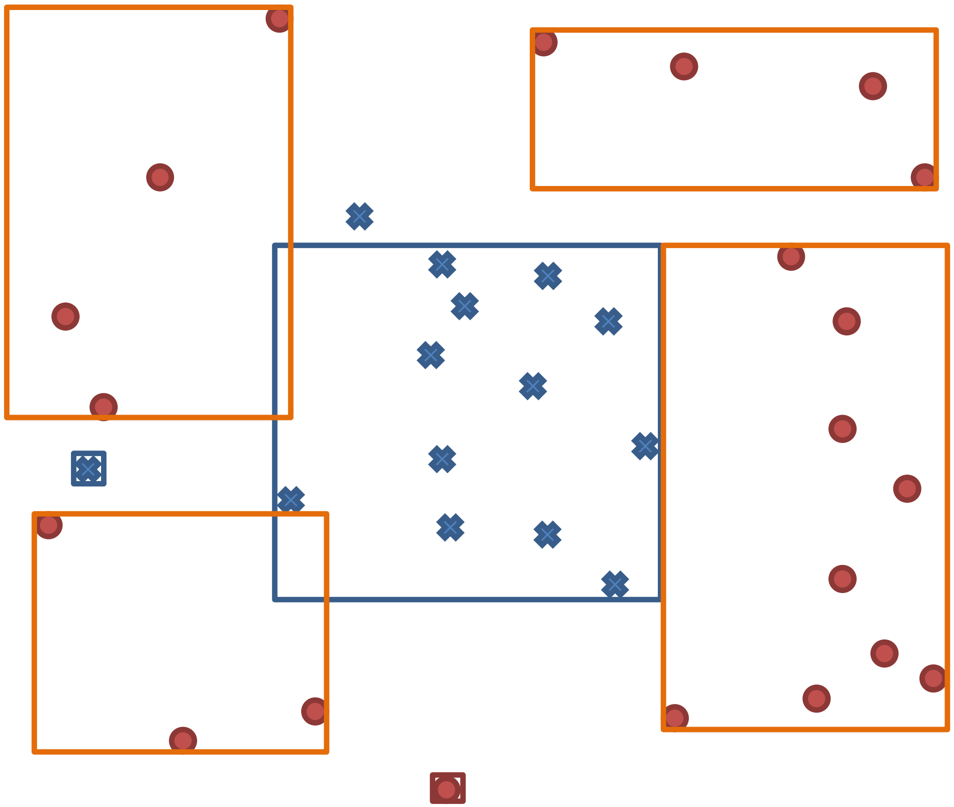


Solution: Abstract both Positive and Negative Points

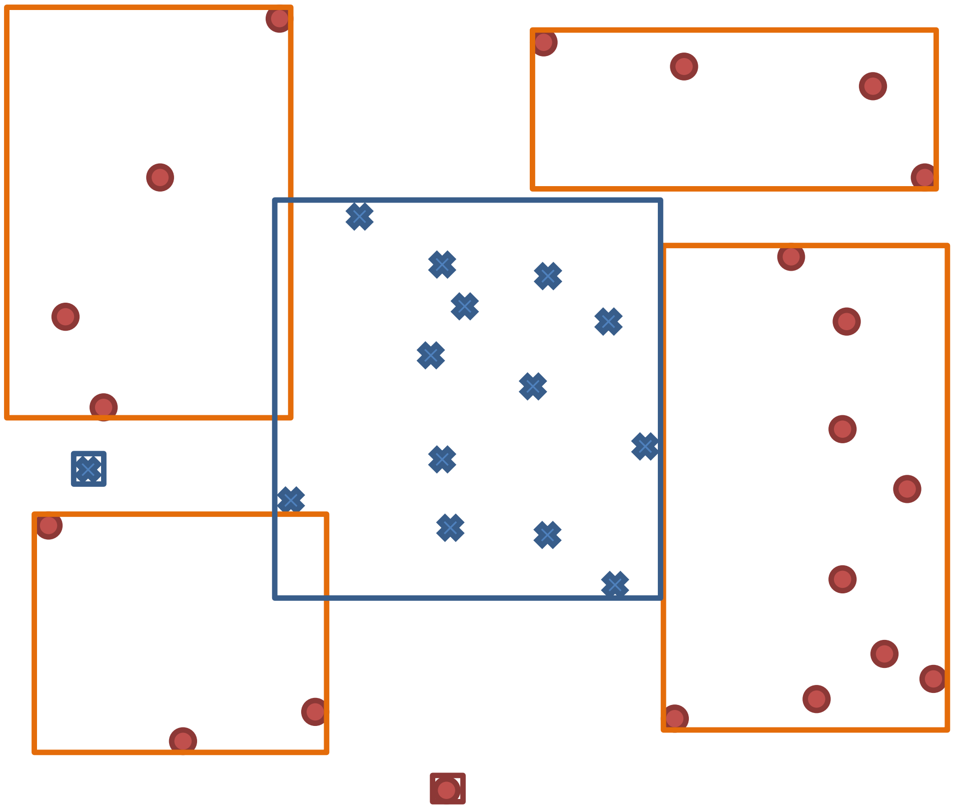
- The negative abstraction does not provide a general bound
- In VS, $\varphi_s \subseteq \varphi_G$
- Explicitly track *disagreement*



Challenge: Update the representation

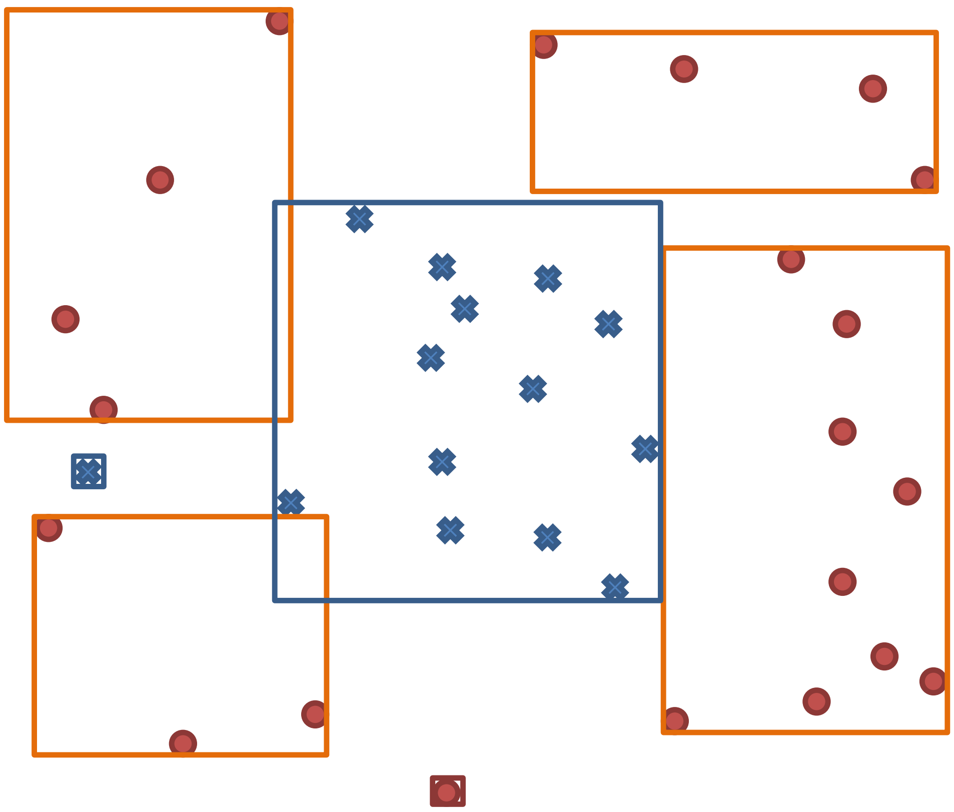


Challenge: Update the representation



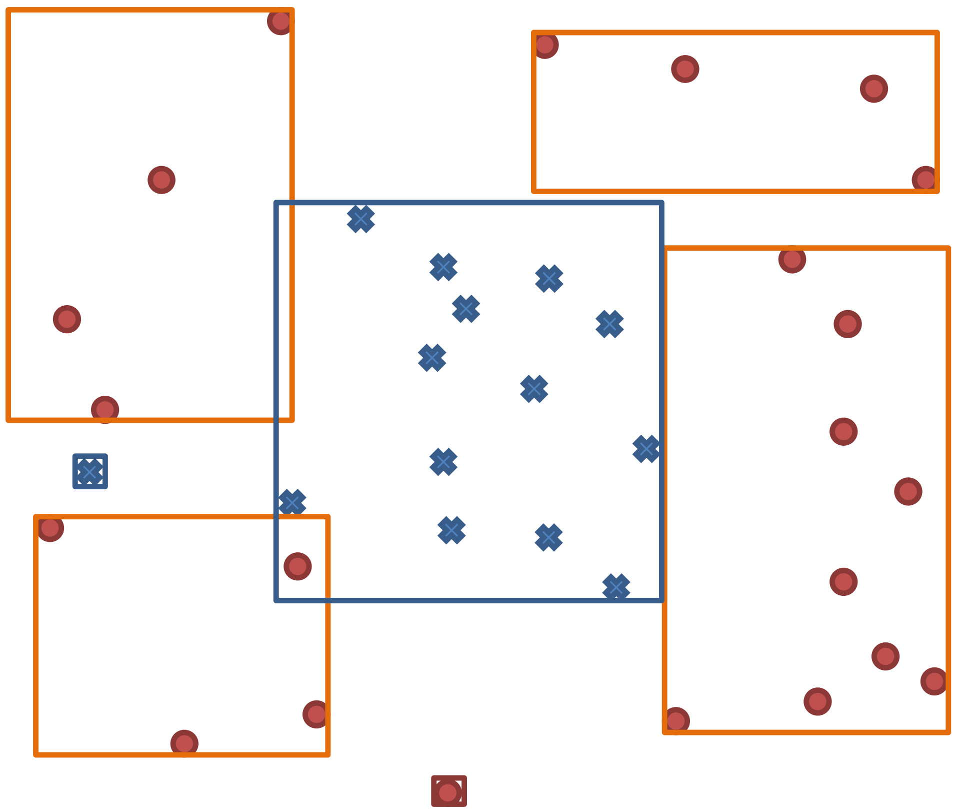
Challenge: Update the representation

- When a new point arrives, the abstractions need to be extended
- But abstractions can also break



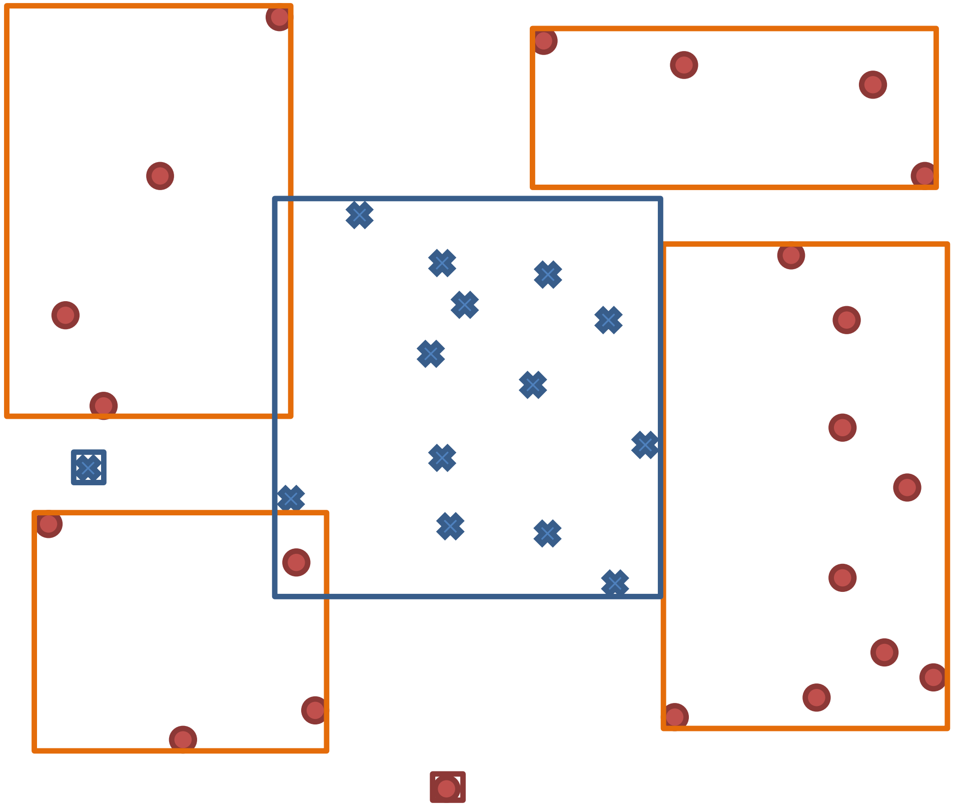
Challenge: Update the representation

- When a new point arrives, the abstractions need to be extended
- But abstractions can also break



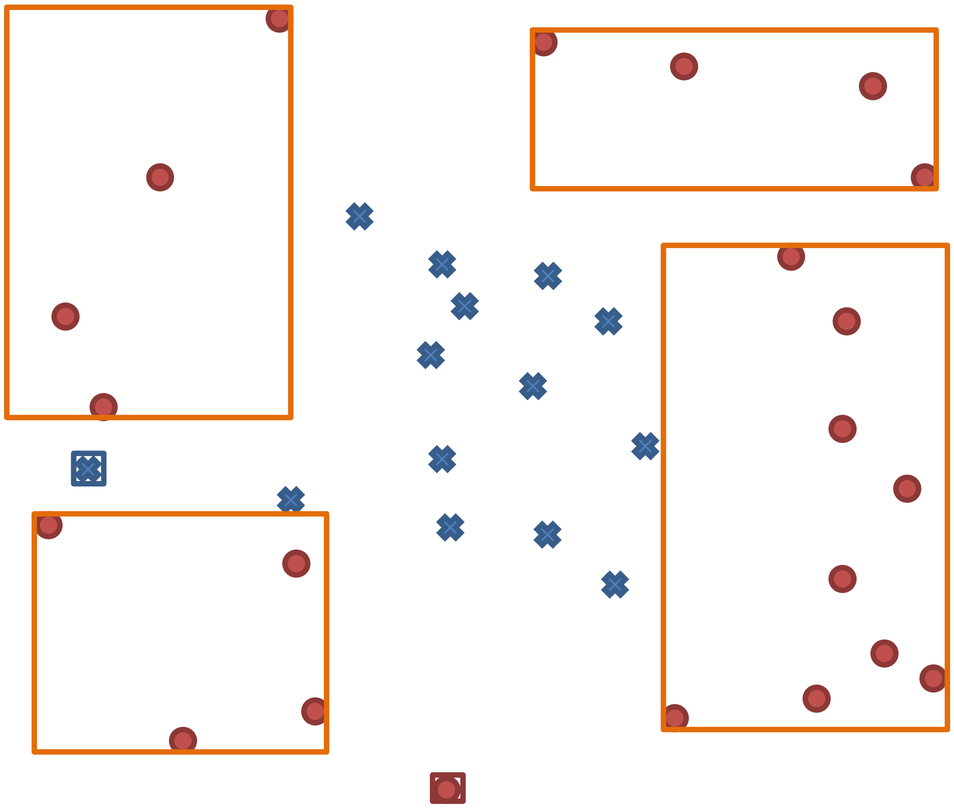
Solution: Refine when needed

- Settle for less abstraction
- φ_{pos} and φ_{neg} can disagree on abstracted points, not on concrete points



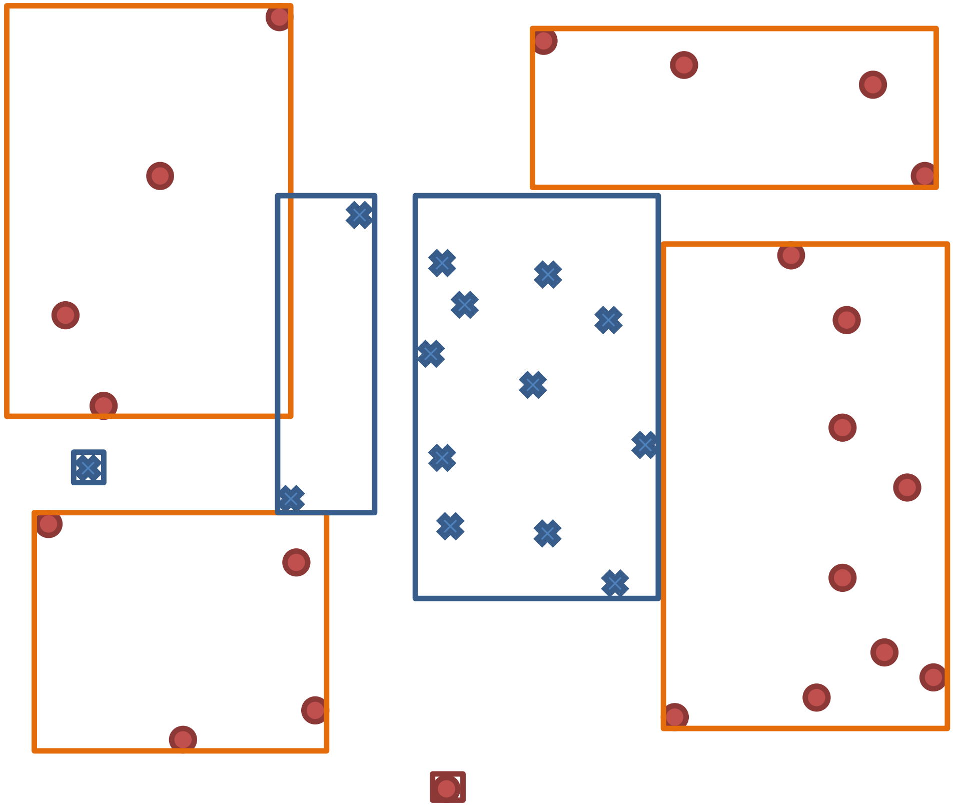
Solution: Refine when needed

- Settle for less abstraction
- φ_{pos} and φ_{neg} can disagree on abstracted points, not on concrete points



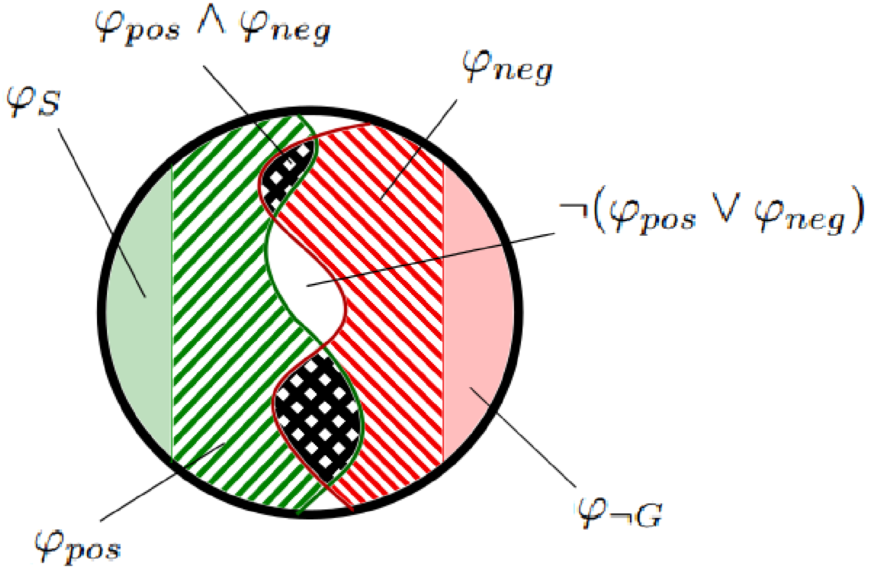
Solution: Refine when needed

- Settle for less abstraction
- φ_{pos} and φ_{neg} can disagree on abstracted points, not on concrete points

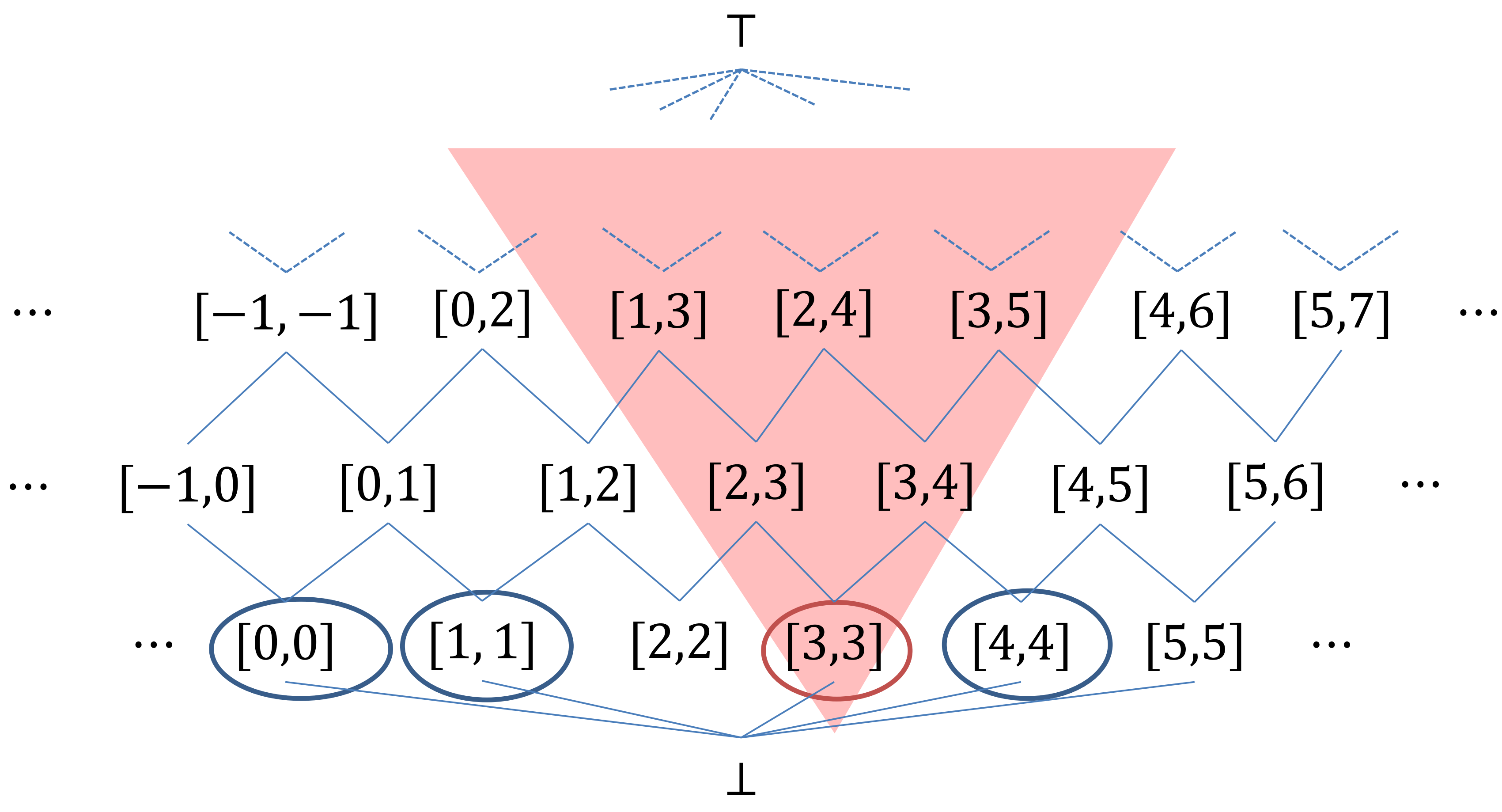


Active Learning with D^3

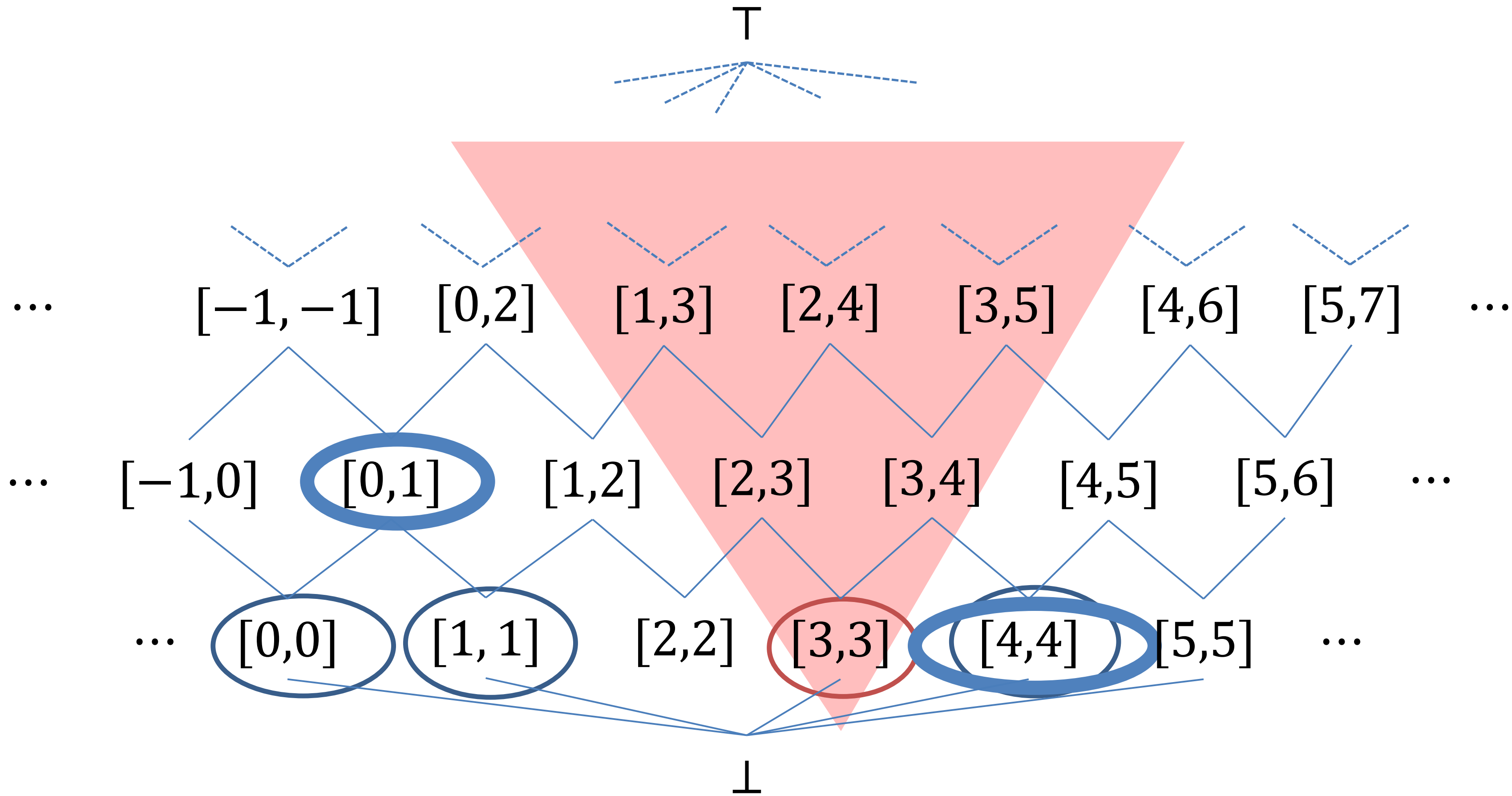
- Keep four formulas
  - Positive: abstract (φ_{pos}) and concrete (φ_S)
  - Negative: abstract (φ_{neg}) and concrete ($\varphi_{\neg G}$)
- Pick the next sample from the regions of disagreement
- Each region has a different potential effect on progress



Safe Generalization



Safe Generalization



Safe Generalization

$SG(A, C_{cex}): 2^L \times D \to 2^L$

- **Abstraction:** $\forall a \in A. \exists a' \in SG(A, C_{cex}). a \preccurlyeq a'$
- **Separation:** $\forall a \in SG(A, C_{cex}). \gamma(a) \cap C_{cex} = \emptyset$
- **Precision:** $\forall a \in SG(A, C_{cex}). \exists A' \subseteq A. a = \bigsqcup A'$

- **Maximality:** for every $a \in L$ that satisfies separation and precision, $\exists a' \in SG(A, C_{cex}). a \preccurlyeq a'$.

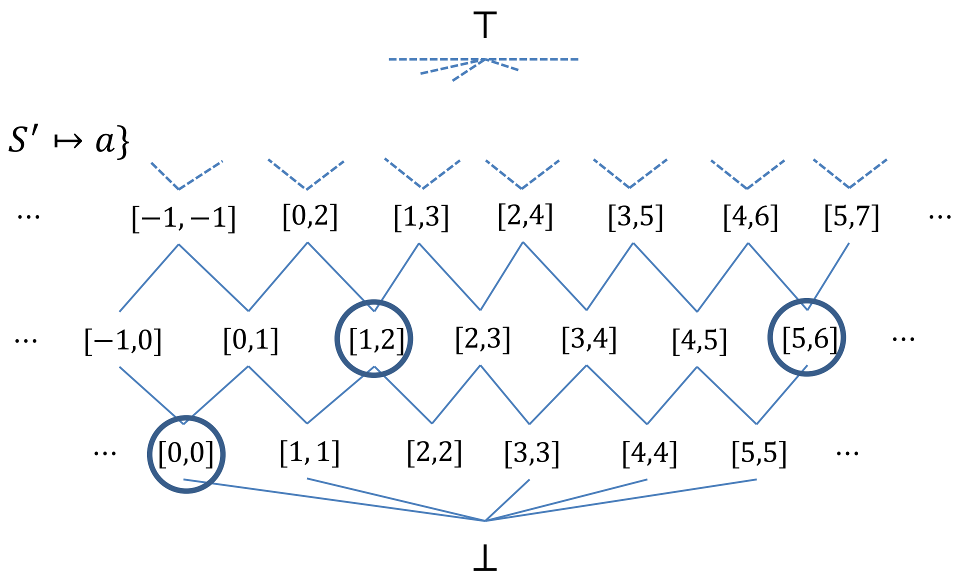
Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \dots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

- $consistent \leftarrow \{\{a_i\} \mapsto a_i | a_i \in A\}$
- for $lvl \leftarrow 2 \dots k$:
  - $prev \leftarrow \{S | S \in dom(consistent), |S| = lvl - 1\}$
  - $pairs \leftarrow \{(S, S') | S, S' \in prev, |S \cup S'| = lvl\}$
  - for $(S, S') \in pairs$:
    - $a \leftarrow consistent(S) \sqcup consistent(S')$
    - if $\gamma(a) \cap C_{cex} = \emptyset$ then
      - $consistent \leftarrow consistent \cup \{S \cup S' \mapsto a\}$
- //next slide

A={[0,0],[1,2],[5,6]}

C_{cex}={3}



Greedy Computation of SG

$SG(A = \{a_1, \dots, a_k\}, C_{cex})$:

$consistent \leftarrow \{\{a_i\} \mapsto a_i | a_i \in A\}$

for $lvl \leftarrow 2 \dots k$:

$prev \leftarrow \{S | S \in dom(consistent), |S| = lvl - 1\}$

$pairs \leftarrow \{(S, S') | S, S' \in prev, |S \cup S'| = lvl\}$

for $(S, S') \in pairs$:

$a \leftarrow consistent(S) \sqcup consistent(S')$

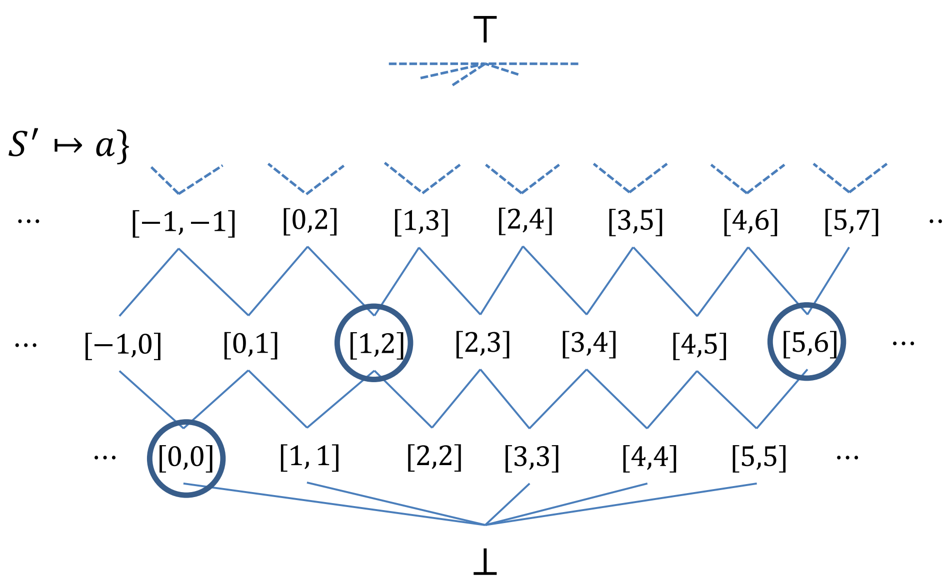
if $\gamma(a) \cap C_{cex} = \emptyset$ then

$consistent \leftarrow consistent \cup \{S \cup S' \mapsto a\}$

//next slide

A={[0,0],[1,2],[5,6]}

C_{cex}={3}



$$consistent = \left\{ \begin{matrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6] \end{matrix} \right\}$$

Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \dots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

$consistent \leftarrow \{\{a_i\} \mapsto a_i | a_i \in A\}$

for $lvl \leftarrow 2 \dots k$:

$prev \leftarrow \{S | S \in dom(consistent), |S| = lvl - 1\}$

$pairs \leftarrow \{(S, S') | S, S' \in prev, |S \cup S'| = lvl\}$

for $(S, S') \in pairs$:

$a \leftarrow consistent(S) \sqcup consistent(S')$

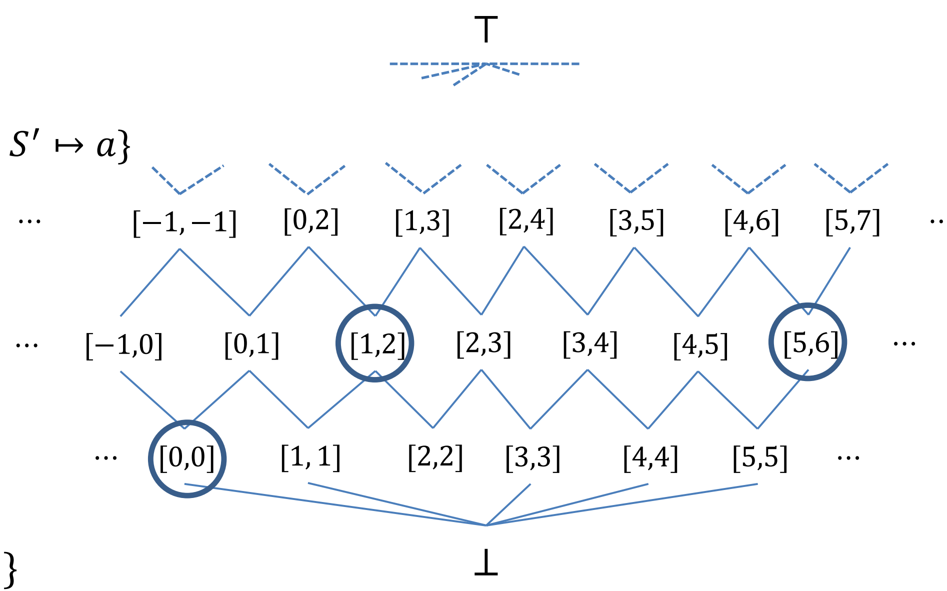
if $\gamma(a) \cap C_{cex} = \emptyset$ then

$consistent \leftarrow consistent \cup \{S \cup S' \mapsto a\}$

//next slide

A={[0,0],[1,2],[5,6]}

C_{cex}={3}



$$consistent = \left\{ \begin{matrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6] \end{matrix} \right\}$$

$prev = \{[0,0],[1,2],[5,6]\}$

$pairs = \{([0,0],[1,2]), ([1,2],[5,6]), ([0,0],[5,6])\}$

Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \ldots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

$consistent \leftarrow \{\{a_i\} \mapsto a_i | a_i \in A\}$

for $lvl \leftarrow 2 \ldots k$:

$prev \leftarrow \{S | S \in dom(consistent), |S| = lvl - 1\}$

$pairs \leftarrow \{(S, S') | S, S' \in prev, |S \cup S'| = lvl\}$

for $(S, S') \in pairs$:

$a \leftarrow consistent(S) \sqcup consistent(S')$

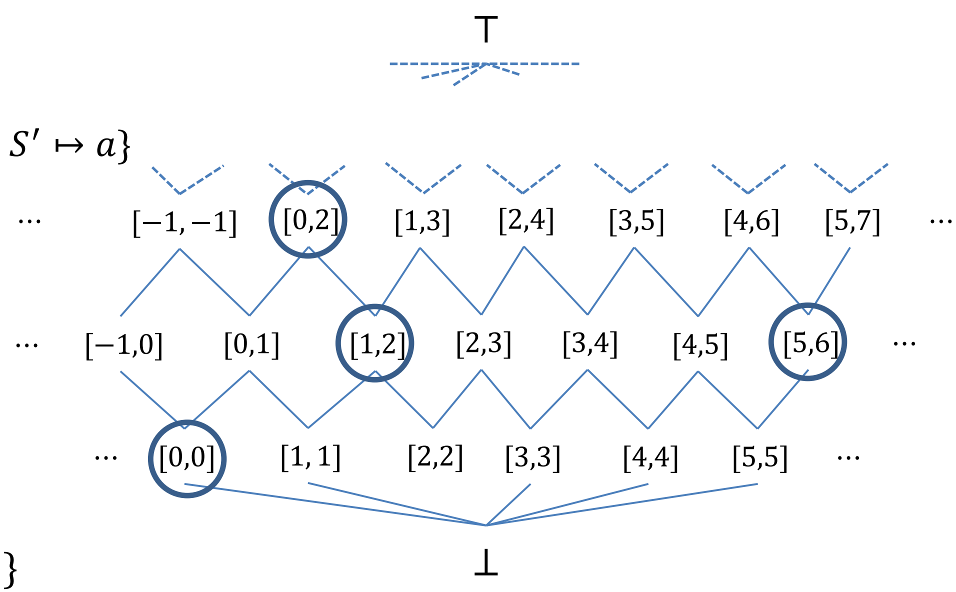
if $\gamma(a) \cap C_{cex} = \emptyset$ then

$consistent \leftarrow consistent \cup \{S \cup S' \mapsto a\}$

//next slide

A={[0,0],[1,2],[5,6]}

C_{cex}={3}



$$consistent = \left\{\begin{matrix}\{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6]\end{matrix}\right\}$$

$prev = \{[0,0],[1,2],[5,6]\}$

$pairs = \{([0,0], [1,2]), ([1,2], [5,6]), ([0,0], [5,6])\}$

Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \dots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

$consistent \leftarrow \{\{a_i\} \mapsto a_i | a_i \in A\}$

for $lvl \leftarrow 2 \dots k$:

$prev \leftarrow \{S | S \in dom(consistent), |S| = lvl - 1\}$

$pairs \leftarrow \{(S, S') | S, S' \in prev, |S \cup S'| = lvl\}$

for $(S, S') \in pairs$:

$a \leftarrow consistent(S) \sqcup consistent(S')$

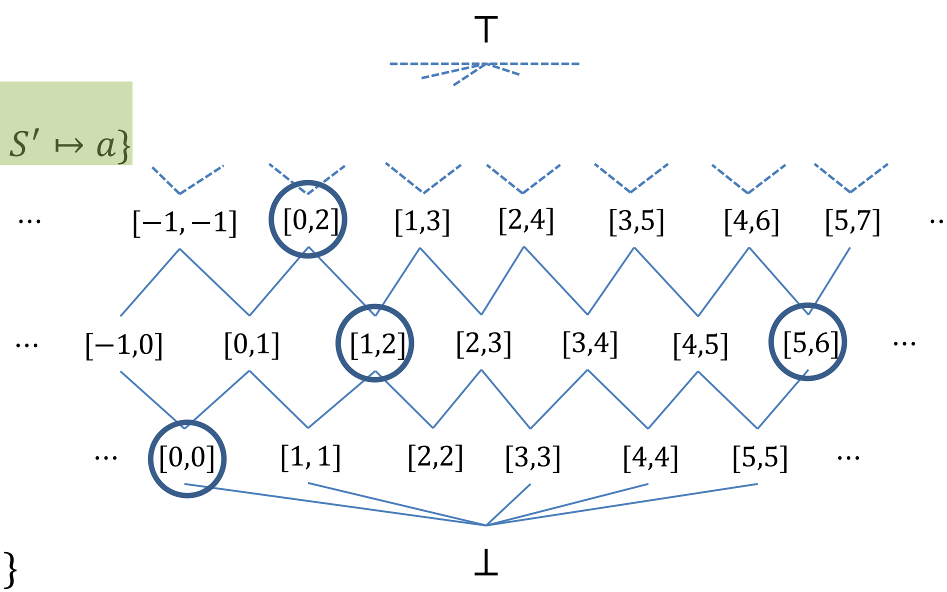
if $\gamma(a) \cap C_{cex} = \emptyset$ then

$consistent \leftarrow consistent \cup \{S \cup S' \mapsto a\}$

//next slide

A={[0,0],[1,2],[5,6]}

C_{cex}={3}



$$consistent = \left\{ \begin{matrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6] \end{matrix} \right\}$$

$prev = \{[0,0],[1,2],[5,6]\}$

$pairs = \{([0,0], [1,2]), ([1,2], [5,6]), ([0,0], [5,6])\}$

Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \dots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_\mathrm{cex})$:

$consistent \leftarrow \{\{a_i\} \mapsto a_i | a_i \in A\}$

for $lvl \leftarrow 2 \dots k$:

$prev \leftarrow \{S | S \in dom(consistent), |S| = lvl - 1\}$

$pairs \leftarrow \{(S, S') | S, S' \in prev, |S \cup S'| = lvl\}$

for $(S, S') \in pairs$:

$a \leftarrow consistent(S) \sqcup consistent(S')$

if $\gamma(a) \cap C_{cex} = \emptyset$ then

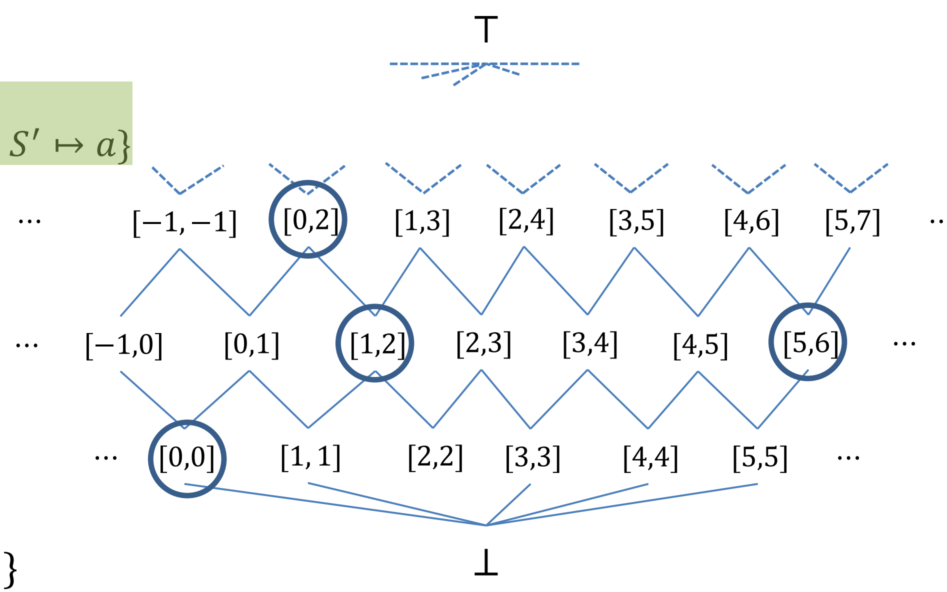
$consistent \leftarrow consistent \cup \{S \cup S' \mapsto a\}$

//next slide

A={[0,0],[1,2],[5,6]}

C_{cex}={3}

$\gamma([0,2]) \cap C_{cex} = \{0,1,2\} \cap \{3\} = \emptyset$



$$consistent = \begin{Bmatrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6] \end{Bmatrix}$$

$prev = \{[0,0],[1,2],[5,6]\}$

$pairs = \{([0,0],[1,2]), ([1,2],[5,6]), ([0,0],[5,6])\}$

Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \dots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

$consistent \leftarrow \{\{a_i\} \mapsto a_i | a_i \in A\}$

for $lvl \leftarrow 2 \dots k$:

$prev \leftarrow \{S | S \in dom(consistent), |S| = lvl - 1\}$

$pairs \leftarrow \{(S, S') | S, S' \in prev, |S \cup S'| = lvl\}$

for $(S, S') \in pairs$:

$a \leftarrow consistent(S) \sqcup consistent(S')$

if $\gamma(a) \cap C_{cex} = \emptyset$ then

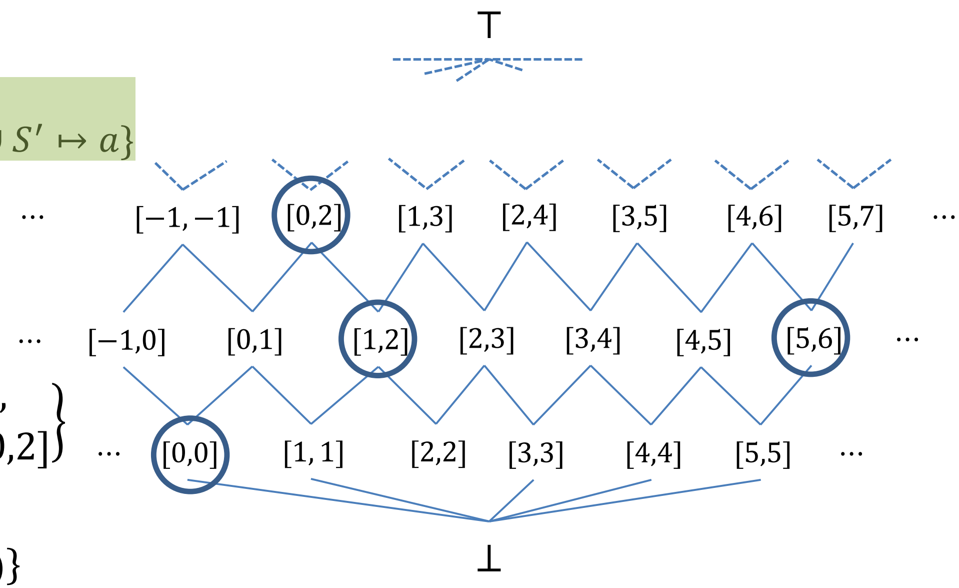
$consistent \leftarrow consistent \cup \{S \cup S' \mapsto a\}$

//next slide

A={[0,0],[1,2],[5,6]}

C_{cex}={3}

$\gamma([0,2]) \cap C_{cex} = \{0,1,2\} \cap \{3\} = \emptyset$



$$consistent = \begin{Bmatrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6], \{[0,0],[1,2]\} \mapsto [0,2] \end{Bmatrix}$$

$prev = \{[0,0],[1,2],[5,6]\}$

$pairs = \{([0,0],[1,2]), ([1,2],[5,6]), ([0,0],[5,6])\}$

Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \dots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

$consistent \leftarrow \{\{a_i\} \mapsto a_i | a_i \in A\}$

for $lvl \leftarrow 2 \dots k$:

$prev \leftarrow \{S | S \in dom(consistent), |S| = lvl - 1\}$

$pairs \leftarrow \{(S, S') | S, S' \in prev, |S \cup S'| = lvl\}$

for $(S, S') \in pairs$:

$a \leftarrow consistent(S) \sqcup consistent(S')$

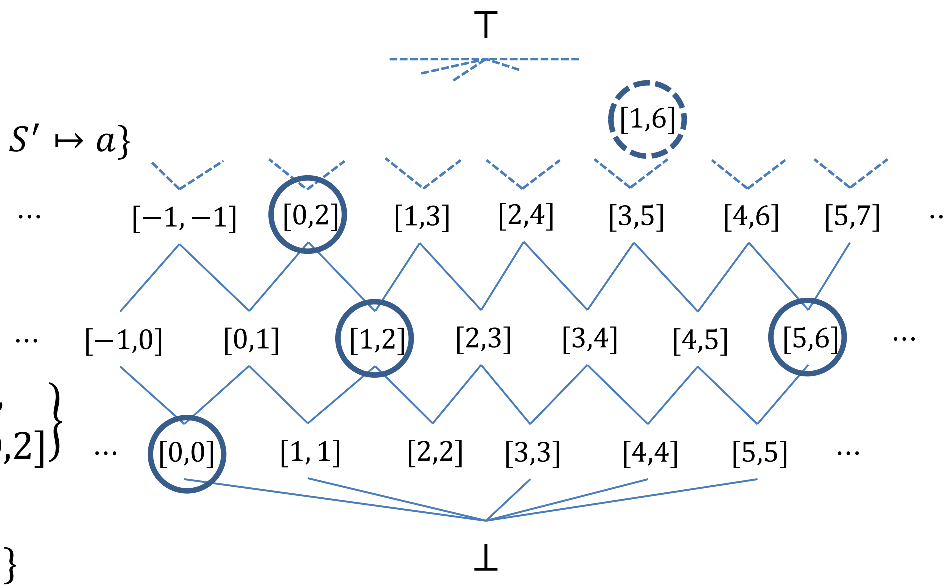
if $\gamma(a) \cap C_{cex} = \emptyset$ then

$consistent \leftarrow consistent \cup \{S \cup S' \mapsto a\}$

//next slide

A={[0,0],[1,2],[5,6]}

C_{cex}={3}



$$consistent = \left\{ \begin{matrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6], \{[0,0],[1,2]\} \mapsto [0,2] \end{matrix} \right\}$$

$prev = \{[0,0],[1,2],[5,6]\}$

$pairs = \{([0,0],[1,2]), ([1,2],[5,6]), ([0,0],[5,6])\}$

Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \dots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

$\quad consistent \leftarrow \{\{a_i\} \mapsto a_i | a_i \in A\}$

$\quad$ for $lvl \leftarrow 2 \dots k$:

$\qquad prev \leftarrow \{S | S \in dom(consistent), |S| = lvl - 1\}$

$\qquad pairs \leftarrow \{(S, S') | S, S' \in prev, |S \cup S'| = lvl\}$

$\qquad$ for $(S, S') \in pairs$:

$\qquad\quad a \leftarrow consistent(S) \sqcup consistent(S')$

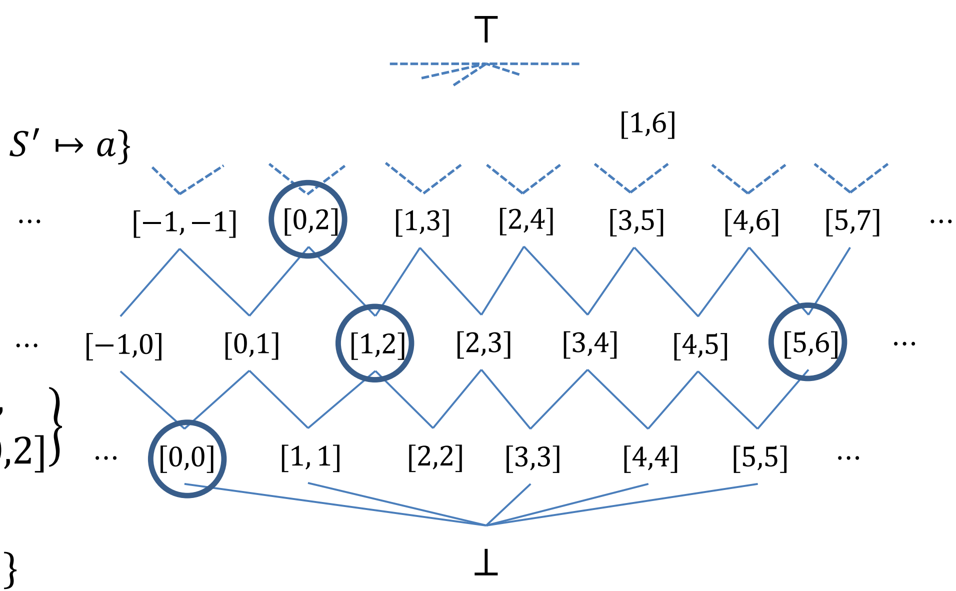
$\qquad\quad$ if $\gamma(a) \cap C_{cex} = \emptyset$ then

$\qquad\qquad consistent \leftarrow consistent \cup \{S \cup S' \mapsto a\}$

$\quad$ //next slide

A={[0,0],[1,2],[5,6]}

C_{cex}={3}



$$consistent = \left\{\begin{array}{c}\{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6], \{[0,0],[1,2]\} \mapsto [0,2]\end{array}\right\}$$

$prev = \{[0,0],[1,2],[5,6]\}$

$pairs = \{([0,0],[1,2]), ([1,2],[5,6]), ([0,0],[5,6])\}$

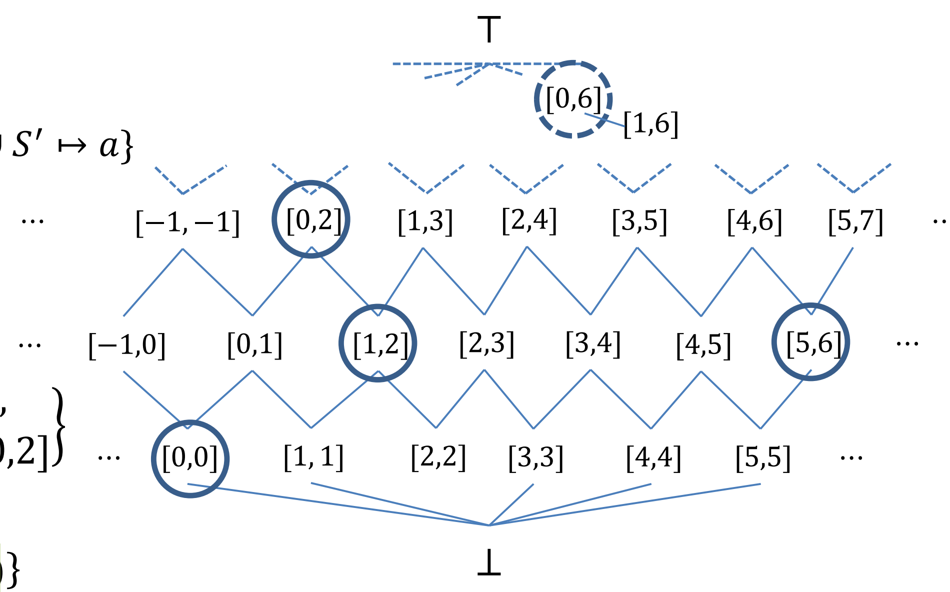
Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \dots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

- $consistent \leftarrow \{\{a_i\} \mapsto a_i | a_i \in A\}$
- for $lvl \leftarrow 2 \dots k$:
  - $prev \leftarrow \{S | S \in dom(consistent), |S| = lvl - 1\}$
  - $pairs \leftarrow \{(S, S') | S, S' \in prev, |S \cup S'| = lvl\}$
  - for $(S, S') \in pairs$:
    - $a \leftarrow consistent(S) \sqcup consistent(S')$
    - if $\gamma(a) \cap C_{cex} = \emptyset$ then
      - $consistent \leftarrow consistent \cup \{S \cup S' \mapsto a\}$
- //next slide

A={[0,0],[1,2],[5,6]}

C_{cex}={3}



$$consistent = \left\{ \begin{matrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6], \{[0,0],[1,2]\} \mapsto [0,2] \end{matrix} \right\}$$

$prev = \{[0,0],[1,2],[5,6]\}$

$pairs = \{([0,0],[1,2]), ([1,2],[5,6]), ([0,0],[5,6])\}$

Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \dots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

//prev slide

$res \leftarrow \emptyset$

$seen \leftarrow \emptyset$

while $seen \neq A$ do:

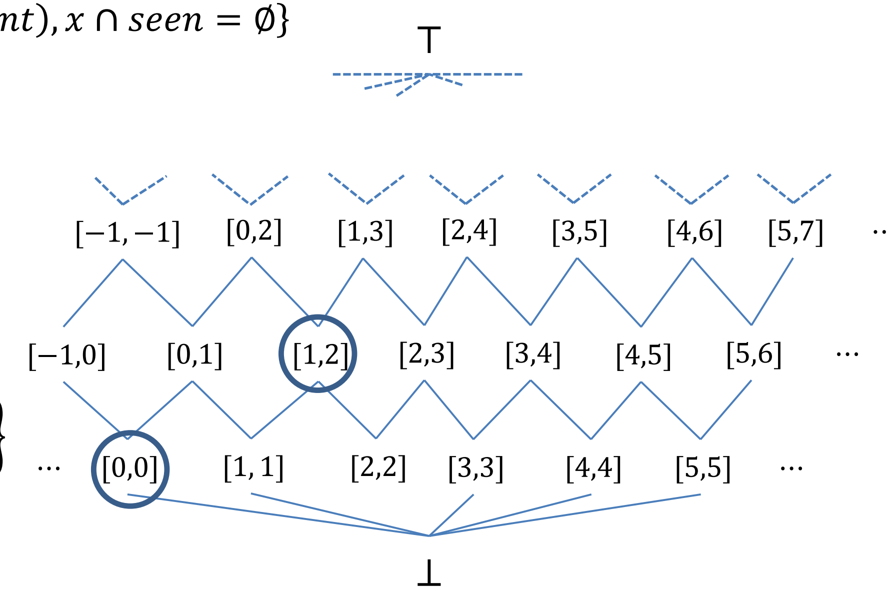
$joint \leftarrow argmax_x\{card(x) | x \in dom(consistent), x \cap seen = \emptyset\}$

$seen \leftarrow seen \cup joint$

$res \leftarrow res \cup consistent(joint)$

return res

$$consistent = \left\{ \begin{matrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6], \{[0,0],[1,2]\} \mapsto [0,2] \end{matrix} \right\}$$



Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \dots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

//prev slide

$res \leftarrow \emptyset$

$seen \leftarrow \emptyset$

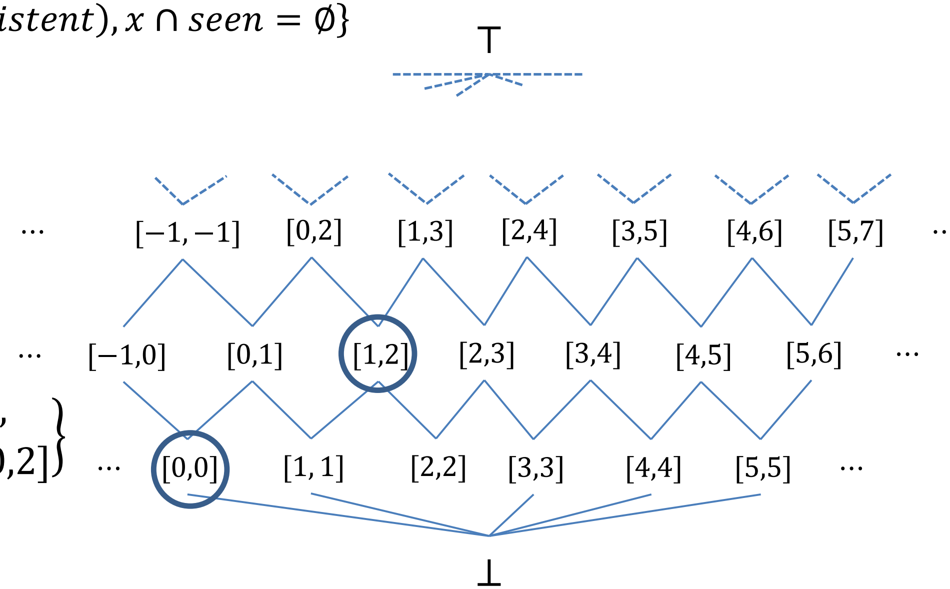
while $seen \neq A$ do:

$joint \leftarrow argmax_x\{card(x) | x \in dom(consistent), x \cap seen = \emptyset\}$

$seen \leftarrow seen \cup joint$

$res \leftarrow res \cup consistent(joint)$

return res



$$consistent = \begin{Bmatrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6], \{[0,0],[1,2]\} \mapsto [0,2] \end{Bmatrix}$$

$res = \emptyset$

$seen = \emptyset$

Greedy Computation of SG

$SG(A = \{a_1, \ldots, a_k\}, C_{cex})$:

//prev slide

$res \leftarrow \emptyset$

$seen \leftarrow \emptyset$

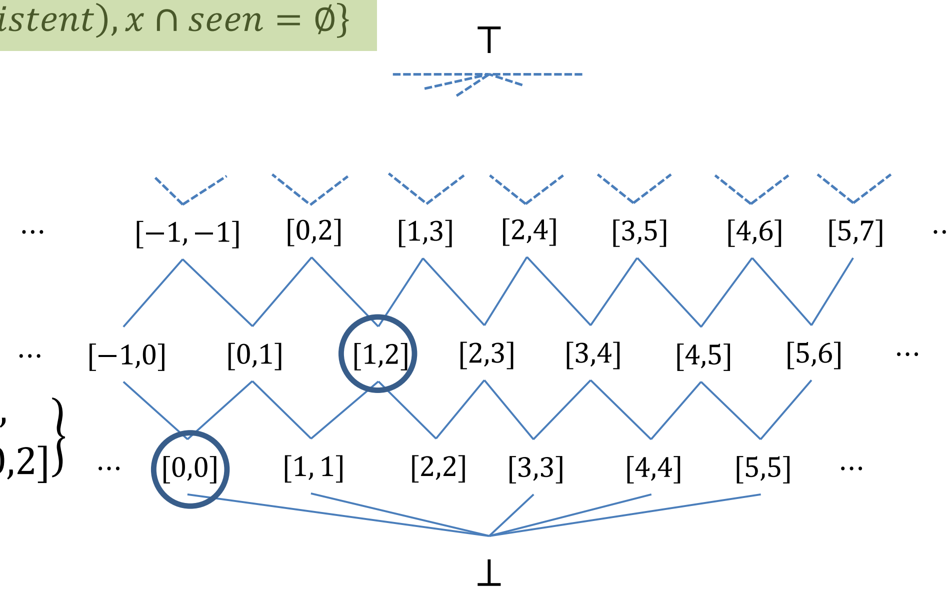
while $seen \neq A$ do:

$joint \leftarrow argmax_x\{card(x) | x \in dom(consistent), x \cap seen = \emptyset\}$

$seen \leftarrow seen \cup joint$

$res \leftarrow res \cup consistent(joint)$

return res



$$consistent = \begin{cases} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6], \{[0,0],[1,2]\} \mapsto [0,2] \end{cases}$$

$res = \emptyset$

$seen = \emptyset$

Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \ldots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

//prev slide

$res \leftarrow \emptyset$

$seen \leftarrow \emptyset$

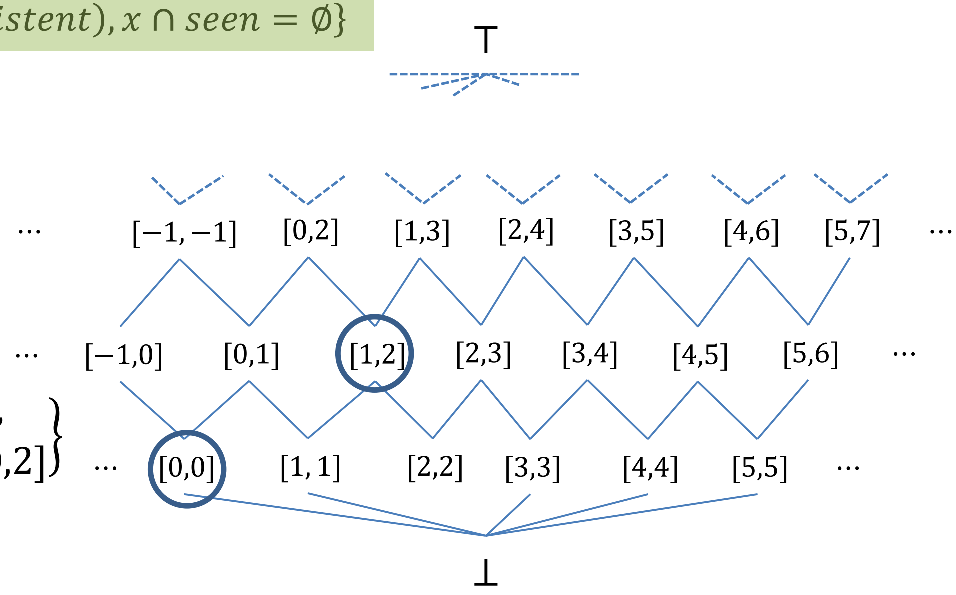
while $seen \neq A$ do:

$joint \leftarrow argmax_x\{card(x) | x \in dom(consistent), x \cap seen = \emptyset\}$

$seen \leftarrow seen \cup joint$

$res \leftarrow res \cup consistent(joint)$

return res



$$consistent = \left\{ \begin{matrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6], \{[0,0],[1,2]\} \mapsto [0,2] \end{matrix} \right\}$$

$res = \emptyset$

$seen = \emptyset$

$joint = \{[0,0],[1,2]\}$

Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \ldots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

//prev slide

$res \leftarrow \emptyset$

$seen \leftarrow \emptyset$

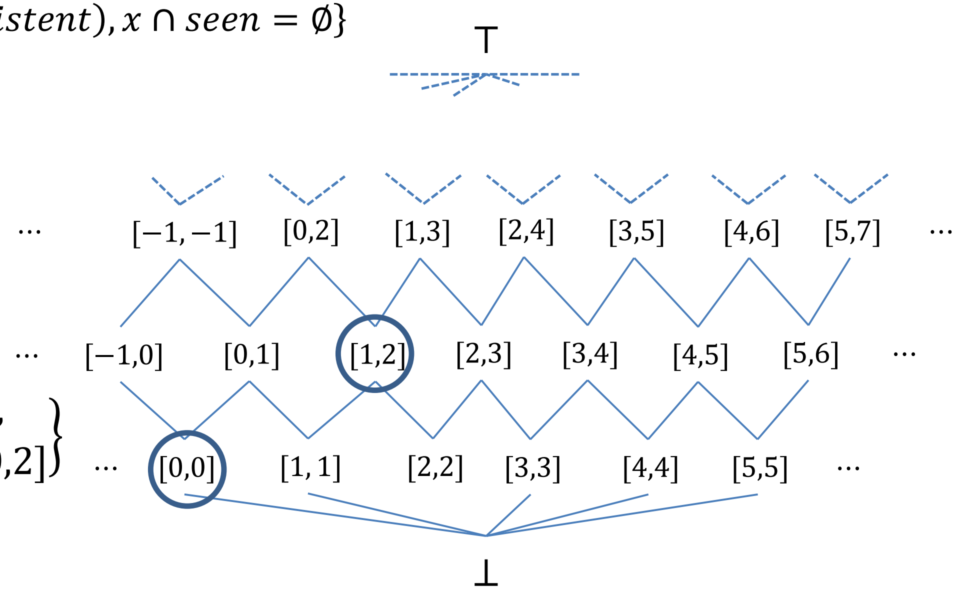
while $seen \neq A$ do:

$joint \leftarrow argmax_x\{card(x) | x \in dom(consistent), x \cap seen = \emptyset\}$

$seen \leftarrow seen \cup joint$

$res \leftarrow res \cup consistent(joint)$

return res



$$consistent = \begin{Bmatrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6], \{[0,0],[1,2]\} \mapsto [0,2] \end{Bmatrix}$$

$res = \emptyset$

$seen = \{[0,0],[1,2]\}$

$joint = \{[0,0],[1,2]\}$

Greedy Computation of SG

$SG(A = \{a_1, \ldots, a_k\}, C_{cex})$:

//prev slide

$res \leftarrow \emptyset$

$seen \leftarrow \emptyset$

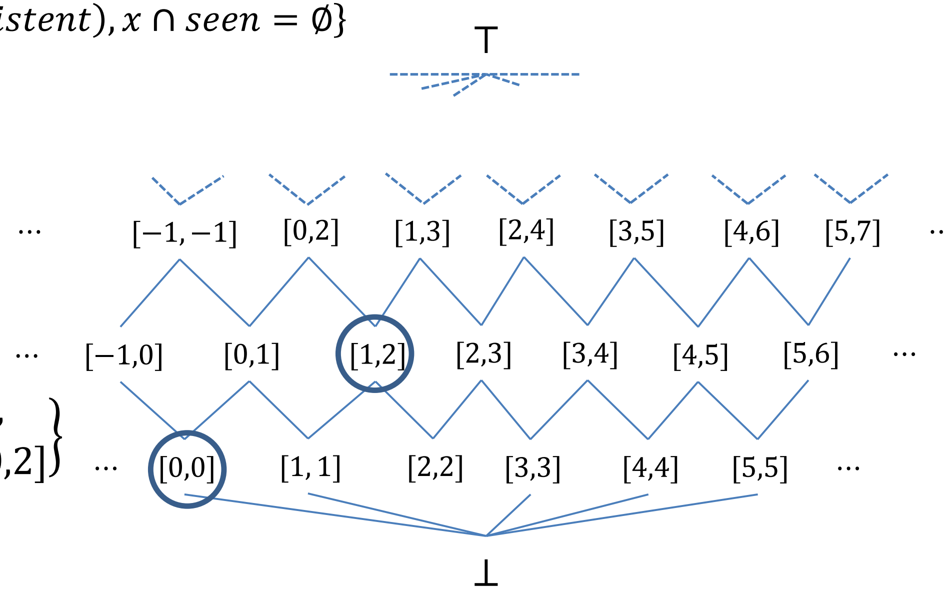
while $seen \neq A$ do:

$joint \leftarrow argmax_x\{card(x) | x \in dom(consistent), x \cap seen = \emptyset\}$

$seen \leftarrow seen \cup joint$

$res \leftarrow res \cup consistent(joint)$

return res



$$consistent = \begin{Bmatrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6], \{[0,0],[1,2]\} \mapsto [0,2] \end{Bmatrix}$$

$res = \emptyset$

$seen = \{[0,0],[1,2]\}$

$joint = \{[0,0],[1,2]\}$

Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \dots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

//prev slide

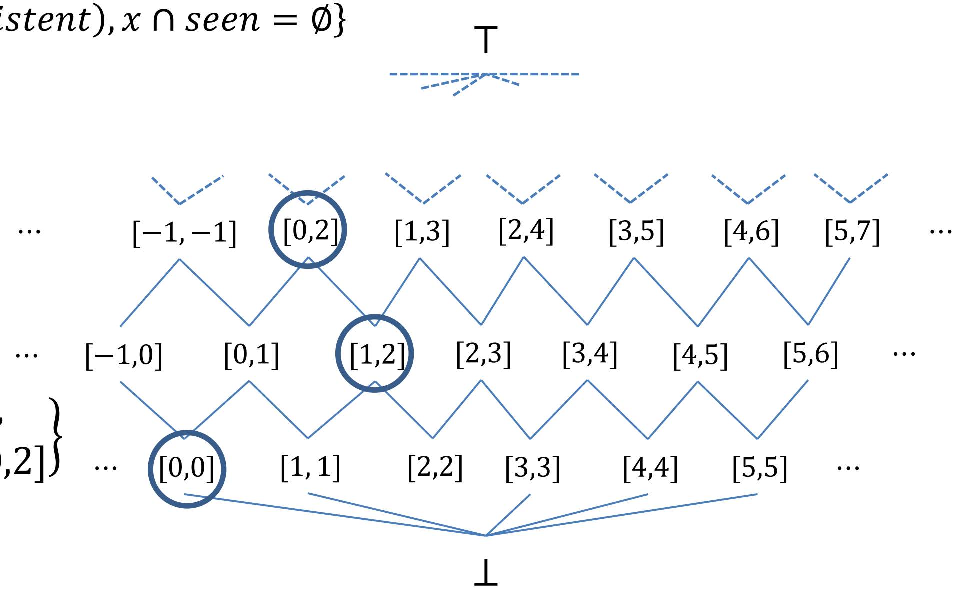
$res \leftarrow \emptyset$

$seen \leftarrow \emptyset$

while $seen \neq A$ do:

$joint \leftarrow argmax_x\{card(x) | x \in dom(consistent), x \cap seen = \emptyset\}$
$seen \leftarrow seen \cup joint$
$res \leftarrow res \cup consistent(joint)$

return res



$$consistent = \begin{Bmatrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6], \{[0,0],[1,2]\} \mapsto [0,2] \end{Bmatrix}$$

$res = \{[0,2]\}$
$seen = \{[0,0],[1,2]\}$

$joint = \{[0,0],[1,2]\}$

Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \ldots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

//prev slide

$res \leftarrow \emptyset$

$seen \leftarrow \emptyset$

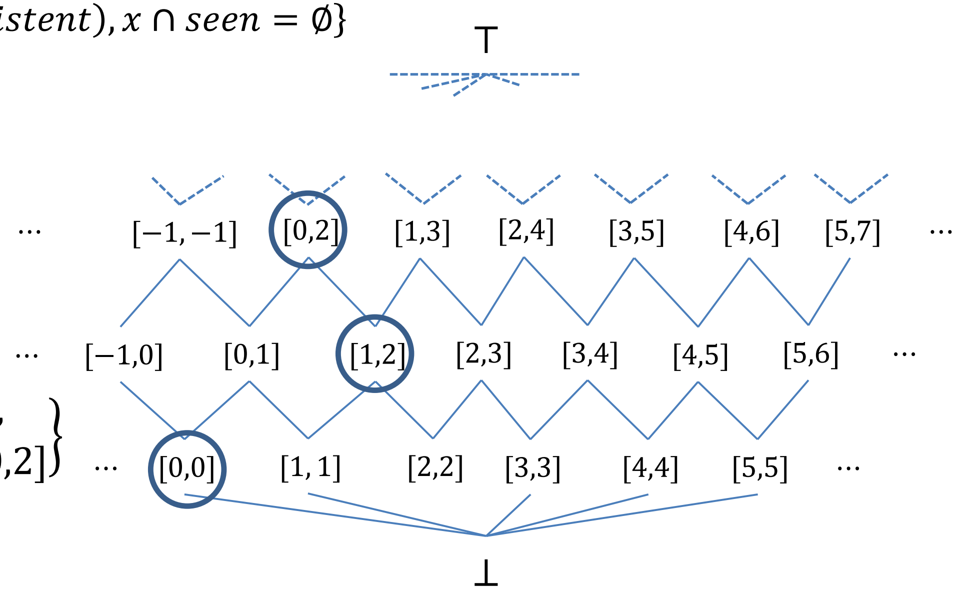
while $seen \neq A$ do:

$joint \leftarrow argmax_x\{card(x) | x \in dom(consistent), x \cap seen = \emptyset\}$

$seen \leftarrow seen \cup joint$

$res \leftarrow res \cup consistent(joint)$

return res



$$consistent = \begin{Bmatrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6], \{[0,0],[1,2]\} \mapsto [0,2] \end{Bmatrix}$$

$res = \{[0,2]\}$

$seen = \{[0,0],[1,2]\}$

Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \dots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

//prev slide

$res \leftarrow \emptyset$

$seen \leftarrow \emptyset$

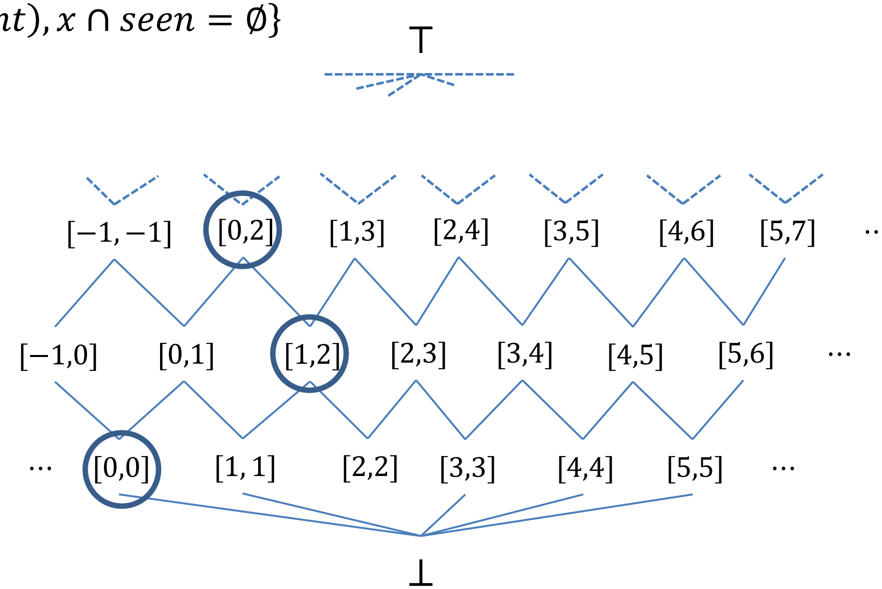
while $seen \neq A$ do:

$joint \leftarrow argmax_x\{card(x) | x \in dom(consistent), x \cap seen = \emptyset\}$

$seen \leftarrow seen \cup joint$

$res \leftarrow res \cup consistent(joint)$

return res



$$consistent = \left\{ \begin{array}{c} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6], \{[0,0],[1,2]\} \mapsto [0,2] \end{array} \right\}$$

$res = \{[0,2]\}$

$seen = \{[0,0],[1,2]\}$

$joint = \{[5,6]\}$

Greedy Computation of SG

$SG(A = \{a_1, \dots, a_k\}, C_{cex})$:

//prev slide

$res \leftarrow \emptyset$

$seen \leftarrow \emptyset$

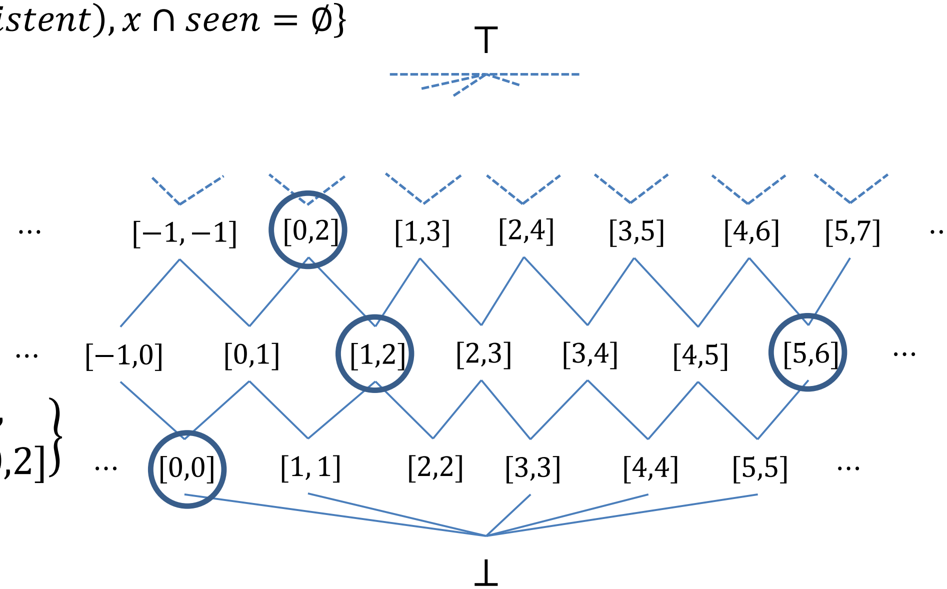
while $seen \neq A$ do:

$joint \leftarrow argmax_x\{card(x) | x \in dom(consistent), x \cap seen = \emptyset\}$

$seen \leftarrow seen \cup joint$

$res \leftarrow res \cup consistent(joint)$

return res



$$consistent = \left\{ \begin{matrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6], \{[0,0],[1,2]\} \mapsto [0,2] \end{matrix} \right\}$$

$res = \{[0,2],[5,6]\}$

$seen = \{[0,0],[1,2],[5,6]\}$

$joint = \{[5,6]\}$

Greedy Computation of SG

$\mathrm{SG}(\mathrm{A} = \{\mathrm{a}_1, \ldots, \mathrm{a}_\mathrm{k}\}, \mathrm{C}_{\mathrm{cex}})$:

//prev slide

$res \leftarrow \emptyset$

$seen \leftarrow \emptyset$

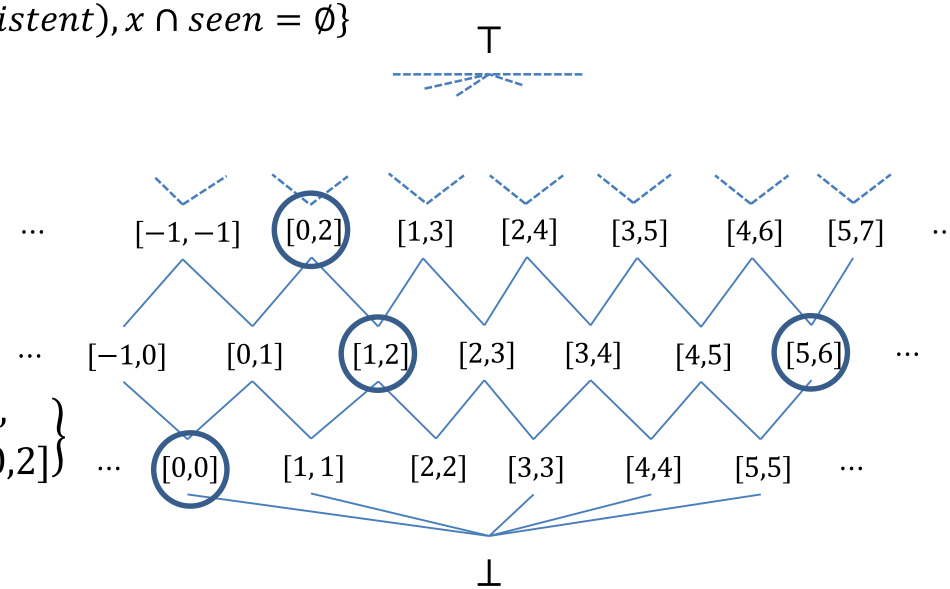
while $seen \neq A$ do:

$joint \leftarrow argmax_x\{card(x) | x \in dom(consistent), x \cap seen = \emptyset\}$

$seen \leftarrow seen \cup joint$

$res \leftarrow res \cup consistent(joint)$

return res



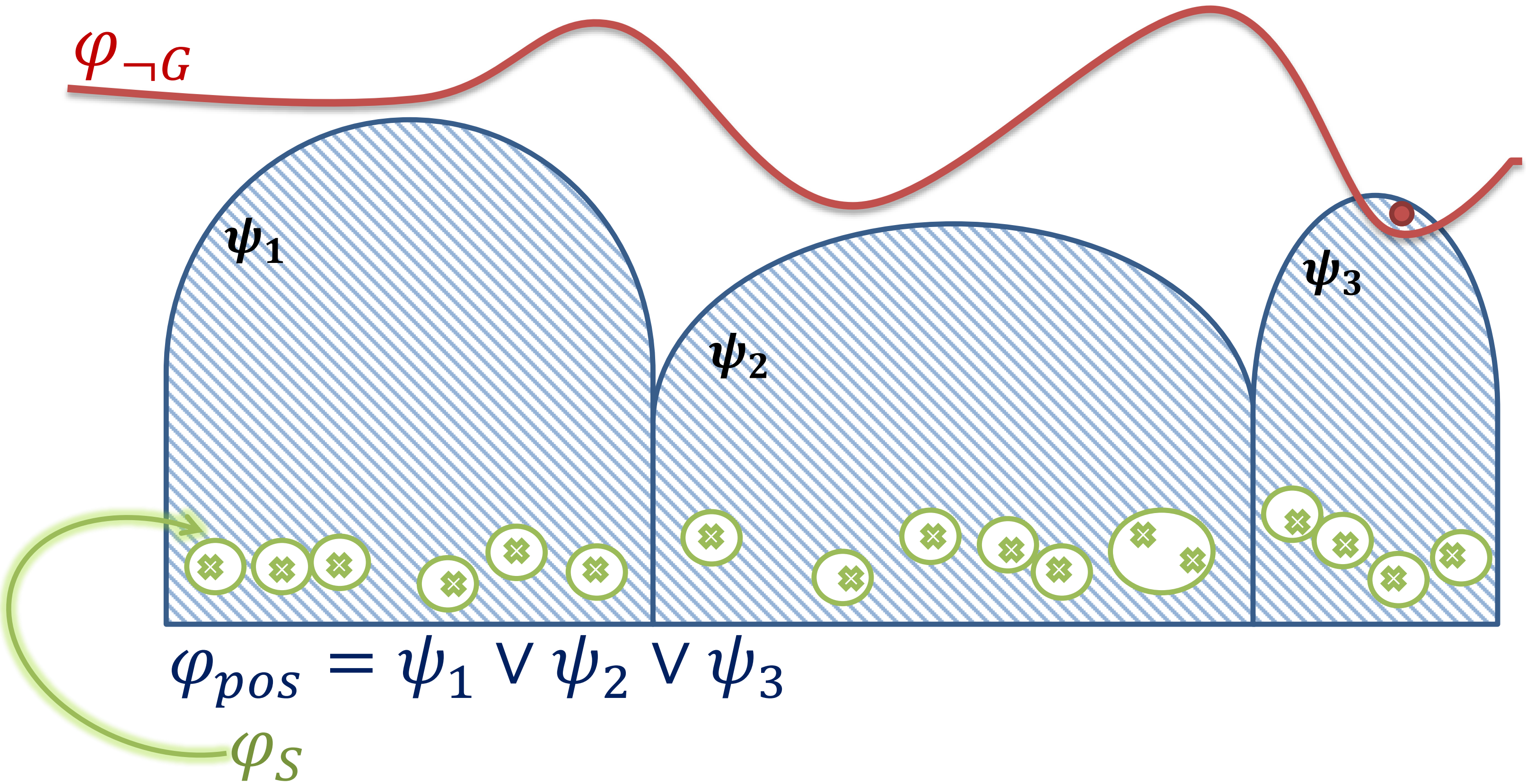
$$consistent = \left\{ \begin{matrix} \{[0,0]\} \mapsto [0,0], \{[1,2]\} \mapsto [1,2], \\ \{[5,6]\} \mapsto [5,6], \{[0,0],[1,2]\} \mapsto [0,2] \end{matrix} \right\}$$

$res = \{[0,2],[5,6]\}$

$seen = \{[0,0],[1,2],[5,6]\}$

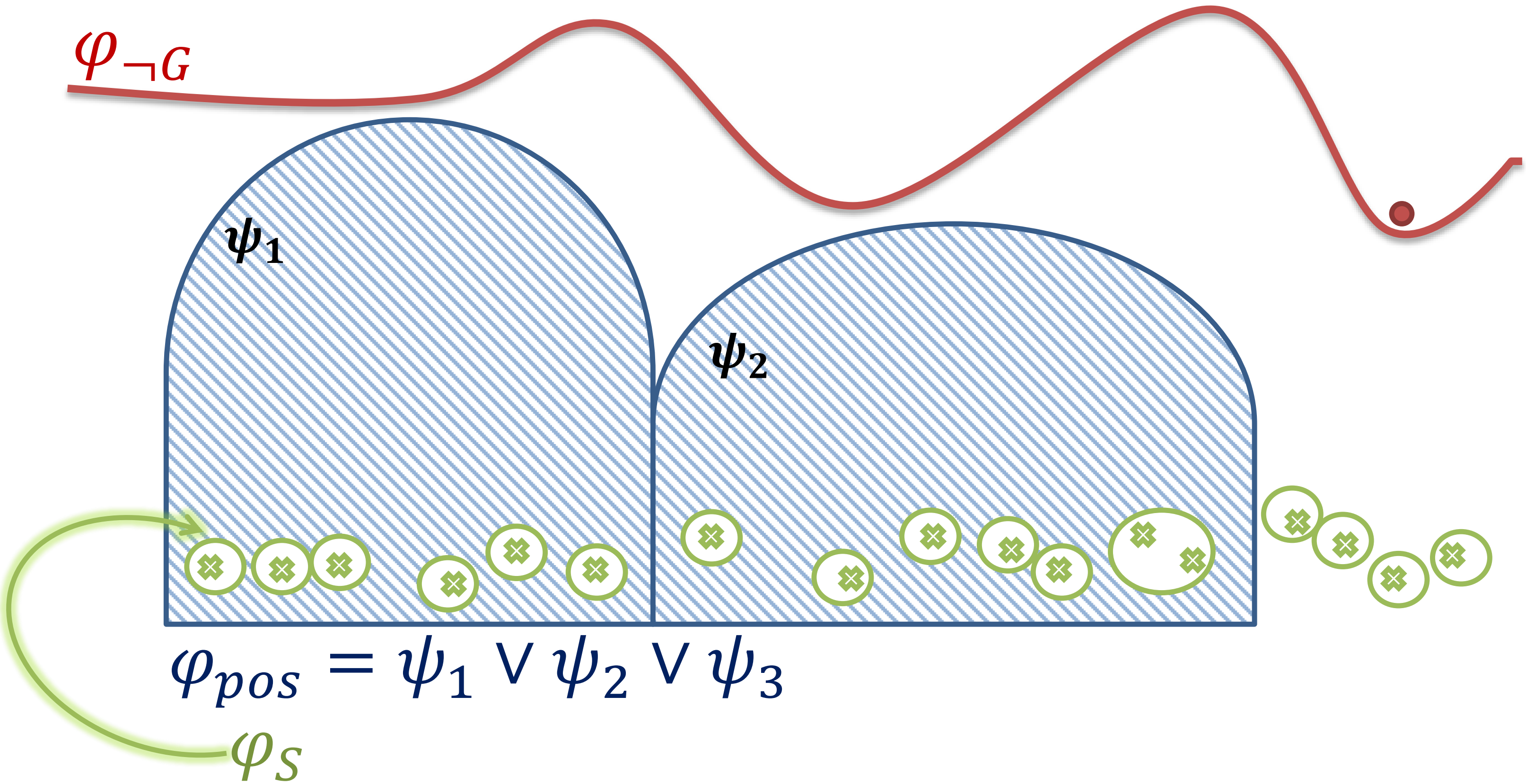
$joint = \{[5,6]\}$

Refinement



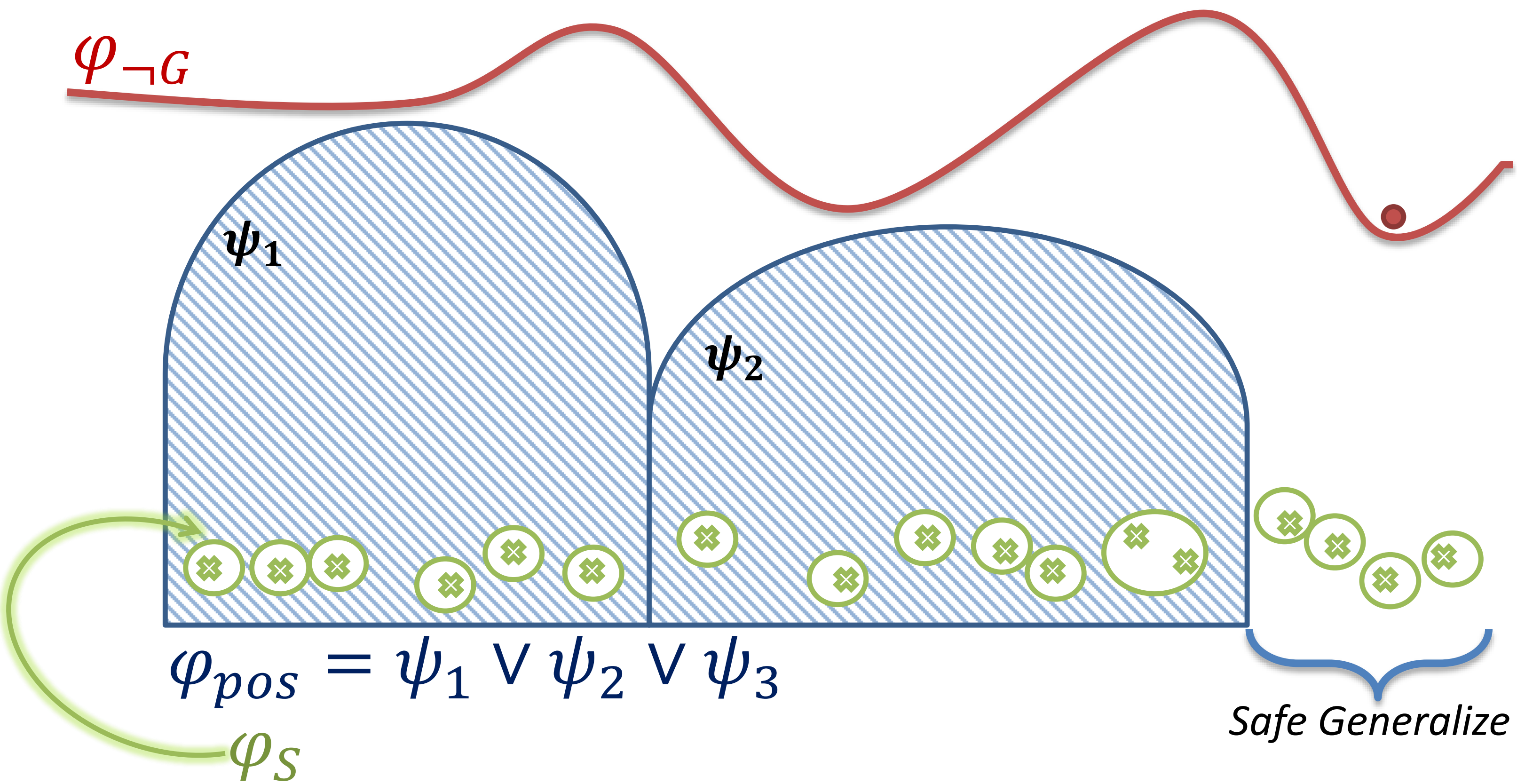
$\varphi_{\neg G}$
ψ_1
ψ_2
ψ_3
$\varphi_{pos} = \psi_1 \vee \psi_2 \vee \psi_3$
φ_S

Refinement



$\varphi_{\neg G}$
ψ_1
ψ_2
$\varphi_{pos} = \psi_1 \vee \psi_2 \vee \psi_3$
φ_S

Refinement



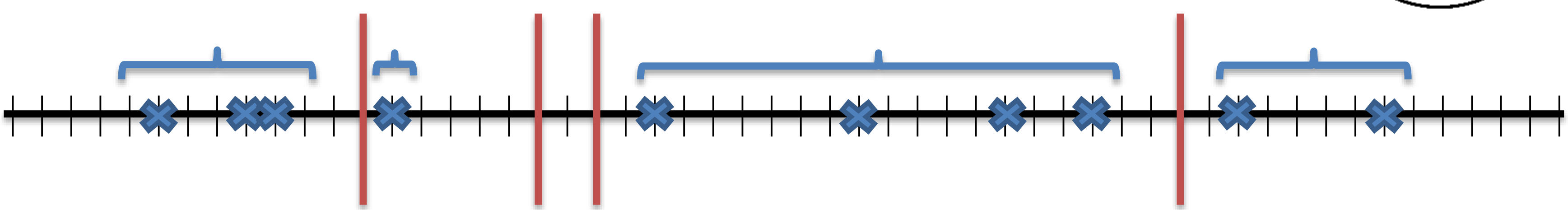
$\varphi_{\neg G}$
ψ_1
ψ_2
$\varphi_{pos} = \psi_1 \vee \psi_2 \vee \psi_3$
φ_S
Safe Generalize

Optimizing Refinement (Sometimes)

- If the domain allows for complementation
- Take a hint from *HYDRA* (Murray, 1987)
- For each formula, also keep partitions derived from negative samples
- Linear number of joins after refinement

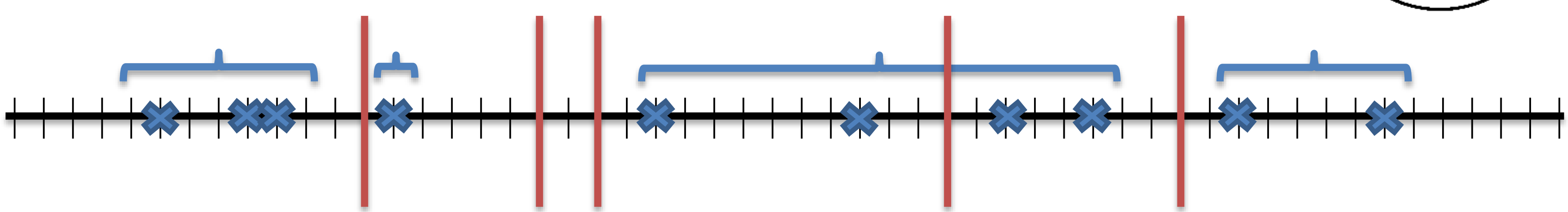
Optimizing Refinement (Sometimes)

- If the domain allows for complementation
- Take a hint from *HYDRA* (Murray, 1987)
- For each formula, also keep partitions derived from negative samples
- Linear number of joins after refinement



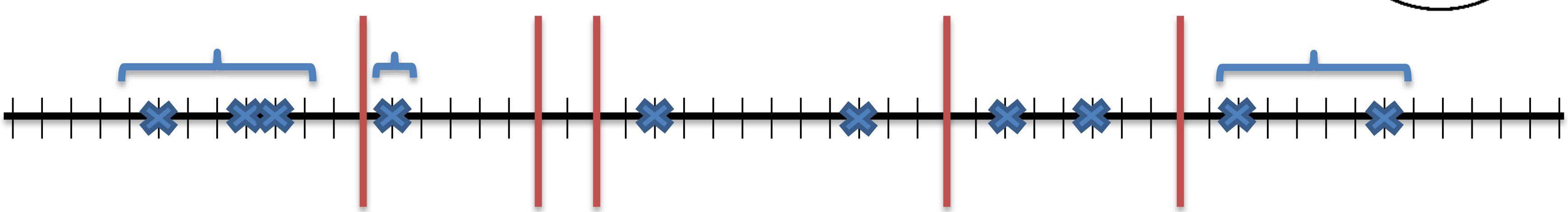
Optimizing Refinement (Sometimes)

- If the domain allows for complementation
- Take a hint from *Hydra* (Murray, 1987)
- For each formula, also keep partitions derived from negative samples
- Linear number of joins after refinement



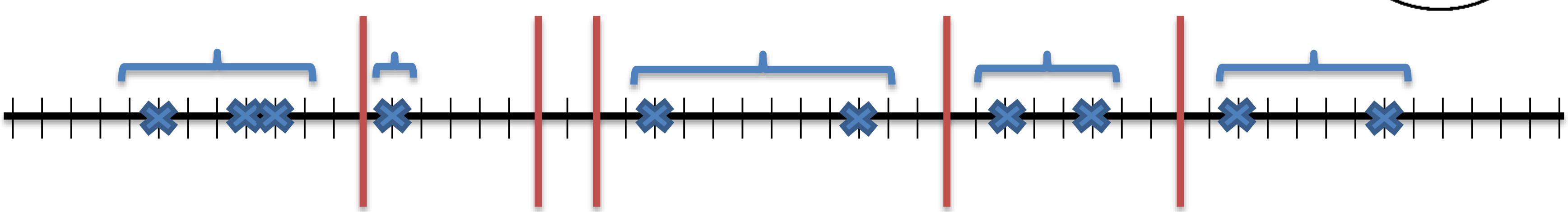
Optimizing Refinement (Sometimes)

- If the domain allows for complementation
- Take a hint from *HYDRA* (Murray, 1987)
- For each formula, also keep partitions derived from negative samples
- Linear number of joins after refinement

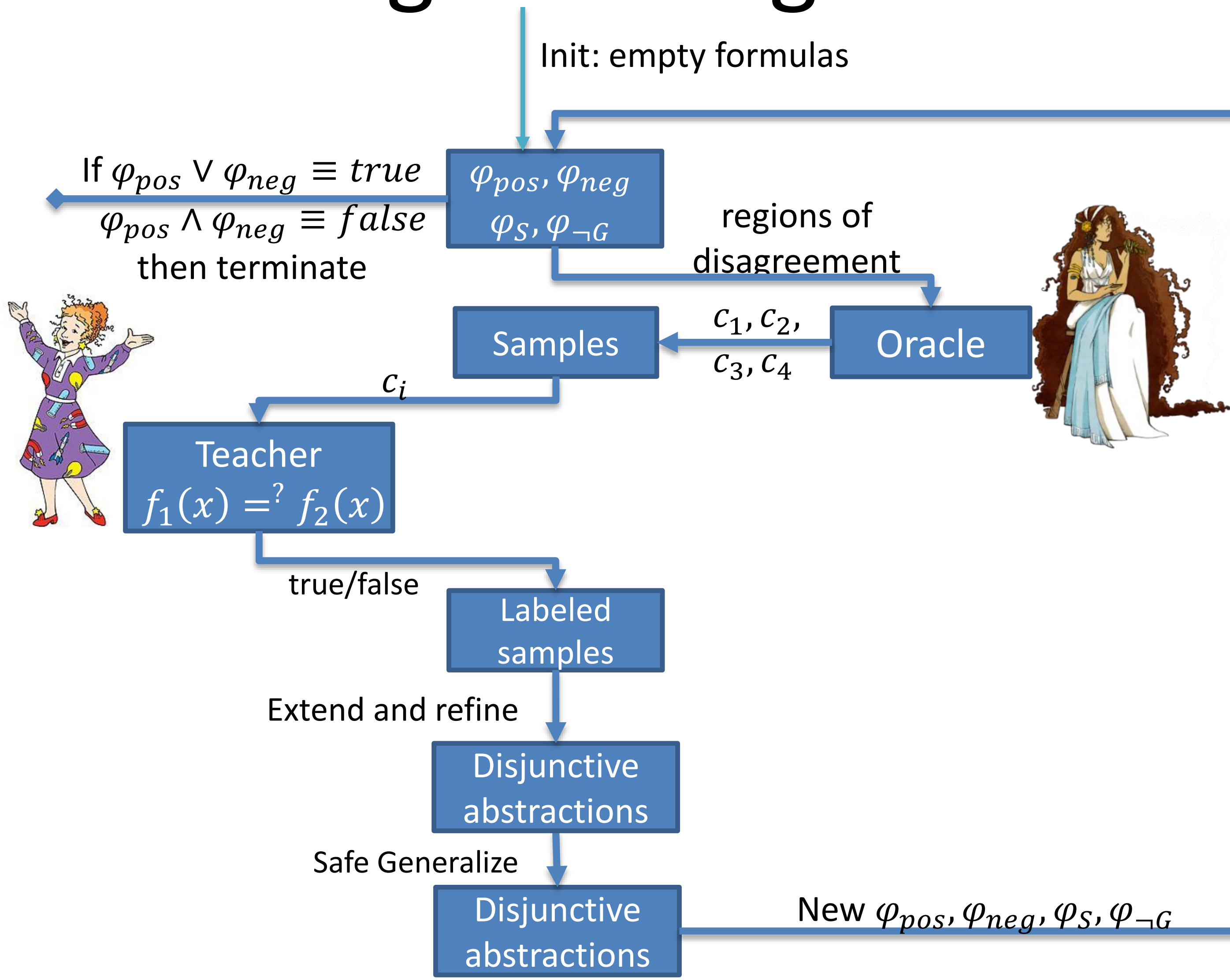


Optimizing Refinement (Sometimes)

- If the domain allows for complementation
- Take a hint from *HYDRA* (Murray, 1987)
- For each formula, also keep partitions derived from negative samples
- Linear number of joins after refinement



Putting It All Together



Data-Driven Disjunctive Abstraction

- Active learning algorithm based on Version Spaces
- *Safe Generalization*, which generalizes an element in the powerset domain while avoiding negative examples
- Implemented for differential analysis
  - Intervals
  - Intervals with congruence
  - Boxes
  - Quantified boolean predicates on arrays