Saturation problems and enumerating maximal solutions

Arnaud Mary[1] **Yann Strozecki**[2]

[1]Baobab, Lyon

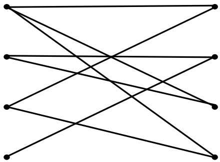
[2]Laboratoire DAVID, Versailles

Clermont-Ferrand, WEPA 2016

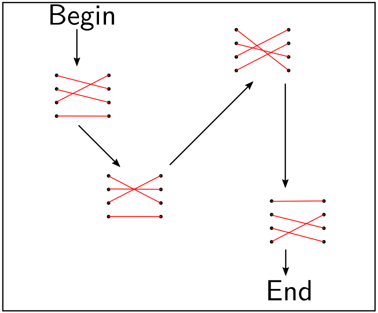
Enumeration problems

- Enumeration problems: list all solutions rather than just deciding whether there is one.
- Complexity measures: total time and delay between solutions.
- Motivations: database queries, optimization, building libraries.

Perfect matching ?



Solution space:



Framework

An enumeration problem A is a function which associates to each input a set of solutions $A(x)$.

An enumeration algorithm must generate every element of $A(x)$ one after the other without repetition.

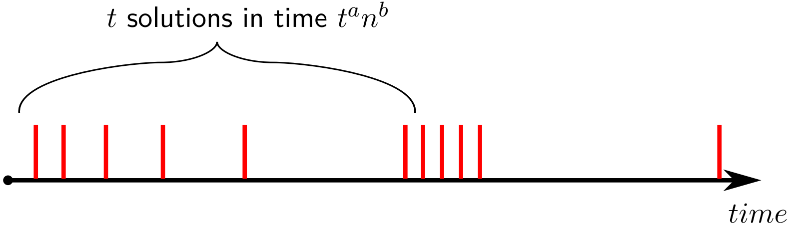
Complexity classes:
A polynomial time precomputation is allowed.

1. Polynomial total time: $\textsc{TotalP}$
2. Incremental polynomial time: $\textsc{IncP}$
3. Polynomial delay: $\textsc{DelayP}$

Incremental time

Definition (Incremental polynomial time)

IncP is the set of enumeration problems such that there is an algorithm which for all t produces t solutions (if they exist) from an input of size n in time $O(t^a n^b)$ with a, b constants.



Saturation algorithm

Most algorithms with an incremental delay are saturation algorithms:

- **begin** with a polynomial number of simple solutions
- **for each** k-uple of already generated solutions apply a rule to produce a new solution
- **stop** when no new solutions are found

Saturation algorithm

Most algorithms with an incremental delay are saturation algorithms:

- **begin** with a polynomial number of simple solutions
- **for each** k-uple of already generated solutions apply a rule to produce a new solution
- **stop** when no new solutions are found

1. Accessible vertices in a graph by flooding.
2. Generate a finite group from a set of generators.
3. Generate all possible unions of sets:

Saturation algorithm

Most algorithms with an incremental delay are saturation algorithms:

- **begin** with a polynomial number of simple solutions
- **for each** k-uple of already generated solutions apply a rule to produce a new solution
- **stop** when no new solutions are found

1. Accessible vertices in a graph by flooding.
2. Generate a finite group from a set of generators.
3. Generate all possible unions of sets:

   - $\{12, 134, 23, 14\}$
   - $\{12, 134, 1234, 23, 14\}$
   - $\{12, 134, 1234, 23, 123, 14\}$
   - $\{12, 134, 1234, 23, 123, 14, 124\}$

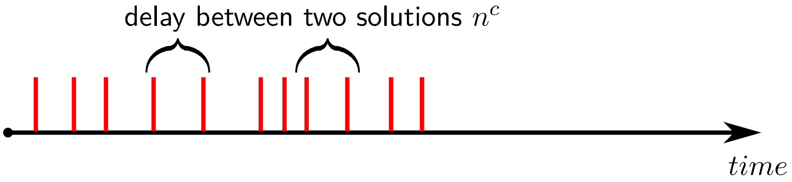
Polynomial Delay

The delay is the maximum time between the production of two consecutive solutions in an enumeration.

Definition (Polynomial delay)

DelayP is the set of enumeration problems such that there is an algorithm whose delay is polynomial in the input.

$$\textsc{DelayP} \subseteq \textsc{IncP}$$



Unions in polynomial delay

Closure by union revisited.
Instance: a set $S = \{s_1, \ldots s_m\}$ with $s_i \subseteq \{1, \ldots, n\}$.
Problem: generate all unions of elements in S.

Unions in polynomial delay

Closure by union revisited.
Instance: a set $S = \{s_1, \dots s_m\}$ with $s_i \subseteq \{1, \dots, n\}$.
Problem: generate all unions of elements in S.

1. Recursive strategy, enumerate first the solutions which contains 1, then those which do not contain 1.

Unions in polynomial delay

Closure by union revisited.
Instance: a set $S = \{s_1, \dots s_m\}$ with $s_i \subseteq \{1, \dots, n\}$.
Problem: generate all unions of elements in S.

1. Recursive strategy, enumerate first the solutions which contains 1, then those which do not contain 1.
2. The algorithm should not explore a branch without solutions (flashlight search), so that we can bound the delay.

Unions in polynomial delay

Closure by union revisited.
Instance: a set $S = \{s_1, \ldots s_m\}$ with $s_i \subseteq \{1, \ldots, n\}$.
Problem: generate all unions of elements in S.

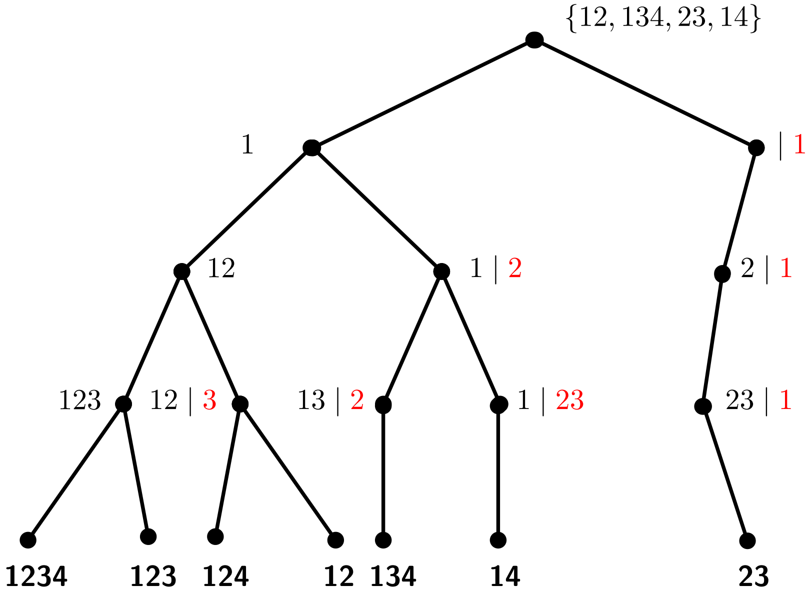
1. Recursive strategy, enumerate first the solutions which contains 1, then those which do not contain 1.
2. The algorithm should not explore a branch without solutions (flashlight search), so that we can bound the delay.
3. We must solve the extension problem: given two sets A and B is there a **solution** S **such that** $A \subseteq S$ **and** $S \cap B = \emptyset$?

Unions in polynomial delay

Closure by union revisited.
Instance: a set $S = \{s_1, \dots s_m\}$ with $s_i \subseteq \{1, \dots, n\}$.
Problem: generate all unions of elements in S.

1. Recursive strategy, enumerate first the solutions which contains 1, then those which do not contain 1.
2. The algorithm should not explore a branch without solutions (flashlight search), so that we can bound the delay.
3. We must solve the extension problem: given two sets A and B is there a **solution** S **such that** $A \subseteq S$ **and** $S \cap B = \emptyset$?
4. The extension problem is easy to solve in time $O(mn)$ thus the backtrack search has delay $O(mn^2)$.

Partial solution tree



From saturation to polynomial delay

Question

Can we solve saturation problems with a polynomial delay ?

From saturation to polynomial delay

Question

Can we solve saturation problems with a polynomial delay ?

No, since saturation problems are "equals" to $\textsc{IncP}$ and we have recently proved $\textsc{IncP} \neq \textsc{DelayP}$.

From saturation to polynomial delay

Question

Can we solve saturation problems with a polynomial delay ?

No, since saturation problems are "equals" to $\textsc{IncP}$ and we have recently proved $\textsc{IncP} \neq \textsc{DelayP}$.

We need to restrict the saturation rules. Since it works for the union, we will consider set operations.

Our goal is twofold:

- design a large toolbox of efficient enumeration algorithms
- classify the easy and the not so easy problems

Set operations

A set over $\{1, \ldots, n\}$ will be represented by its characteristic vector of size n.

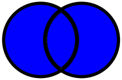
A set operation is a boolean operation $\{0,1\}^k \to \{0,1\}$ applied componentwise to k boolean vectors.

Set operations

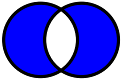
A set over $\{1, \ldots, n\}$ will be represented by its characteristic vector of size n.

A set operation is a boolean operation $\{0,1\}^k \to \{0,1\}$ applied componentwise to k boolean vectors.

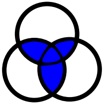
$\vee$ $\quad \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \vee \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad$ $\cup$



$+$ $\quad \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \quad$ $\triangle$



$\mathrm{maj}(x,y,z)$ $\quad \mathrm{maj}(\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad$ Majority



Closure by set operation

Let $\mathcal{S}$ be a set of boolean vectors of size n and $\mathcal{F}$ be a finite set of boolean operations.

Closure:

- $\mathcal{F}^0(\mathcal{S}) = \mathcal{S}$
- $\mathcal{F}^i(\mathcal{S}) = \mathcal{F}^{i-1}(\mathcal{S}) \cup$
  $\{f(v_1, \ldots, v_t) \mid v_1, \ldots, v_t \in \mathcal{F}^{i-1}(S)$ and $f \in \mathcal{F}\}$
- $Cl_{\mathcal{F}}(\mathcal{S}) = \cup_i \mathcal{F}^i(\mathcal{S})$

Closure by set operation

Let $\mathcal{S}$ be a set of boolean vectors of size n and $\mathcal{F}$ be a finite set of boolean operations.

Closure:

- $\mathcal{F}^0(\mathcal{S}) = \mathcal{S}$
- $\mathcal{F}^i(\mathcal{S}) = \mathcal{F}^{i-1}(\mathcal{S}) \cup$
  $\{f(v_1, \ldots, v_t) \mid v_1, \ldots, v_t \in \mathcal{F}^{i-1}(S)$ and $f \in \mathcal{F}\}$
- $Cl_{\mathcal{F}}(\mathcal{S}) = \cup_i \mathcal{F}^i(\mathcal{S})$

Our enumeration problem is then to list the elements of $\mathrm{C}l_{\mathcal{F}}(\mathcal{S})$.

Extension problem

$\textsc{Closure}_{\mathcal{F}}$:

Input: $\mathcal{S}$ a set of vectors of size n, and a vector v of size n
Problem: decide whether $v \in Cl_{\mathcal{F}}(\mathcal{S})$.

$\textsc{Closure}_{\mathcal{F}}$ is the extension problem associated to the computation of $Cl_{\mathcal{F}}(\mathcal{S})$.

Extension problem

$\textsc{Closure}_{\mathcal{F}}$:

Input: $\mathcal{S}$ a set of vectors of size n, and a vector v of size n
Problem: decide whether $v \in Cl_{\mathcal{F}}(\mathcal{S})$.

$\textsc{Closure}_{\mathcal{F}}$ is the extension problem associated to the computation of $Cl_{\mathcal{F}}(\mathcal{S})$.

Goal: prove that $\textbf{Closure}_{\mathcal{F}} \in \mathsf{P}$ for as many sets $\mathcal{F}$ as possible, to use the backtrack search.

Clones and reduction

There are many finite families of boolean operations, how to reduce their number ?

Clones and reduction

There are many finite families of boolean operations, how to reduce their number ?

Definition

Let $\mathcal{F}$ be a finite set of operations, the functional clone generated by $\mathcal{F}$, denoted by $< \mathcal{F} >$, is the set of operations obtained by any composition of the operations of $\mathcal{F}$ and of the projections π_k^n defined by $\pi_k^n(x_1, \ldots, x_n) = x_k$.

For instance $(x \vee y) + x + z \ \in < \vee, + >$.

Clones and reduction

There are many finite families of boolean operations, how to reduce their number ?

Definition

Let $\mathcal{F}$ be a finite set of operations, the functional clone generated by $\mathcal{F}$, denoted by $< \mathcal{F} >$, is the set of operations obtained by any composition of the operations of $\mathcal{F}$ and of the projections π_k^n defined by $\pi_k^n(x_1, \ldots, x_n) = x_k$.

For instance $(x \vee y) + x + z \in < \vee, + >$.

Lemma

For all set of operations $\mathcal{F}$ and all set of vectors $\mathcal{S}$, $Cl_{\mathcal{F}}(\mathcal{S}) = Cl_{<\mathcal{F}>}(\mathcal{S})$.

Clones and reduction

There are many finite families of boolean operations, how to reduce their number ?

Definition

Let $\mathcal{F}$ be a finite set of operations, the functional clone generated by $\mathcal{F}$, denoted by $< \mathcal{F} >$, is the set of operations obtained by any composition of the operations of $\mathcal{F}$ and of the projections π_k^n defined by $\pi_k^n(x_1, \dots, x_n) = x_k$.

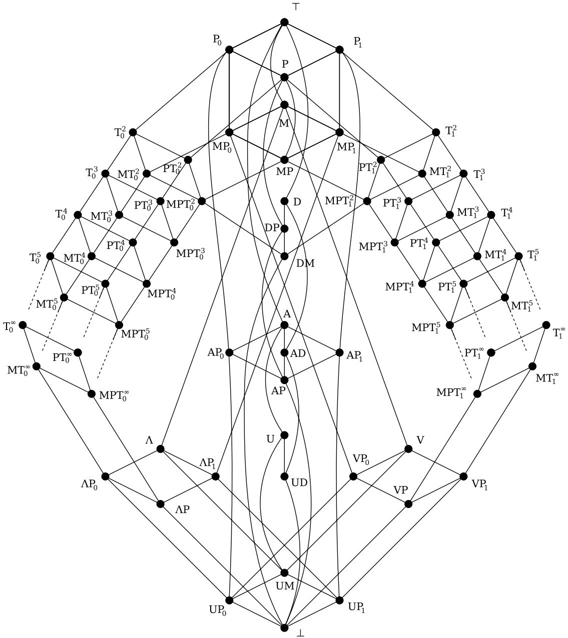
For instance $(x \vee y) + x + z \in < \vee, + >$.

Lemma

For all set of operations $\mathcal{F}$ and all set of vectors $\mathcal{S}$,
$Cl_{\mathcal{F}}(\mathcal{S}) = Cl_{<\mathcal{F}>}(\mathcal{S})$.

There are less clones than families and they are well described and organized in Post's lattice.

Post's lattice



How to reduce Post's lattice

To an operation f we can associate its dual $\overline{f}$ defined by $\overline{f}(s_1, \ldots, s_t) = \neg f(\neg s_1, \ldots, \neg s_t)$.

Proposition

The following problems can be polynomially reduced to $\textsc{Closure}_{\mathcal{F}}$*:*

1. $\textsc{Closure}_{\overline{\mathcal{F}}}$
2. $\textsc{Closure}_{\mathcal{F} \cup \{\neg\}}$ *when* $\mathcal{F} = \overline{\mathcal{F}}$
3. $\textsc{Closure}_{\mathcal{F} \cup \{\mathbf{0}\}}$, $\textsc{Closure}_{\mathcal{F} \cup \{\mathbf{1}\}}$, $\textsc{Closure}_{\mathcal{F} \cup \{\mathbf{0},\mathbf{1}\}}$

Reduced Post's lattice

Clone	Base
I_2	$\emptyset$
L_2	$x + y + z$
L_0	$+$
E_2	$\wedge$
S_{10}	$x \wedge (y \vee z)$
S_{10}^k	$Th_k^{k+1}, x \wedge (y \vee z)$
S_{12}	$x \wedge (y \rightarrow z)$
S_{12}^k	$Th_k^{k+1}, x \wedge (y \rightarrow z)$
D_2	maj
D_1	maj, $x + y + z$
M_2	$\vee, \wedge$
R_2	$x \,?\, y \,:\, z$
R_0	$\vee, +$

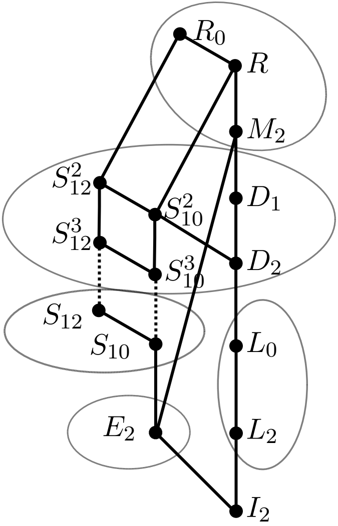


Figure: Reduced Post's lattice, the edges represent inclusions of clones

The result

Theorem

For all sets $\mathcal{F}$ of boolean operations, $\textsc{Closure}_{\mathcal{F}} \in \mathsf{P}$.

Corollary

For all sets $\mathcal{F}$ of boolean operations, enumerating $Cl_{\mathcal{F}}$ is in $\textsc{DelayP}$.

The result

Theorem

For all sets $\mathcal{F}$ of boolean operations, $\textsc{Closure}_{\mathcal{F}} \in \mathsf{P}$.

Corollary

For all sets $\mathcal{F}$ of boolean operations, enumerating $Cl_{\mathcal{F}}$ is in $\textsc{DelayP}$.

We succeeded and we failed: everything is easy but everything is the same.

Five families

1. Conjunction is easy, a simple combinatorial algorithm.
2. $< + >$ (vector space) and $< \vee, \neg >$ (boolean algebra) are easy because of their algebraic structure. You can compute a basis of the solutions in polynomial time.
3. $< \mathrm{maj} >$ is easy because only projection of size two matters. Another form of algebraic structure.

Majority

Proposition

Let $\mathcal{S}$ be a vector set, a vector v belongs to $Cl_{<\mathrm{maj}>}(\mathcal{S})$ if and only if for all $i, j \in [n]$, $i \neq j$, there exists $x \in \mathcal{S}$ such that $x_{i,j} = v_{i,j}$.

Idea of the proof: you build incrementally the vector v by using a sequence of vectors which have the same pairs as v.

Majority

Proposition

Let $\mathcal{S}$ be a vector set, a vector v belongs to $Cl_{<\mathrm{maj}>}(\mathcal{S})$ if and only if for all $i, j \in [n]$, $i \neq j$, there exists $x \in \mathcal{S}$ such that $x_{i,j} = v_{i,j}$.

Idea of the proof: you build incrementally the vector v by using a sequence of vectors which have the same pairs as v.

The same phenomenon is true as soon as there is a near unanimity term in the clone by the Baker-Pixley theorem. If the term is of arity k, you need to consider all projections of size $k-1$.

Larger domains

What does not work:

- The lattice of clones is uncountable and not well described.
- Over $D = \{0, 1, 2\}$, let $f(x, y) = x + y$ when $x + y <= 2$ otherwise $f(x, y) = 2$. $\textsc{Closure}_{<f>}$ is NP-hard.

Larger domains

What does not work:

- The lattice of clones is uncountable and not well described.
- Over $D = \{0, 1, 2\}$, let $f(x, y) = x + y$ when $x + y <= 2$ otherwise $f(x, y) = 2$. $\textsc{Closure}_{<f>}$ is NP-hard.

What does work:

- Near unanimity.
- Field, ring, group, some semi-group via the extension problem (subpower membership problem).
- Associative binary operations with an alternative algorithm and exponential space.

Generalizing the closure operation

Two remarks:

- Interesting set systems closed under inclusion, that is if $A \in \mathcal{S}$, $A \subseteq B \Rightarrow B \in \mathcal{S}$.
- All our framework relies on the fact that the closure operators act componentwise.

Generalizing the closure operation

Two remarks:

- Interesting set systems closed under inclusion, that is if $A \in \mathcal{S}$, $A \subseteq B \Rightarrow B \in \mathcal{S}$.
- All our framework relies on the fact that the closure operators act componentwise.

Let us combine those two remarks and study the operator $\uparrow \mathcal{S} = \{B \mid A \subseteq B, A \in \mathcal{S}\}$ which does not act componentwise.

Generalizing the closure operation

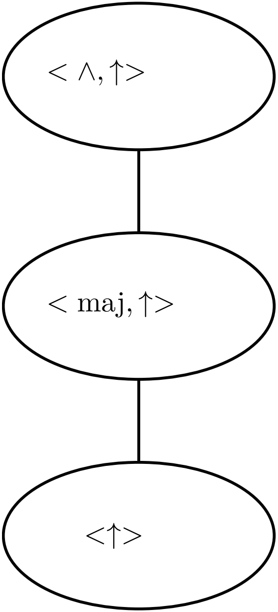
Two remarks:

- Interesting set systems closed under inclusion, that is if $A \in \mathcal{S}$, $A \subseteq B \Rightarrow B \in \mathcal{S}$.
- All our framework relies on the fact that the closure operators act componentwise.

Let us combine those two remarks and study the operator $\uparrow \mathcal{S} = \{B \mid A \subseteq B, A \in \mathcal{S}\}$ which does not act componentwise.

If we allow any set operation and $\uparrow$, the Post's lattice changes dramatically. Consider $< \wedge, \uparrow >$, $< \neg, \uparrow >$, $< +, \uparrow >$.

A simpler picture



Complexity of the three classes

We have only three different cases :

- $Cl_{<\uparrow>}$ is equivalent to enumerate the models of a monotone DNF formula. Can be done with delay $O(n|\mathcal{S}|)$.

Complexity of the three classes

We have only three different cases :

- $Cl_{<\uparrow>}$ is equivalent to enumerate the models of a monotone DNF formula. Can be done with delay $O(n|\mathcal{S}|)$.
- $Cl_{<\mathcal{F},\uparrow>}$ where $\mathcal{F}$ contains a near unanimity term of arity k can be solved with delay $O(n^{k-1})$. Just apply $\uparrow$ to the projections.

Complexity of the three classes

We have only three different cases :

- $Cl_{<\uparrow>}$ is equivalent to enumerate the models of a monotone DNF formula. Can be done with delay $O(n|\mathcal{S}|)$.
- $Cl_{<\mathcal{F},\uparrow>}$ where $\mathcal{F}$ contains a near unanimity term of arity k can be solved with delay $O(n^{k-1})$. Just apply $\uparrow$ to the projections.
- $Cl_{<\wedge,\uparrow>}$, one can compute the minimum solution and then generate all solutions by Gray code enumeration with delay $O(1)$.

Complexity of the three classes

We have only three different cases :

- $Cl_{<\uparrow>}$ is equivalent to enumerate the models of a monotone DNF formula. Can be done with delay $O(n|\mathcal{S}|)$.
- $Cl_{<\mathcal{F},\uparrow>}$ where $\mathcal{F}$ contains a near unanimity term of arity k can be solved with delay $O(n^{k-1})$. Just apply $\uparrow$ to the projections.
- $Cl_{<\wedge,\uparrow>}$, one can compute the minimum solution and then generate all solutions by Gray code enumeration with delay $O(1)$.

Open question: Can we get rid of the $O(|\mathcal{S}|)$ in the algorithm sovling $Cl_{<\uparrow>}$ or $Cl_{<\cup>}$?

Another take on $\uparrow$

Apply $\uparrow$ only once after taking the closure by some clone. It amounts to enumerate $\uparrow Cl_{\mathcal{F}}(\mathcal{S})$.
No interaction between the operator $\uparrow$ and the operators in $\mathcal{F}$: less degenerate problems.

Another take on $\uparrow$

Apply $\uparrow$ only once after taking the closure by some clone. It amounts to enumerate $\uparrow Cl_{\mathcal{F}}(\mathcal{S})$.
No interaction between the operator $\uparrow$ and the operators in $\mathcal{F}$: less degenerate problems.

Using extension problem and a variant of backtrack search, everything is in polynomial delay.

Another take on $\uparrow$

Apply $\uparrow$ only once after taking the closure by some clone. It amounts to enumerate $\uparrow Cl_{\mathcal{F}}(\mathcal{S})$.
No interaction between the operator $\uparrow$ and the operators in $\mathcal{F}$: less degenerate problems.

Using extension problem and a variant of backtrack search, everything is in polynomial delay.

Let $Min(\mathcal{S}) = \{s \in \mathcal{S} \mid \forall e \neq s \in \mathcal{S}, s \subsetneq e\}$.
The previous problem is equivalent to generate $\uparrow Min(Cl_{\mathcal{F}}(\mathcal{S}))$.
Efficient algorithm when $Min(Cl_{\mathcal{F}}(\mathcal{S}))$ is small.

Minimal solutions

We want algorithms to enumerate $Min(Cl_{\mathcal{F}}(\mathcal{S}))$.

Easy cases:

- the clone contains $\cap$: only one element.
- the operators are increasing, compute the minimal elements of the input.

Interesting cases:

- $< + >$, the circuits of a binary matroid, incremental polynomial. Extension problem NP-hard.
- $< x + y + z >$, is it different from $< + >$?
- $< \mathrm{maj} >$ and the other classes with a near unanimity: good algorithms.

How to solve $< \mathrm{maj} >$

Let $\mathcal{S}_{[i]}$ be the set of vectors in $\mathcal{S}$ restricted to the first i coordinates.

General idea:

- Compute the set $Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i]}))$ from the set $Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i-1]}))$ iteratively until $i = n$.

How to solve $< \mathrm{maj} >$

Let $\mathcal{S}_{[i]}$ be the set of vectors in $\mathcal{S}$ restricted to the first i coordinates.

General idea:

- Compute the set $Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i]}))$ from the set $Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i-1]}))$ iteratively until $i = n$.
- Do a DFS to avoid redundancy and obtain a polynomial delay and space.

How to solve $< \mathrm{maj} >$

Let $\mathcal{S}_{[i]}$ be the set of vectors in $\mathcal{S}$ restricted to the first i coordinates.

General idea:

- Compute the set $Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i]}))$ from the set $Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i-1]}))$ iteratively until $i = n$.
- Do a DFS to avoid redundancy and obtain a polynomial delay and space.
- We should define the ancestor of a solution as in reverse search.

Ancestor

Let's take a vector $v \in Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i]}))$ and let's define its ancestor $v' \in Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i-1]}))$.

Ancestor

Let's take a vector $v \in Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i]}))$ and let's define its ancestor $v' \in Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i-1]}))$.

If $\mathbf{v_i} = \mathbf{1}$

$$v : \begin{array}{c} {\scriptstyle 1} \\ {\scriptstyle 2} \\ {\scriptstyle 3} \\ \vdots \\ {\scriptstyle i-2} \\ {\scriptstyle i-1} \\ {\scriptstyle i} \end{array} \begin{pmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

Ancestor

Let's take a vector $v \in Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i]}))$ and let's define its ancestor $v' \in Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i-1]}))$.

If $\mathbf{v_i} = \mathbf{1}$

$$v : \begin{array}{c} 1 \\ 2 \\ 3 \\ \vdots \\ i-2 \\ i-1 \\ i \end{array} \begin{pmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 1 \\ 1 \end{pmatrix} \longrightarrow \begin{array}{c} \begin{pmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \\ 1 \end{array} \quad v_{[i-1]} \text{ is minimal in } Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i-1]})$$

Otherwise it would contradict the minimality of v in $Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i]})$

Case $\mathbf{v_i = 1}$

v will be obtained from $v_{[i-1]} \in Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i-1]}))$ by appending a "1" at the i$^{\text{th}}$ coordinate.

Case $\mathbf{v_i = 0}$

$$v: \quad \begin{array}{c} {\scriptstyle 1} \\ {\scriptstyle 2} \\ {\scriptstyle 3} \\ \vdots \\ {\scriptstyle i-2} \\ {\scriptstyle i-1} \\ {\scriptstyle i} \end{array} \begin{pmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 1 \\ \color{red}{0} \end{pmatrix}$$

Case $\mathbf{v_i = 0}$

$$v : \quad \begin{array}{c c} {\scriptstyle 1} & \\ {\scriptstyle 2} & \\ {\scriptstyle 3} & \\ \vdots & \\ {\scriptstyle i-2} & \\ {\scriptstyle i-1} & \\ {\scriptstyle i} & \end{array} \!\!\! \begin{pmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 1 \\ 0 \end{pmatrix} \longrightarrow v_{[i-1]} \text{ need not be in } Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i-1]})).$$

Case $\mathbf{v_i = 0}$

$$v : \begin{array}{c} {\scriptstyle 1} \\ {\scriptstyle 2} \\ {\scriptstyle 3} \\ \vdots \\ {\scriptstyle i-2} \\ {\scriptstyle i-1} \\ {\scriptstyle i} \end{array} \begin{pmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 1 \\ \color{red}{0} \end{pmatrix} \longrightarrow v_{[i-1]} \text{ need not be in } Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i-1]})).$$

It is the case if the vector $\begin{pmatrix} 1 \\ 0 \\ \color{red}{0} \\ \vdots \\ 0 \\ \color{red}{0} \\ 1 \end{pmatrix}$ belongs to $Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i]})$

Successors of a vector

v cannot be obtained directly from a vector of $Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i-1]}))$ by appending a "0" at the i^{th} coordinate.

- From a vector $v \in Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i-1]}))$ we generate vectors of $Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i]}))$ by another way than only appending a "0".

Successors of a vector

v cannot be obtained directly from a vector of $Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i-1]}))$ by appending a "0" at the i$^{\text{th}}$ coordinate.

- From a vector $v \in Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i-1]}))$ we generate vectors of $Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i]}))$ by another way than only appending a "0".
- If a generated vector can be produced by several vectors in $Min(Cl_{<\mathrm{maj}>}(\mathcal{S}_{[i-1]}))$, we generate it only if v is the lexicographically smallest vector among them.

Successor

Assume $\begin{array}{c c} {\scriptstyle 1} & \\ {\scriptstyle 2} & \\ {\scriptstyle 3} & \\ \vdots & \\ {\scriptstyle i-2} & \\ {\scriptstyle i-1} & \\ {\scriptstyle i} & \end{array}\!\!\begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ \color{red}{0} \end{pmatrix} \notin Cl_{<\text{maj}>}(\mathcal{S}_{[i]})$.

We append 0 in the i^{th} coordinate and modify the values of the $i-1$ first coordinates that are "incompatible" with the 0 at the i^{th} coordinate.

By the way, it is $< \mathrm{maj} >$

- Let $T \subseteq [i-1]$ be the set of coordinates such that $j \in T$ if
  - $v_j = 0$
  - There is no $x \in \mathcal{S}$ with $x_{j,i} = (0,0)$
- Then let v' obtained from v by turning the coordinates of T to 1 and by appending 0 to the i^{th} coordinate.

By the way, it is $< \mathrm{maj} >$

- Let $T \subseteq [i-1]$ be the set of coordinates such that $j \in T$ if
  - $v_j = 0$
  - There is no $x \in \mathcal{S}$ with $x_{j,i} = (0,0)$
- Then let v' obtained from v by turning the coordinates of T to 1 and by appending 0 to the i^{th} coordinate.

$$v = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \end{pmatrix} \rightarrow v' = \begin{pmatrix} 1 \\ 1 \\ 0 \\ \vdots \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Conclusion

Results:

- For all sets $\mathcal{F}$ of boolean operations, $\textsc{Closure}_{\mathcal{F}} \in$ P and we have an efficient enumeration algorithm of $Cl_{\mathcal{F}}$.
- Add $\uparrow$ and everything is still in $\textsc{DelayP}$.
- Add Min or Max and using a different algorithm everything is in $\textsc{DelayP}$ but the circuits of a binary matroid.

Conclusion

Results:

- For all sets $\mathcal{F}$ of boolean operations, $\textsc{Closure}_{\mathcal{F}} \in$ P and we have an efficient enumeration algorithm of $Cl_{\mathcal{F}}$.
- Add $\uparrow$ and everything is still in $\textsc{DelayP}$.
- Add Min or Max and using a different algorithm everything is in $\textsc{DelayP}$ but the circuits of a binary matroid.

Open questions:

- Deal with the clones characterized by projections in an uniform way.
- Enumerate the circuits of a binary matroids in $\textsc{DelayP}$.
- Improve the algorithm for monotone DNF.

Thanks !

Questions ?