

Smart Automaton Implementations

Thomas Claveirole thomas.claveirole@lrde.epita.fr

LRDE seminar, June 4, 2003



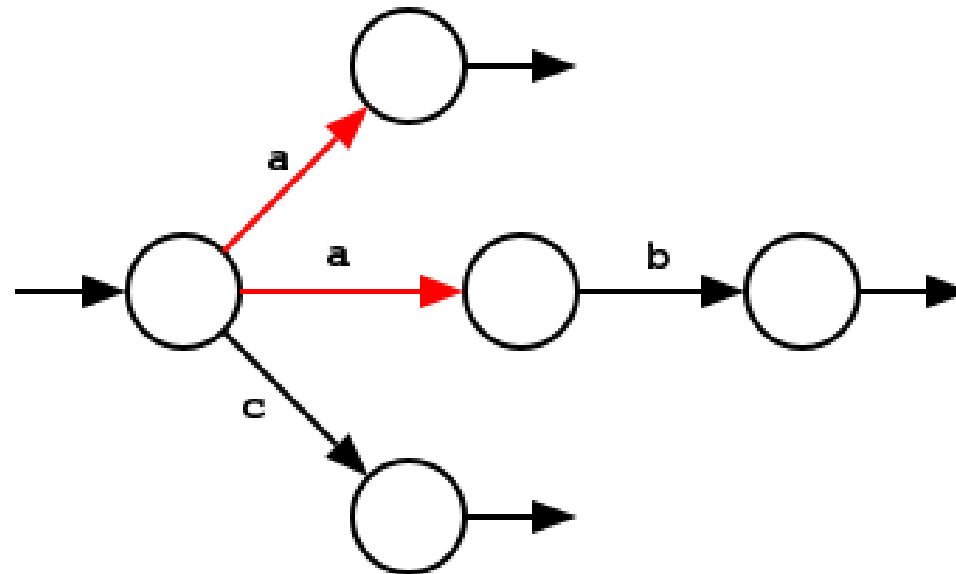
Introduction

The aim is to perform regular expression (regex) search on text.

Classically, there are two approaches :

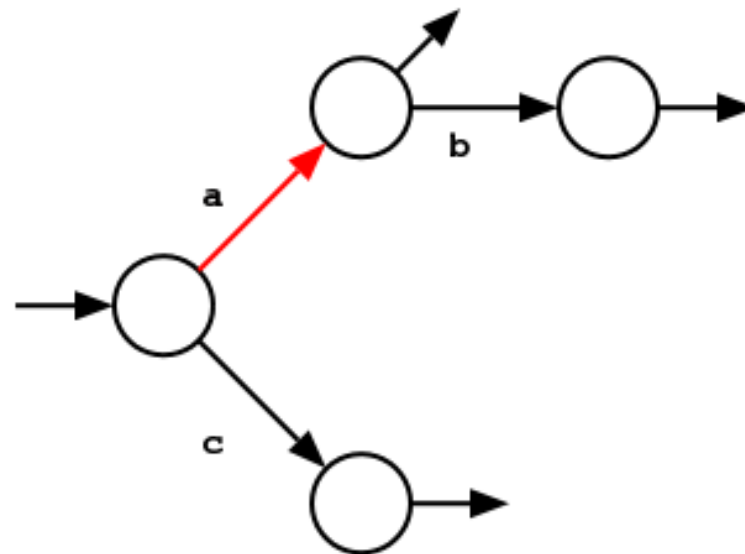
1. **Using Non-deterministic Finite Automata (NFA).**
2. **Using Deterministic Finite Automata (DFA).**

Non-deterministic Finite Automata (NFA)



- Requires a small amount of memory : linear with the regex length.
- **Slow** : depends both on the regex length and the text size.

Deterministic Finite Automata (DFA)



- Fast : linear with the input text size.
- **May require lots of memory** : exponential with the regex size.

Problem

We need to perform fast regex search on large texts, with big regex, but have a limited amount of memory.

As a solution, we could avoid using NFA, but we would like to implement DFA an intelligent way, in order to reduce the memory needed.

Table of Contents

Introduction	1
Reminder and basis	7
Navarro-Raffinot technic	16
Partial determinization (Block automaton)	22
CCP algorithm (Champarnaud, Coulon and Paranthoën)	27
Conclusion	33
References	36

Partial determinization and homogeneous automata

Two technics are combined:

- Glushkov algorithm
- Partial determinization

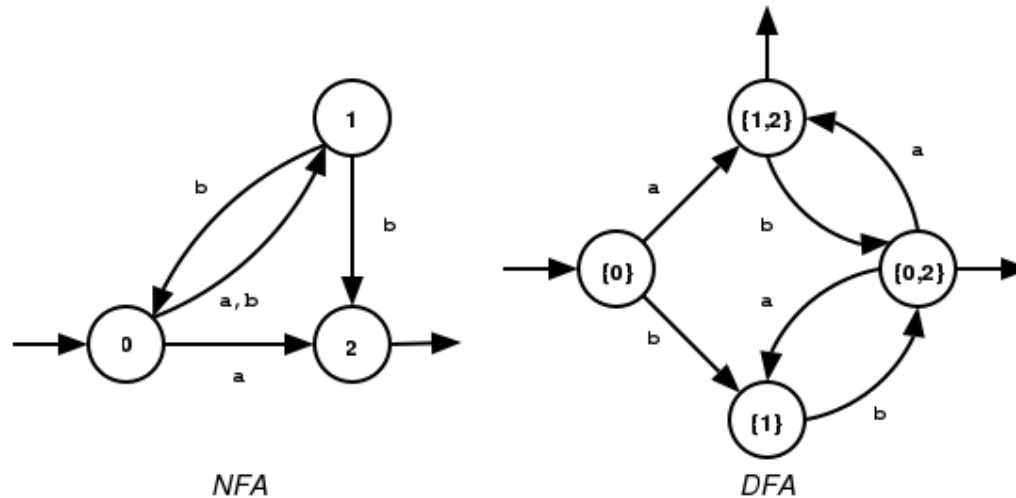
This builds special NFAs.

Using those NFA, one could simulate the corresponding subset automaton.

Fast and compact DFA can also be produced, but with some conditions on the automaton size which limits a lot the interest.

Reminder and basis

From a non-deterministic finite automaton, one could build a deterministic finite automaton. This is the *subset automaton*.



Let $A = \langle Q, \Sigma, \delta, I, F \rangle$ be a NFA,

$\mathcal{A} = \langle 2^Q, \Sigma, \Delta, 0_A, \mathcal{F} \rangle$ is its associated subset automaton.

Bitset

Working on automata implies working on states sets. As states are often treated as integers, efficient integer sets are needed.

A classical method is to use *bitsets*.

In example, for $Q = \{2, 5, 6\}$, one has :

i	0	1	2	3	4	5	6	7
\dot{Q}_i	0	0	1	0	0	1	1	0

The bitmask which represents Q is noted \dot{Q} .

In example, for $Q = \{0, 1, 2, 3\}$, $\dot{Q} = 1111$.

Bitset automaton

With those notations, it is possible to associate to a NFA its *bitset automaton*, which is nothing else than the bitset representation of its subset automaton :

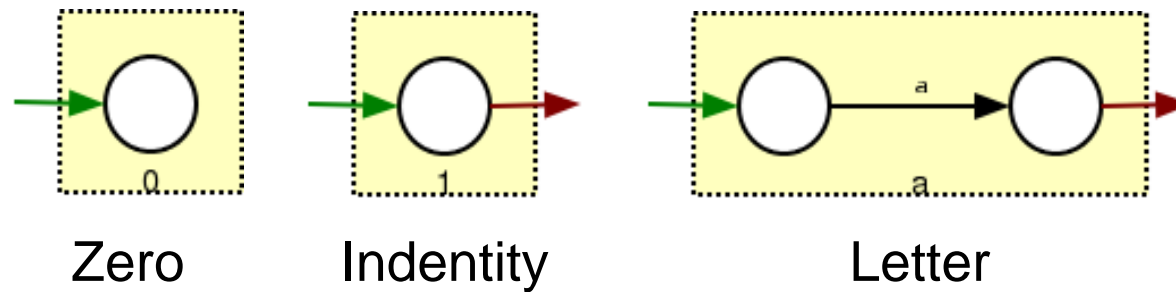
$$\dot{A} = \langle \llbracket 0, \dot{Q} \rrbracket, \Sigma, \dot{\Delta}, \dot{O}_A, \dot{F} \rangle$$

with $\dot{\Delta}$ a function from $\llbracket 0, \dot{Q} \rrbracket \times \Sigma$ onto $\llbracket 0, \dot{Q} \rrbracket$ such as $\dot{\Delta}[\dot{X}, a]$ is the bitmask for $\Delta[X, a]$.

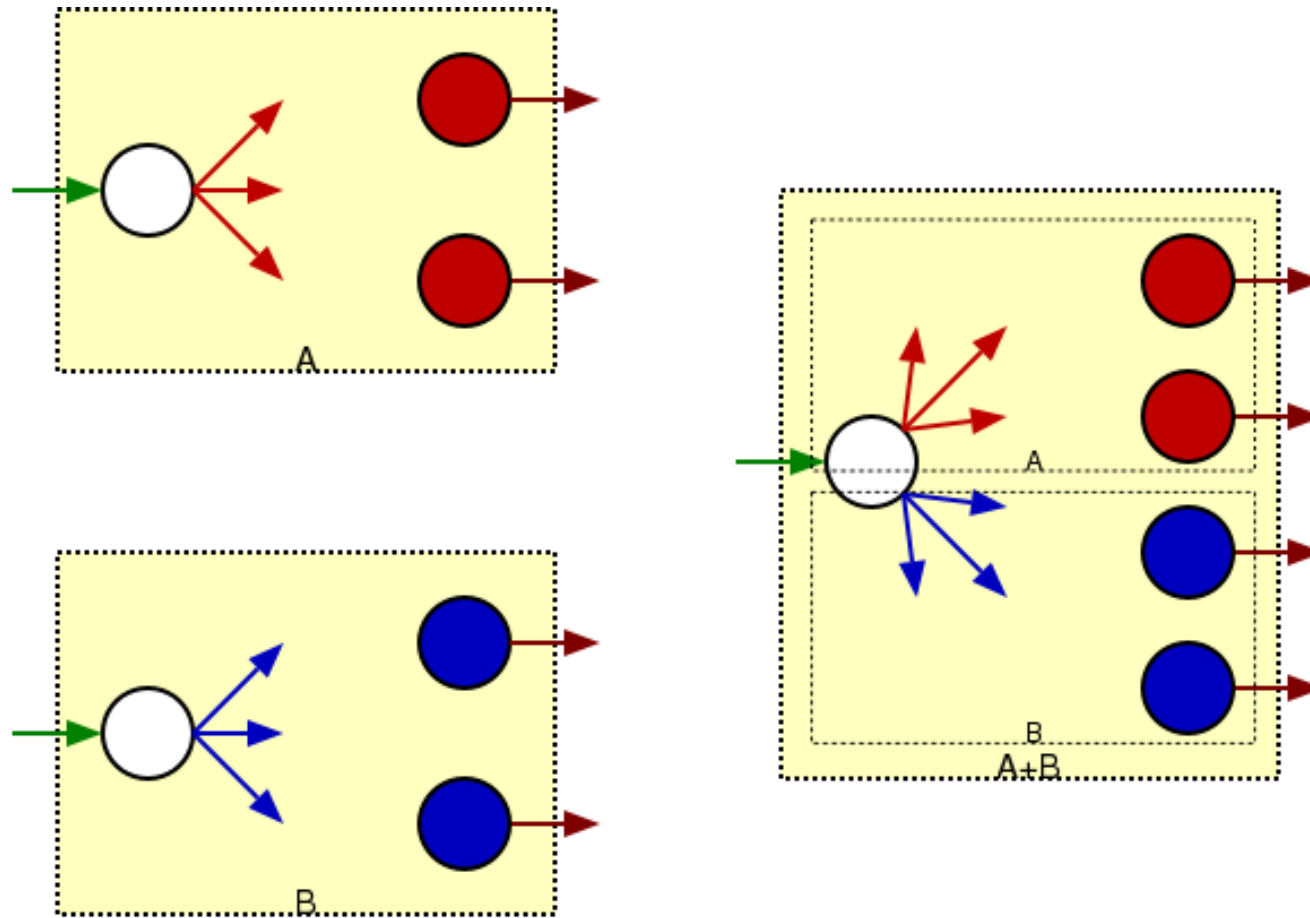
Glushkov's automaton

- It is a “Thompson like” algorithm.
- It builds an automaton recursively from sub-expressions.

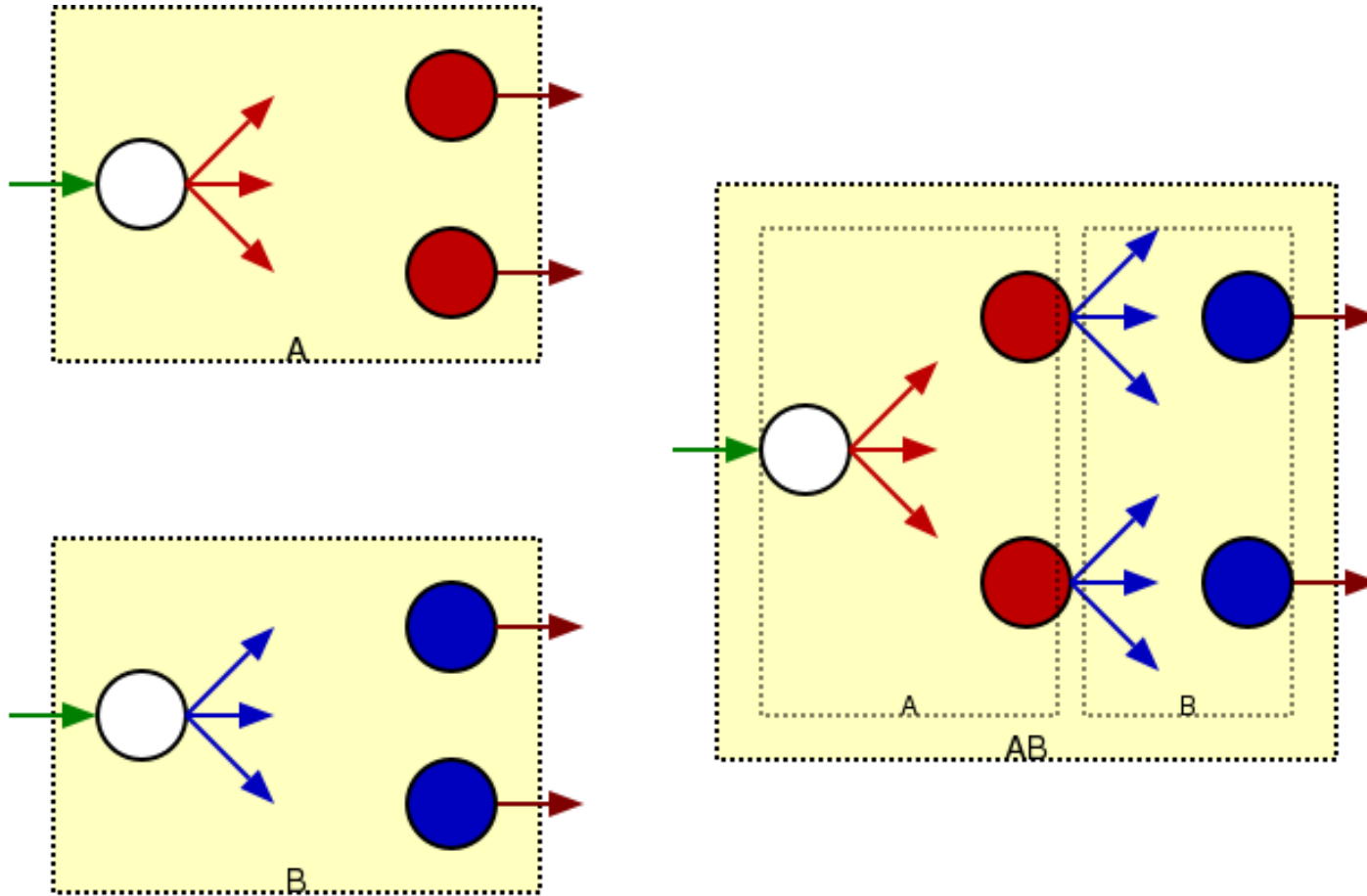
It starts from constants and letter :



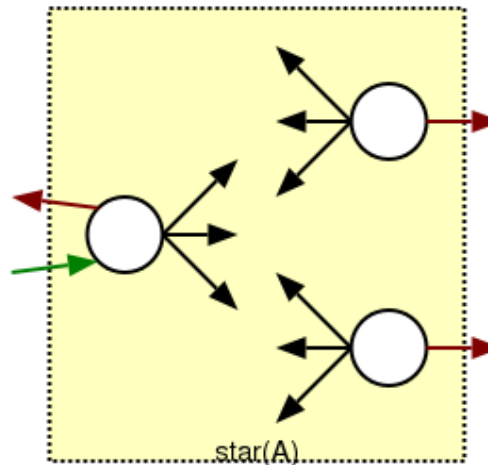
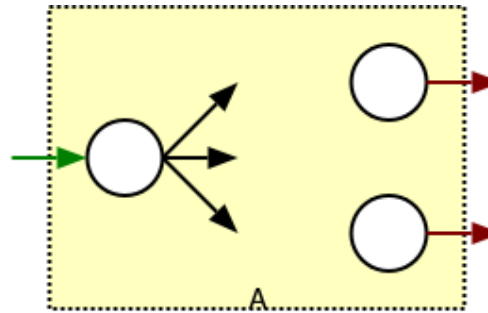
Glushkov – addition



Glushkov – product

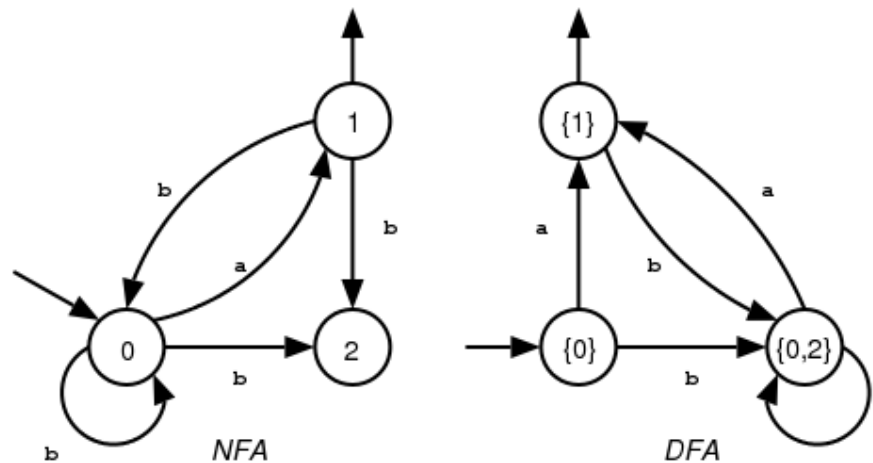


Glushkov – star



Homogeneous automata

Definition : Let $A = \langle Q, \Sigma, \delta, I, F \rangle$ be an automaton. A is homogeneous if and only if, $\forall q \in Q$, all transition entering q are labeled by the same symbol.



Properties : The subset automaton of an homogeneous NFA is homogeneous.

Homogeneous automata and Glushkov

- Given a regular expression with n symbols, the corresponding Glushkov automaton has $n + 1$ states.
- The Glushkov automaton of a regex is homogeneous.
- Therefore, the bitset automaton of a Glushkov automaton is homogeneous.

So, building such a bitset automaton should need $2^{n+1} \times (n + 1) \times |\Sigma|$ bits.

But, using the fact that this automaton is homogeneous, it is possible to have only $2^{n+1} \times (n + 1)$ bits.

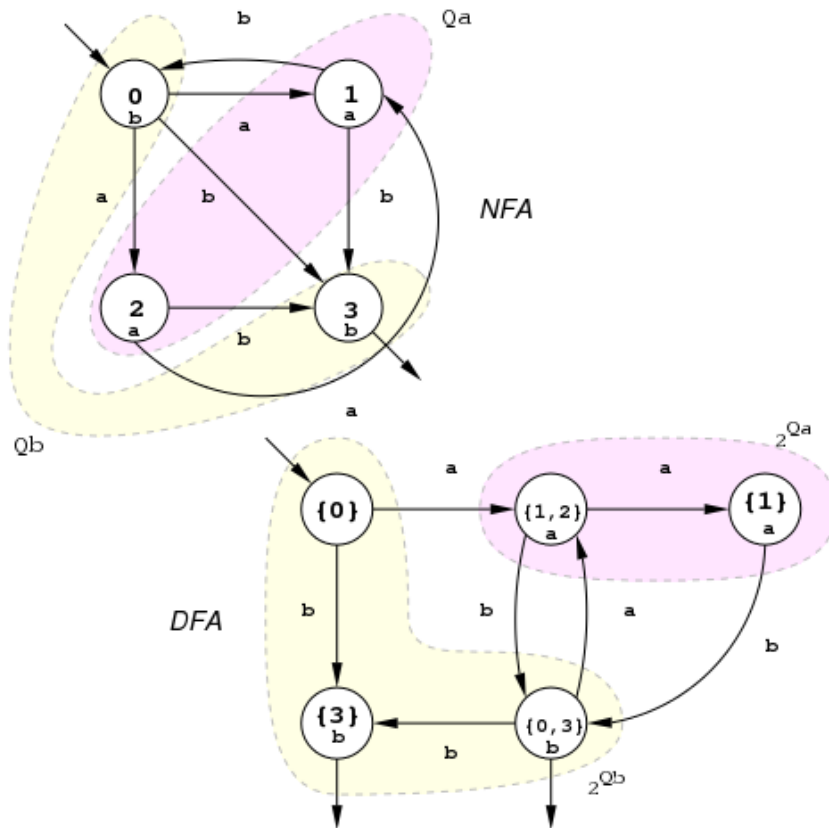
Navarro-Raffinot technic

- Label each states with the corresponding label of its entering transitions.
- For each letter, keep a mask with corresponding states.
- For each states, only memorize outgoing transitions, without their corresponding letter (Needs only one bitset instead of one bitset per letter).

When on a state q , given a letter l :

- follow all outgoing transitions of q which goes on a l labeled state.
- This is a bitwise and between the output mask of q and the mask of state associated to l .

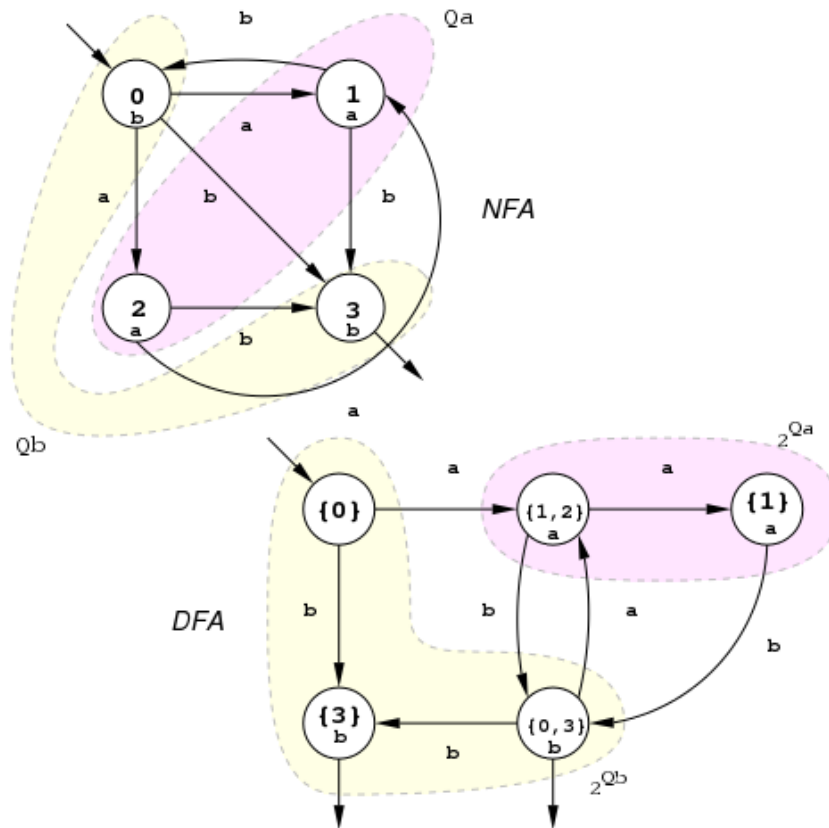
[1/3] DFA construction using the nr algorithm



x	$\dot{Q}x$	Qx
a	0110	{1; 2}
b	1001	{0; 3}

Q	\dot{Q}	$T[\dot{Q}]$	$T[Q]$
\emptyset	0000	0000	\emptyset
{3}	0001	0000	\emptyset
{2}	0010	0101	{1; 3}
{2; 3}	0011	0101	{1; 3}
{1}	0100	1001	{0; 3}
{1; 3}	0101	1001	{0; 3}
{1; 2}	0110	1101	{0; 1; 3}
{1; 2; 3}	0111	1101	{0; 1; 3}
{0}	1000	0111	{1; 2; 3}
{0; 3}	1001	0111	{1; 2; 3}
{0; 2}	1010	0111	{1; 2; 3}
{0; 2; 3}	1011	0111	{1; 2; 3}
{0; 1}	1100	1111	{0; 1; 2; 3}
{0; 1; 3}	1101	1111	{0; 1; 2; 3}
{0; 1; 2}	1110	1111	{0; 1; 2; 3}
{0; 1; 2; 3}	1111	1111	{0; 1; 2; 3}

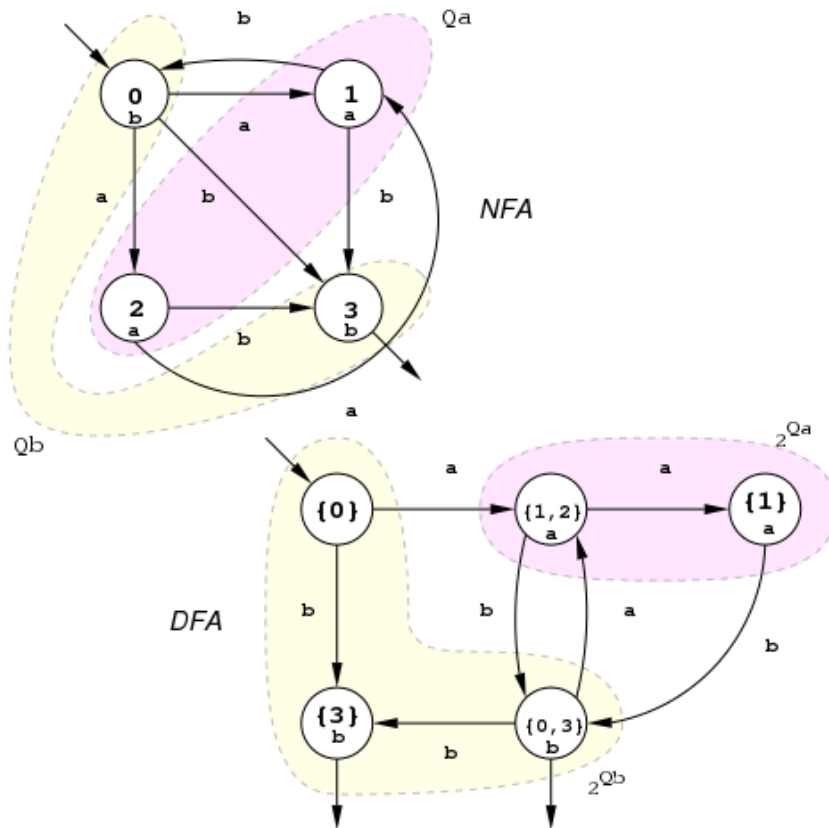
[2/3] DFA construction using the nr algorithm



x	$\dot{Q}x$	Qx
a	0110	{1; 2}
b	1001	{0; 3}

Q	\dot{Q}	$T[\dot{Q}]$	$T[Q]$
\emptyset	0000	0000	\emptyset
{3}	0001	0000	\emptyset
{2}	0010	0101	{1; 3}
{2; 3}	0011	0101	{1; 3}
{1}	0100	1001	{0; 3}
{1; 3}	0101	1001	{0; 3}
{1; 2}	0110	1101	{0; 1; 3}
{1; 2; 3}	0111	1101	{0; 1; 3}
{0}	1000	0111	{1; 2; 3}
{0; 3}	1001	0111	{1; 2; 3}
{0; 2}	1010	0111	{1; 2; 3}
{0; 2; 3}	1011	0111	{1; 2; 3}
{0; 1}	1100	1111	{0; 1; 2; 3}
{0; 1; 3}	1101	1111	{0; 1; 2; 3}
{0; 1; 2}	1110	1111	{0; 1; 2; 3}
{0; 1; 2; 3}	1111	1111	{0; 1; 2; 3}

[3/3] DFA construction using the nr algorithm



x	$\dot{Q}x$	Qx
a	0110	{1; 2}
b	1001	{0; 3}

Q	\dot{Q}	$T[\dot{Q}]$	$T[Q]$
\emptyset	0000	0000	\emptyset
{3}	0001	0000	\emptyset
{2}	0010	0101	{1; 3}
{2; 3}	0011	0101	{1; 3}
{1}	0100	1001	{0; 3}
{1; 3}	0101	1001	{0; 3}
{1; 2}	0110	1101	{0; 1; 3}
{1; 2; 3}	0111	1101	{0; 1; 3}
{0}	1000	0111	{1; 2; 3}
{0; 3}	1001	0111	{1; 2; 3}
{0; 2}	1010	0111	{1; 2; 3}
{0; 2; 3}	1011	0111	{1; 2; 3}
{0; 1}	1100	1111	{0; 1; 2; 3}
{0; 1; 3}	1101	1111	{0; 1; 2; 3}
{0; 1; 2}	1110	1111	{0; 1; 2; 3}
{0; 1; 2; 3}	1111	1111	{0; 1; 2; 3}

Bitset projection

Let Q be a set, $R \subset Q$ and $q \in Q$. Here we defines :

$$\pi(q, R) = \begin{cases} \emptyset & \text{when } q \notin R \\ \{q\text{'s rank in } R, \text{ starting from } 0\} & \text{else} \end{cases}$$

and

$$\pi(S, R) = \bigcup_{q \in S} \pi(q, R)$$

typically, $Q = \llbracket 0, n - 1 \rrbracket$, *i.e.* a subset of states.

Bitset projection by practice !

Q	1111	$\llbracket 0, 3 \rrbracket$
R	0011	$\{2; 3\}$
S	0101	$\{1; 3\}$
$\pi(S, R)$	01	$\{1\}$

The result of $\pi(S, R)$ needs $|R|$ bits.

A good partitioning of Q implies:

- Less bits to work on.
- Always possible to have bitsets which can be hold in a machine word.

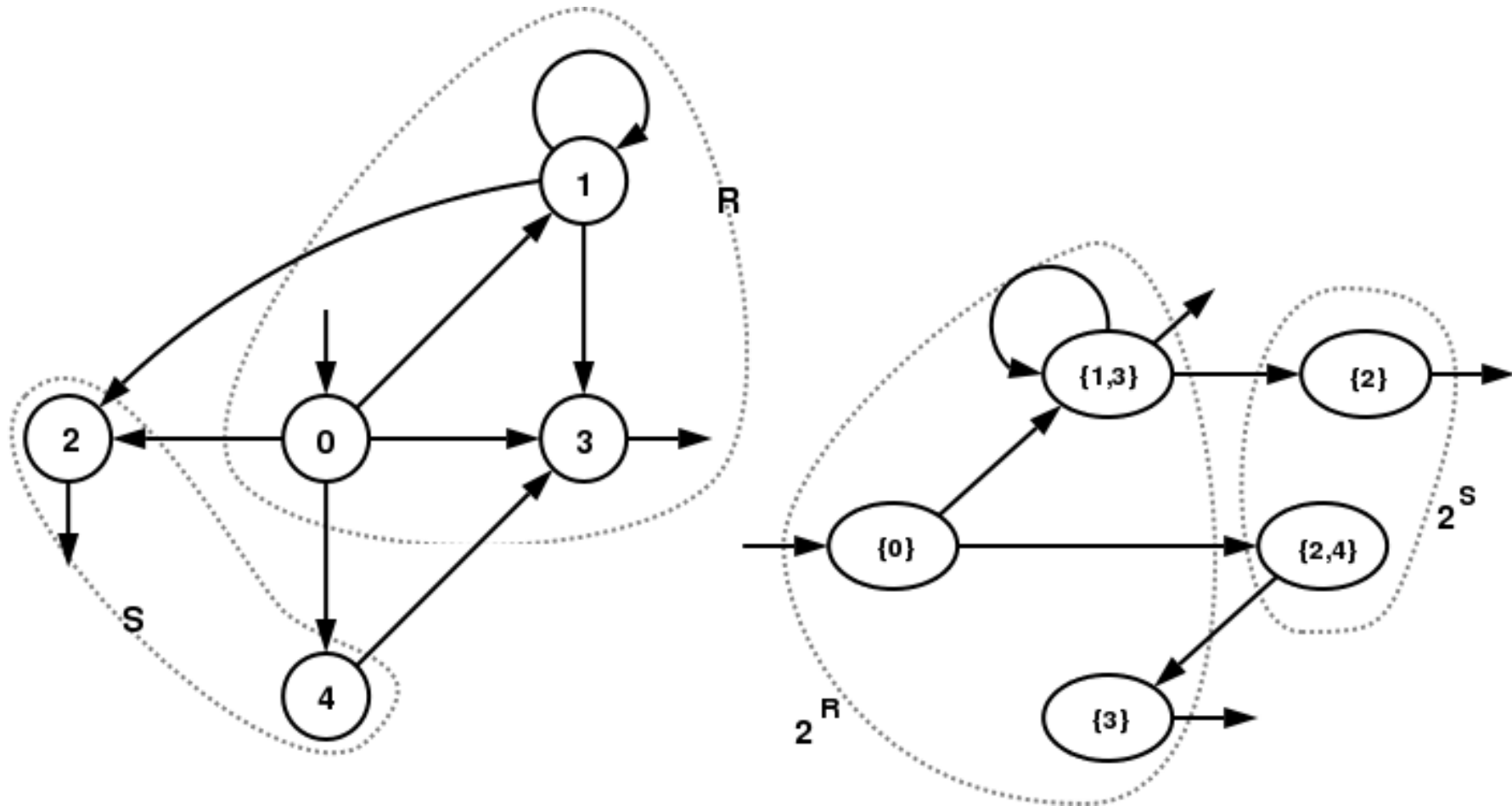
Partial determinization (Block automaton)

Let $A = \langle Q, \Sigma, \delta, 0_A, F \rangle$ be a NFA and $B = (Q_i)_{i \in \llbracket 0, k-1 \rrbracket}$ a partitioning of Q . Let $\Delta_{i,j}(P, a) = \pi(\Delta(\pi^{-1}(P, Q_i), a), Q_j)$.

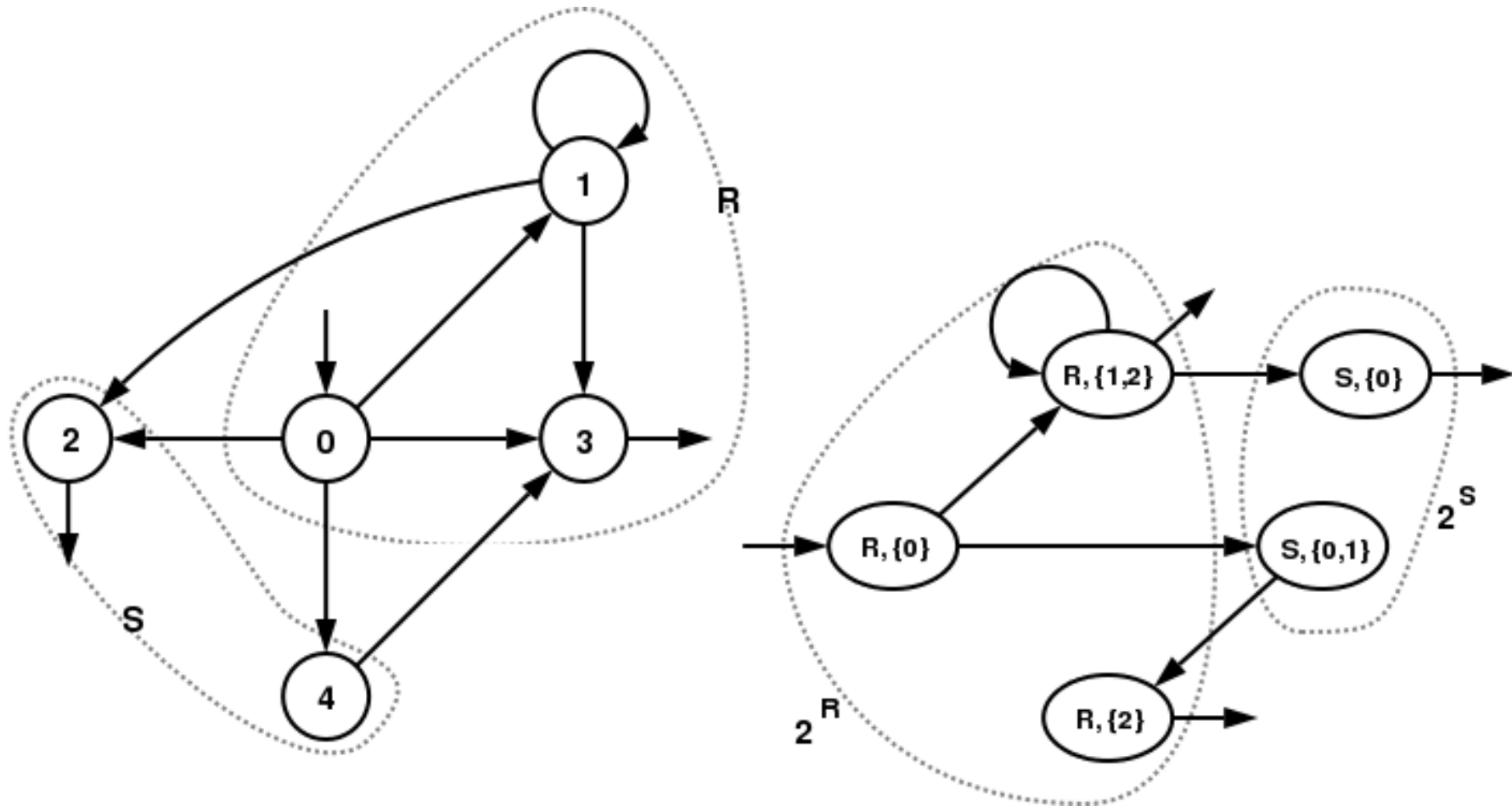
The *block automaton* of A w.r.t. B is $\mathcal{A}_B = \langle \mathcal{Q}_B, \Sigma, \Delta_B, 0_B, \mathcal{F}_B \rangle$, with :

$$\begin{aligned}\mathcal{Q}_B &= \{(i, P) \mid i \in \llbracket 0, k-1 \rrbracket, P \subset \llbracket 0, |Q_i| - 1 \rrbracket\} \\ \Delta_B((i, P), a) &= \{(j, \Delta_{i,j}(P, a)) \mid j \in \llbracket 0, k-1 \rrbracket\} \\ 0_B &= \{(i, \pi(0_A, Q_i)) \mid i \in \llbracket 0, k-1 \rrbracket\} \\ \mathcal{F}_B &= \{(i, P) \in Q_i \mid i \in \llbracket 1, k-1 \rrbracket, \pi^{-1}(P, Q_i) \cap F \neq \emptyset\}\end{aligned}$$

[1/2] Partial determinization example



[2/2] Partial determinization example



Partial determinization complexity

l is the input text length.

k is the number of partitions.

n is the NFA's number of states.

- Processing a text with a block-bit automaton takes an $l \times k^2$ time.
- A block-bit automaton needs $|\Sigma| \times n \sum_{i=0}^{k-1} 2^{|Q_i|}$ bits.

Navarro-Raffinot technic with partial determinization

With homogeneous automata, there is no need to memorize the letters which are associated to a transition.

So, combining the Navarro-Raffinot algorithm and partial determinization only needs $n \sum_{i=0}^{k-1} 2^{|Q_i|}$ bits.

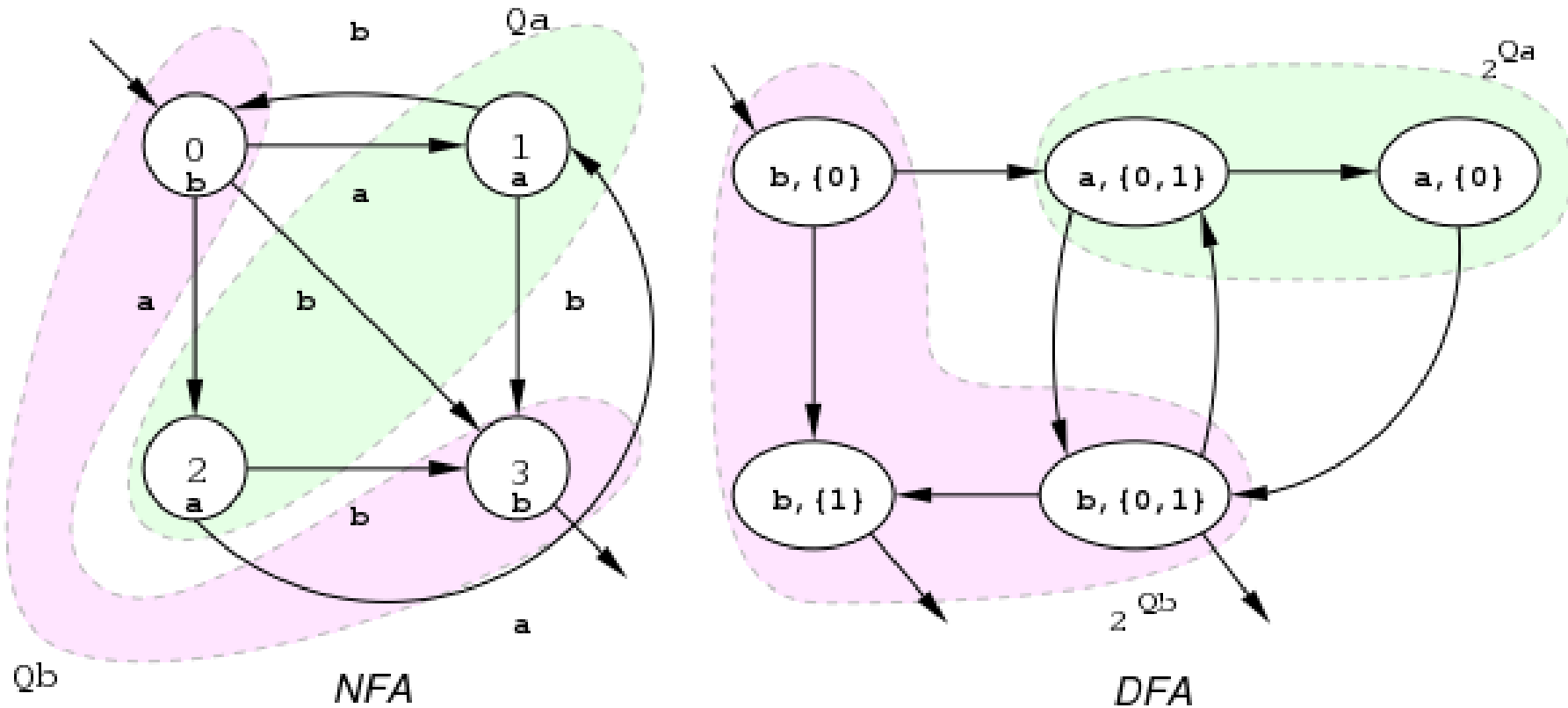
CCP algorithm (Champarnaud, Coulon and Paranthoën)

Let $A = \langle Q, \Sigma, \delta, 0_A, F \rangle$ be an homogeneous automaton and $B = (Q_a)_{a \in \Sigma}$ a partitioning of Q such as :

$$Q_a = \{q \mid q \text{ is labelled by } a\}$$

Properties : The block-automaton of A *w.r.t.* B is deterministic and needs $n \left(\sum_{a \in \Sigma} 2^{|Q_a|} \right)$ bits to be stored.

CCP Algorithm example



CCP algorithm with partial determinization

Problem : Sometimes, $|Q_a|$ needs too much bits to be stored in a machine word.

So, we need smaller partitions of Q . The idea is to have

$$B = (Q_{a,i})_{\substack{a \in \Sigma \\ i \in [0, k_a - 1]}}$$

with k_a the number of “sub-partitions” of Q_a .

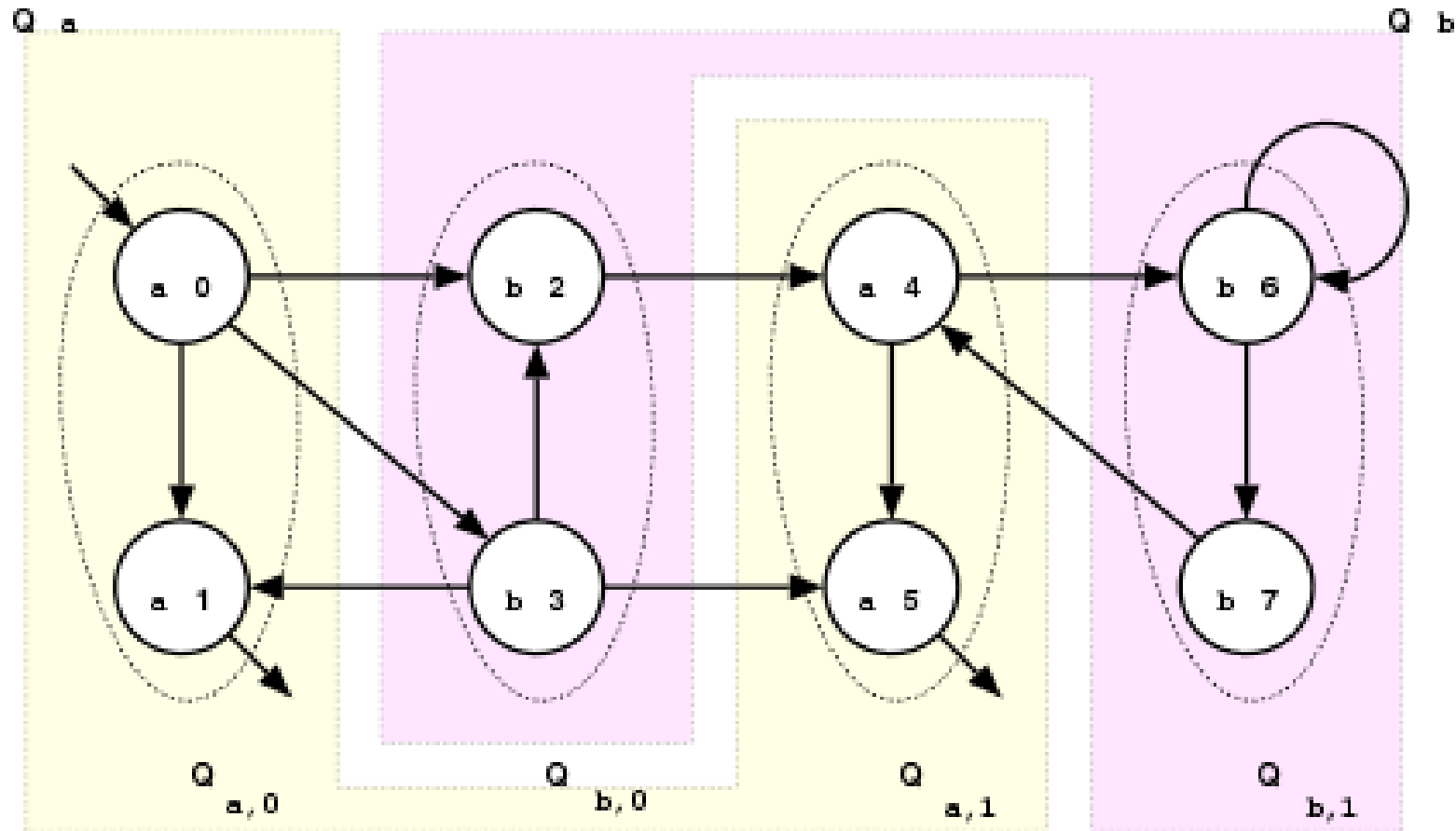
CCP algorithm specifications (oumph)

Let $\Delta_{(a,i)(b,j)}(X) = \pi(\Delta(\pi^{-1}(X, Q_{a,i}), b), Q_{b,j})$.

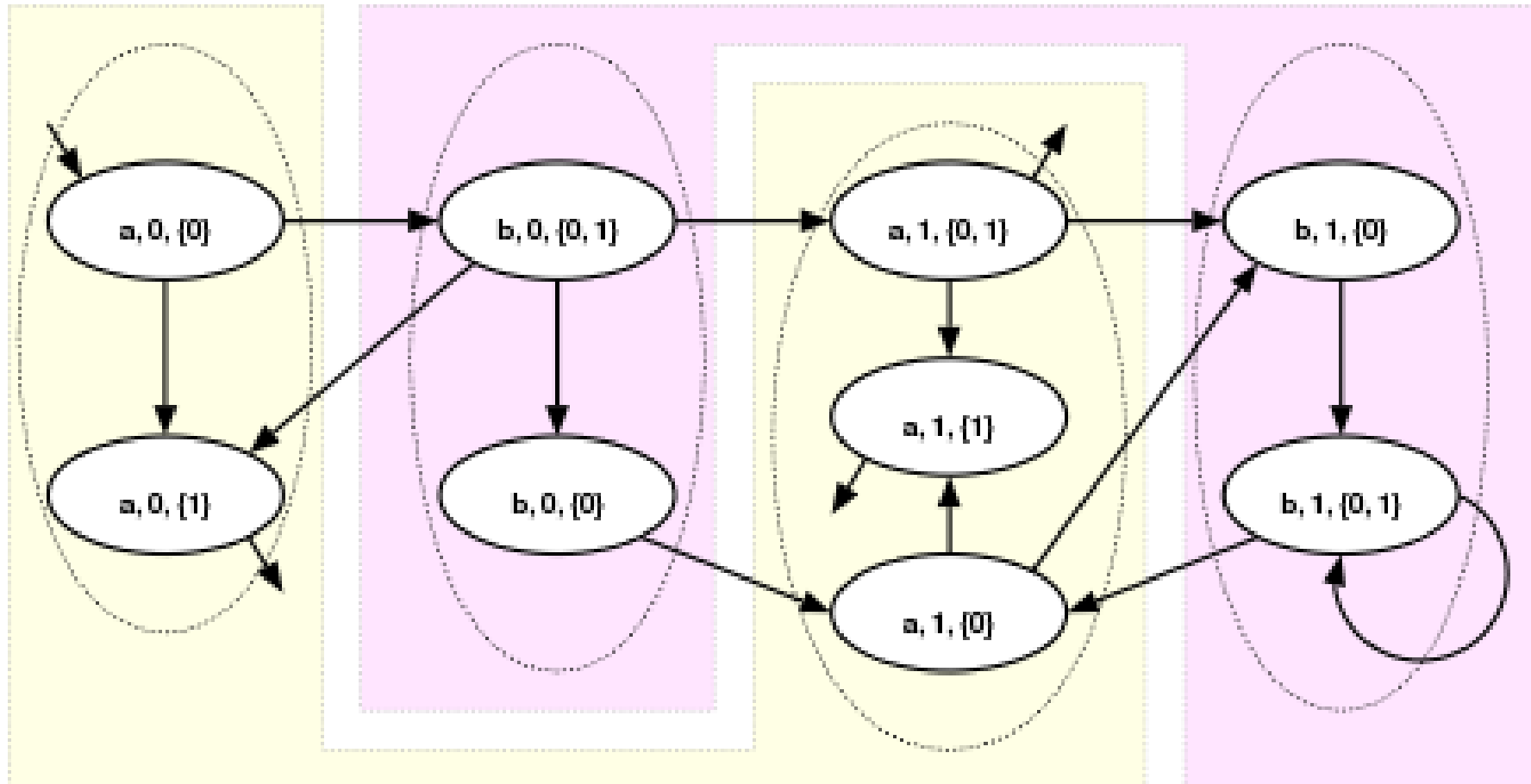
The homogeneous subset automaton *w.r.t.* \mathbf{B} is $\mathcal{H}_S = \langle \mathcal{Q}_S, \Sigma, \Phi_S, 0_S, \mathcal{F}_S \rangle$ where :

$$\begin{aligned} \mathcal{Q}_S &= \{(a, (i, P)) \mid a \in \Sigma, i \in \llbracket 0, k_a - 1 \rrbracket, P \subset \llbracket 0, |Q_{a,i}| - 1 \rrbracket\} \\ \Phi_S((a, (i, P)), b) &= \{(b, (j, \Delta_{(a,i)(b,j)}(P))) \mid j \in \llbracket 0, k_b - 1 \rrbracket\} \\ 0_S &= \{(a, (i, \pi(I, Q_{a,i}))) \mid a \in \Sigma, i \in \llbracket 0, k_a - 1 \rrbracket\} \\ \mathcal{F}_S &= \{(a, (i, P)) \in \mathcal{Q}_S \mid \pi^{-1}(P, Q_{a,i}) \cap F \neq \emptyset\} \end{aligned}$$

[1/2] CCP algorithm example



[2/2] CCP algorithm example



Conclusion

- When do I need to use it ?
- Perspectives.

When do I need to use it ?

- I have heavy non-deterministic automata.
- I have big regex.
- I have a limited amount of memory.
- I can't afford a NFA because of processing time.
- **NR**: I have many character classes (*i.e.* [a - z]).
- **CCP**: Symbols are well distributed in my regex.

Perspectives

- Implement it in Vaucanson.
- Re-think with multiplicities.

References

Aho, A., Sethi, R., and Ullman, J. (1986). *Compilers, Principles, Techniques, and Tools*, chapter Lexical Analysis. Addison-Wesley.

Champarnaud, J.-M., Coulon, F., and Paranthoën, T. (2003). Compact and fast algorithms for regular expression search. Technical report, LIFAR, University of Rouen, France, <http://www.univ-rouen.fr/LIFAR/aia/ccp.ps>.

Claveirole, T. (2003). Smart automaton implementations. Technical report, LRDE.