Mical

a regular grammar inference library

Maxime Rey
<max@lrde.epita.fr>

LRDE seminar, June 4, 2003

Introduction

- **Goal of grammar inference:** learn a language.
  We need:
  - a set of words contained in this language
  - eventually a set of words not contained in this language

- regular languages $\equiv$ finite state machines
  $\Rightarrow$regular language inference $\equiv$ inference on automata
  $\Rightarrow$ automata manipulation
  $\Rightarrow$ use Vaucanson, a generic library for manipulation of automata

Table of Contents

Mical: an external point of view

- Positive Sample: set of words which are contained in the language to infer
- Negative Sample: set of words which are **not** contained in the language to infer
- Two kinds of inference algorithms:
  - without negative sample
  - with negative sample

Inference algorithms without negative sample
General Way

- Gold’s Theorem: inference on all regular languages is impossible with a positive sample only.
- We can infer on particular subclasses of regular languages:
  - k-testable languages
  - k-reversible languages

Algorithms without negative sample supported in Mical

- k-TSSI (k-Testable in the Strict Sense of Inference)
- k-RI (k-Reversible Inference)
- MGGI (Morphic Generator Grammatical Inference)
- ECGI (Error Correcting Grammatical Inference)
- Services: is deterministic with anticipation k ? is k-reversible ? etc...

Expression power needed

Primitives for this kind of algorithm $\equiv$ automata's manipulation

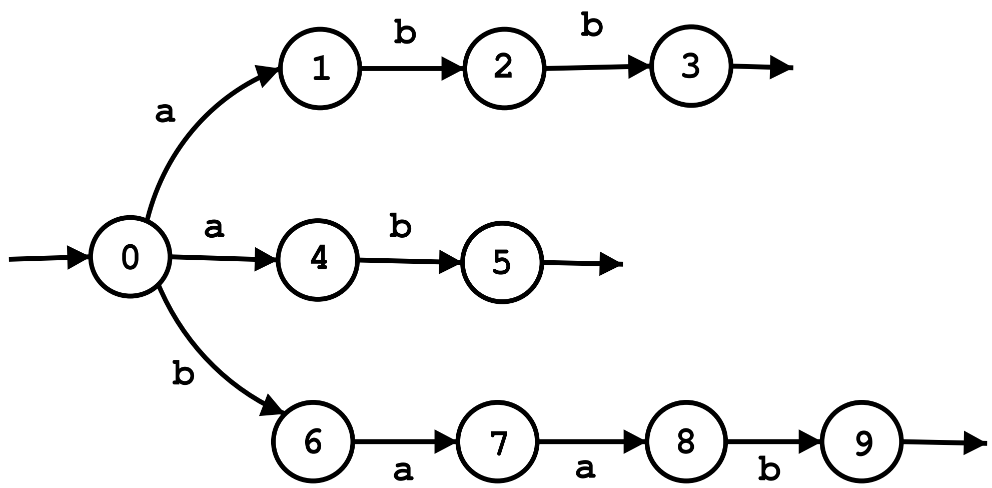
$\Rightarrow$ For inference algorithms without negative sample, Vaucanson is enough.

Inference algorithms with negative sample

- Better theoretical framework: we can really infer over regular languages.
- **Principle:** start from an hyper-specialized automaton and generalize it under verification of the negative sample.
- We need:
  - an hyper-specialized automaton to start inference
  - a generalization operation

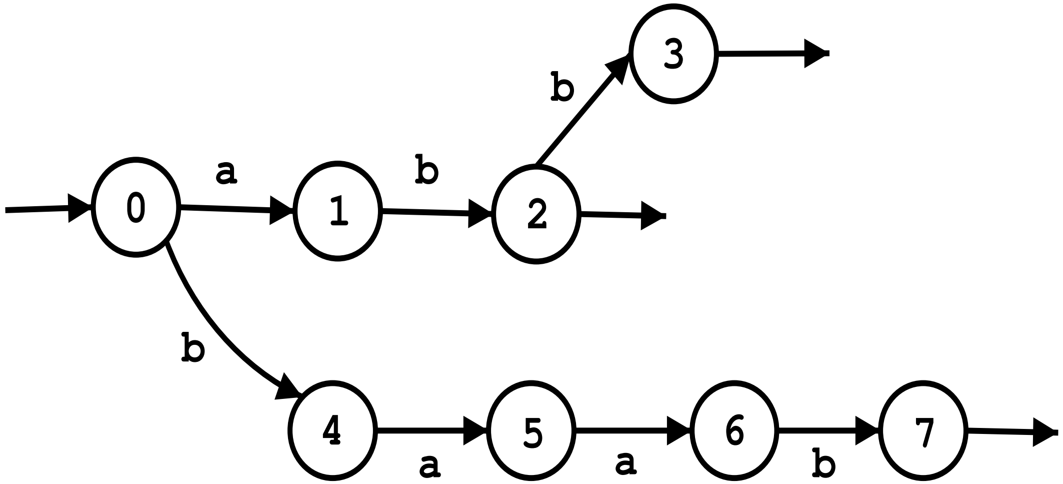
MCA (Maximal Canonical Automaton)

- $I_+ = \{abb, ab, baab\}$

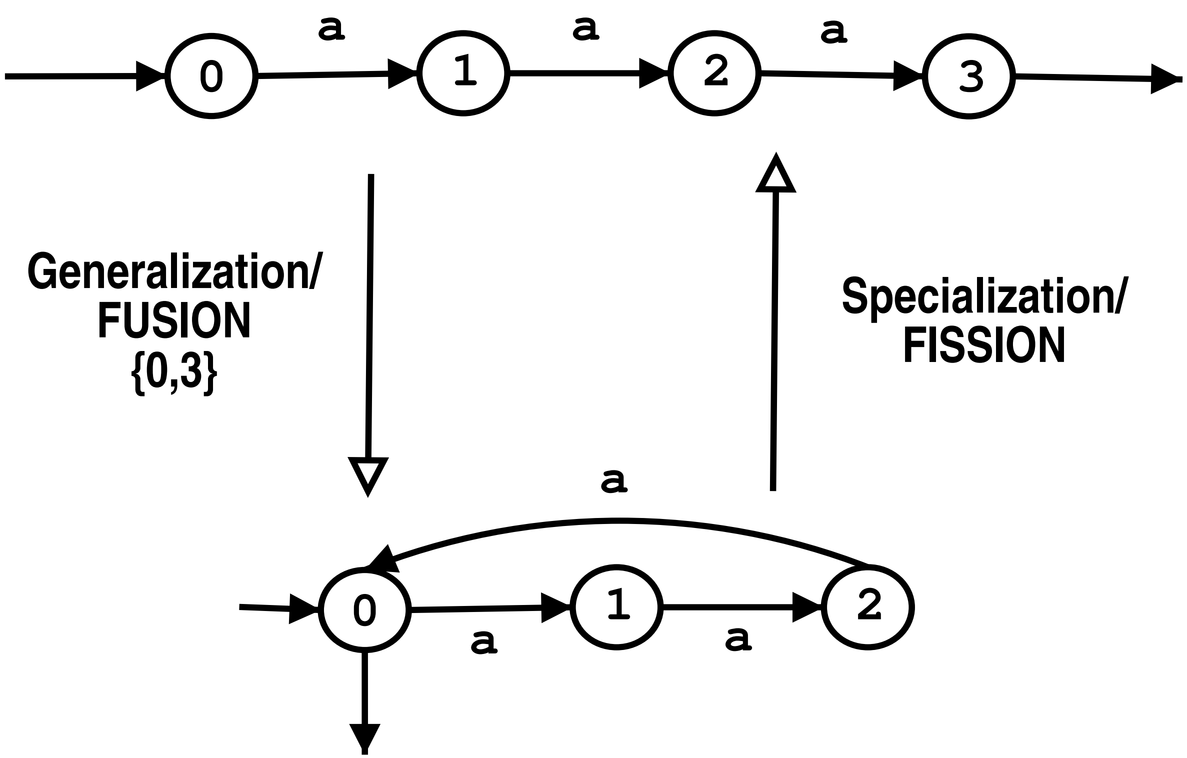


PTA (Prefix Tree Acceptor)

- $I_+ = \{abb, ab, baab\}$



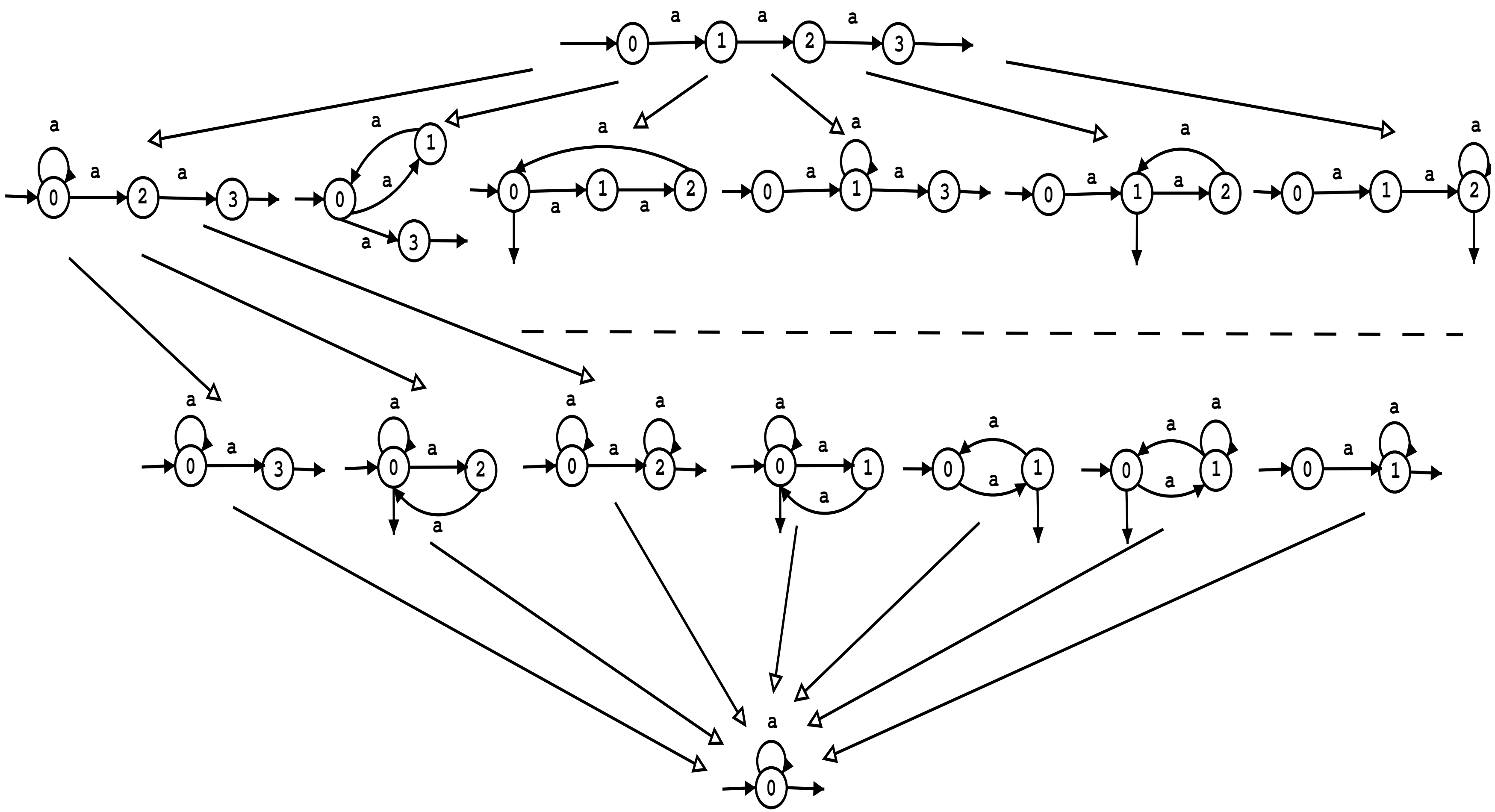
Fusion/Fission



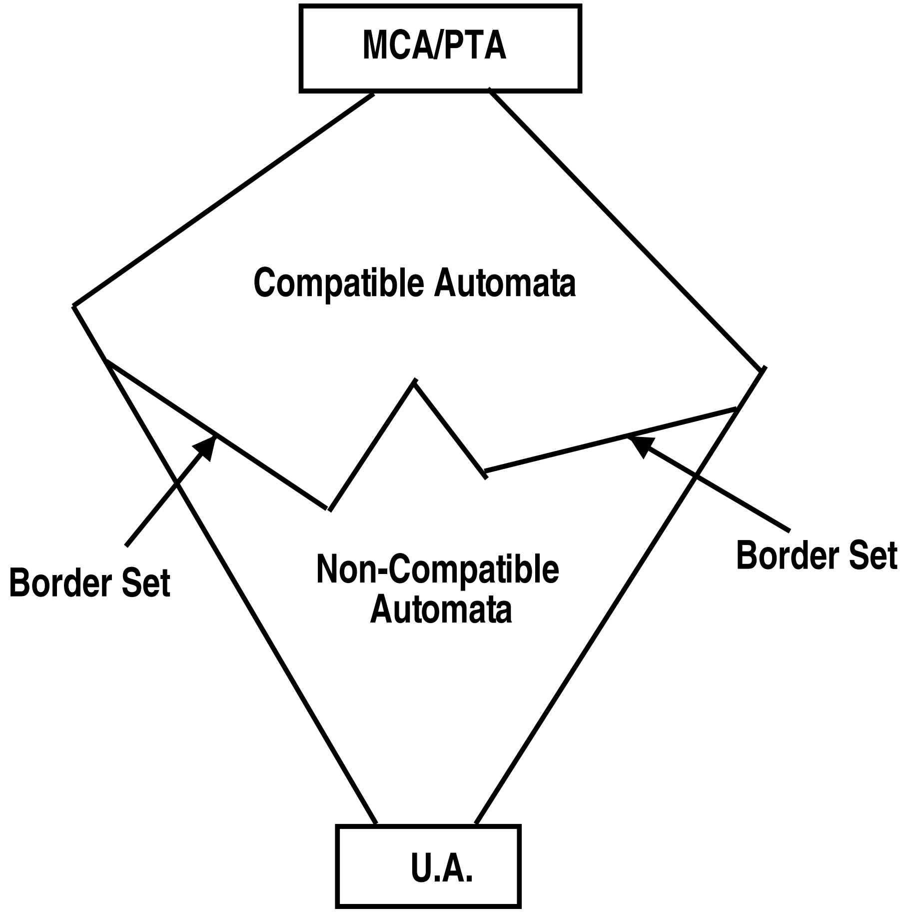
Search space

- Hyper-specialized automaton (MCA/PTA) + generalization operation (fusion) $\Rightarrow$ we can infer.
- **Search space:** set of all automata produced by recursive application of fusion.
- Our search space is a lattice.
- It's not an exhaustive search space.

An example of lattice / search space



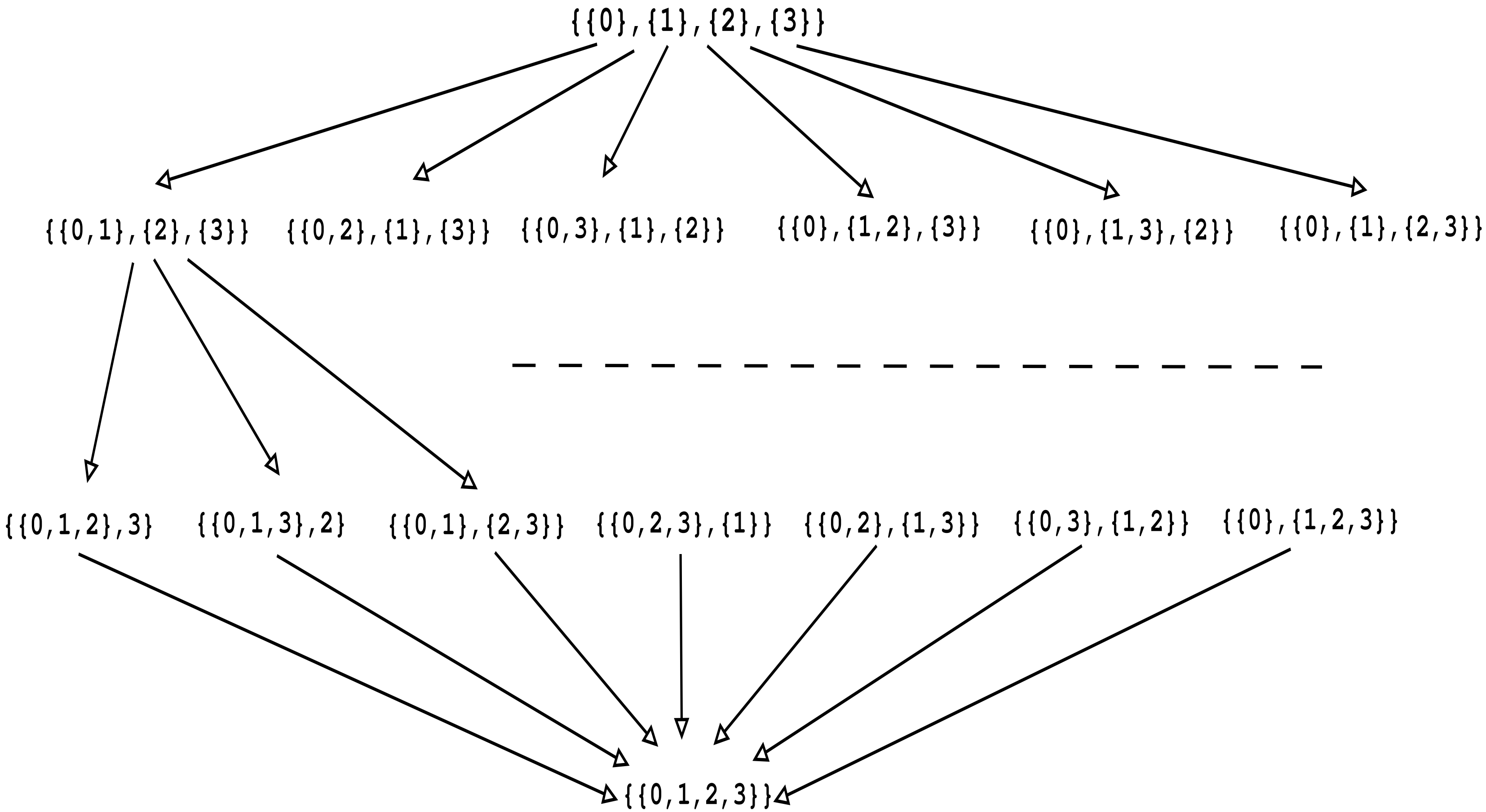
Lattice and Border Set



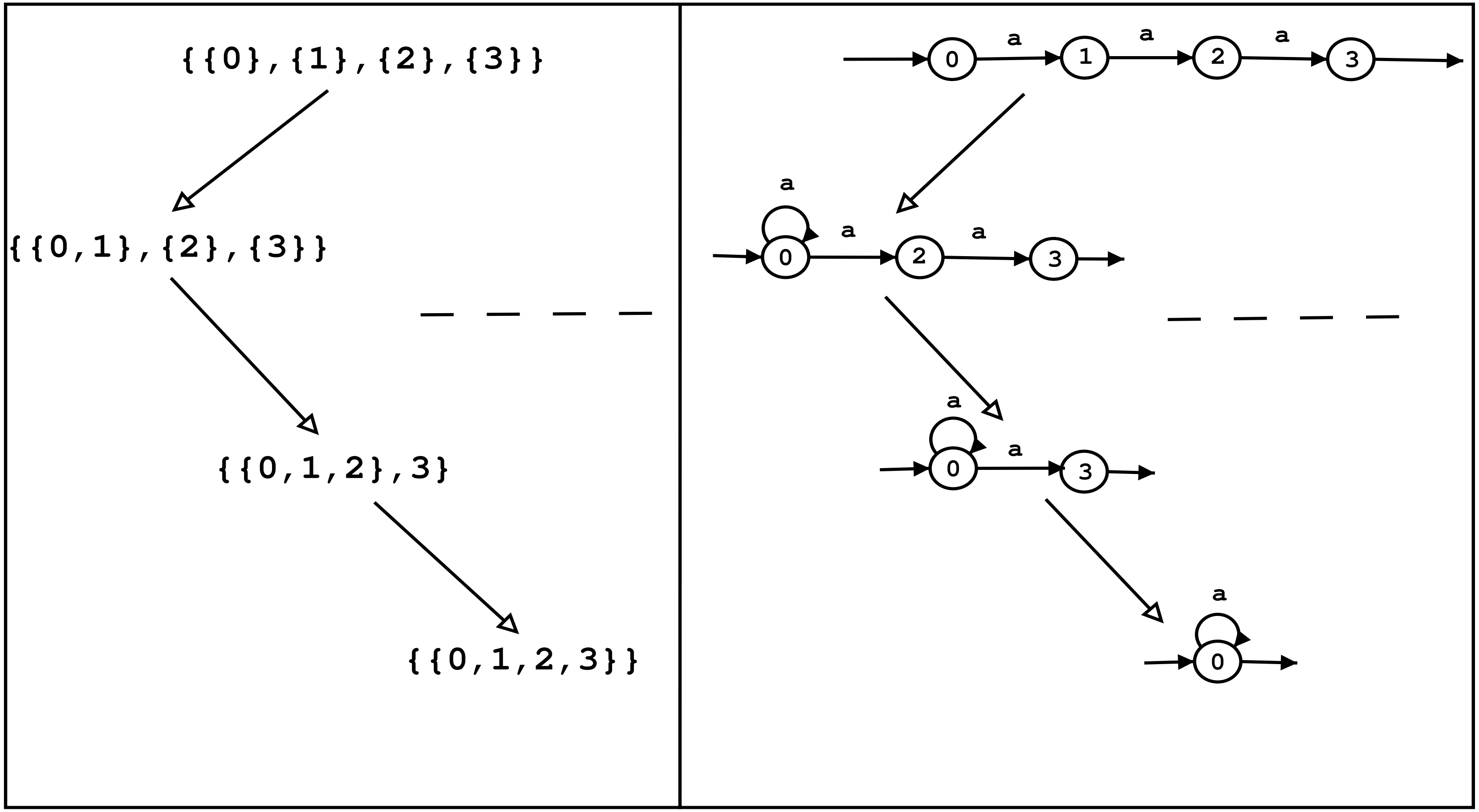
Algorithms with negative sample supported in Mical

- RIG (Regular Inference of Grammars) [inefficient]
- BRIG (Boosted beam-search Regular Inference of Grammars) [inefficient]
- RPNI: [efficient - polynomial]
  - traditional implementation
  - with final suppression (other version)
- RPNI2: an incremental algorithm

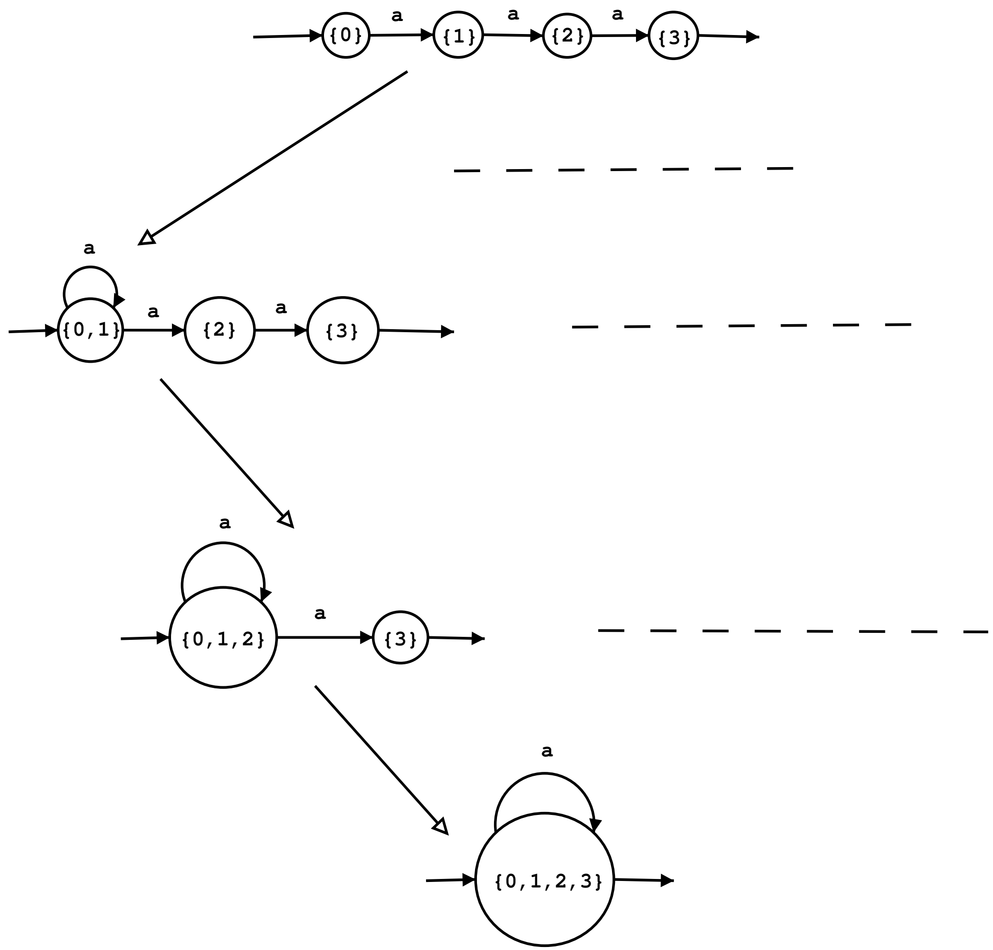
Lattice of partitions (1/2)



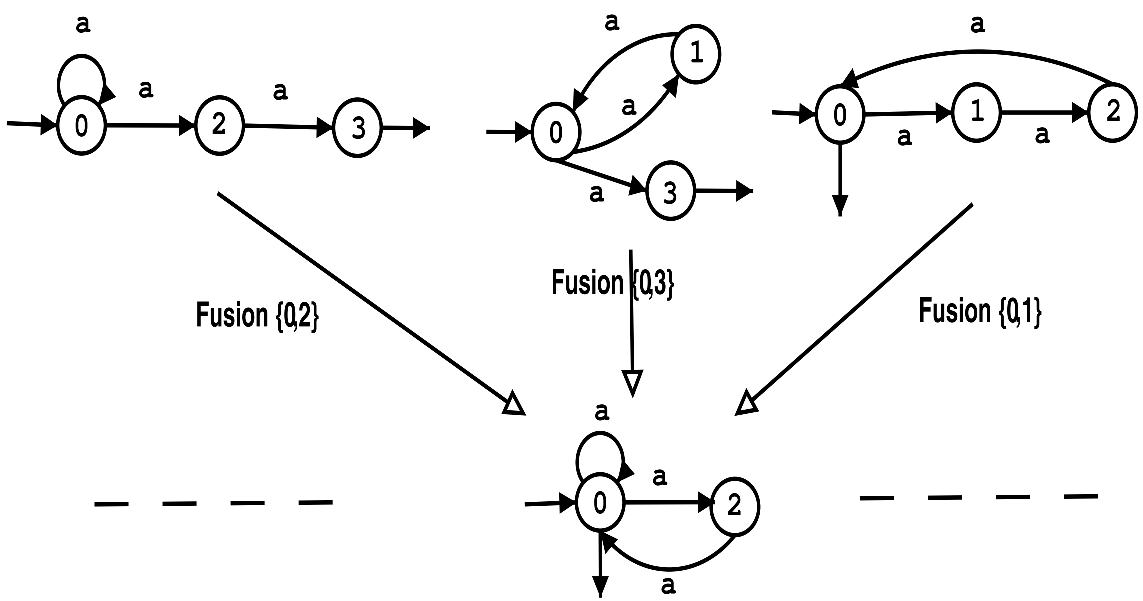
Lattice of partitions (2/2)



Wanted search space



The Oracle



- A MCA (or PTA) with 30 states $\Rightarrow 8.5 \times 10^{23}$ elements in our lattice/space search $\Rightarrow$ we can't label the lattice elements.

Mical’s services

- Needed properties on samples:
  - Structurally Complete
  - Characteristic Samples
  - ...
- However user do not have to use them explicitly.

Mical: inside it

Specific patterns for regular grammar inference.

The Walkers (1/3)

- Goal: Permit to automate a course in an automaton.
- Intuitive point of view: concept near to Visitor pattern.
- They are Functors: higher order functions.

The Walkers (2/3)

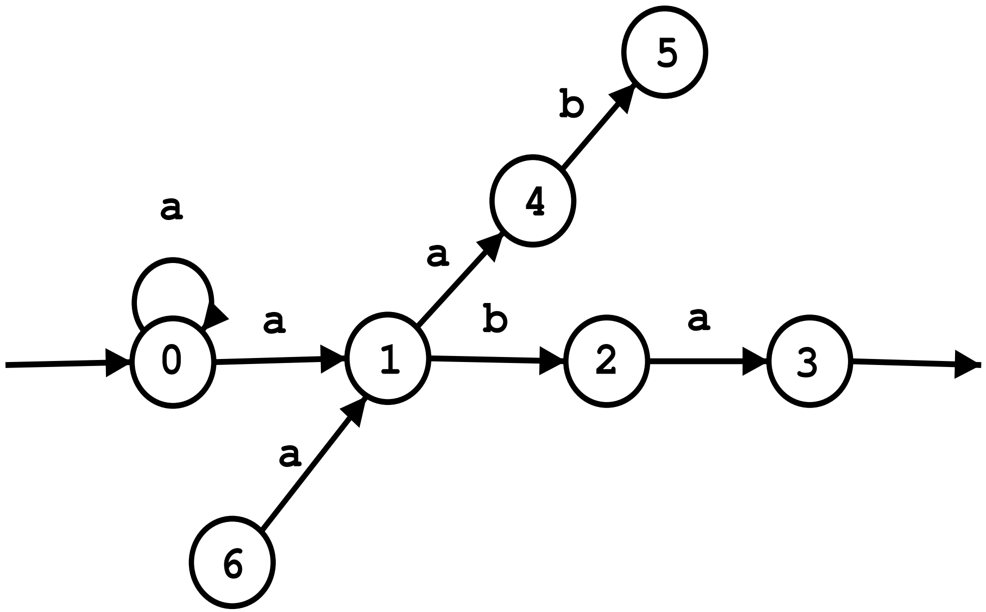
- Three kinds of walkers/courses:
  - Depth Walker
  - Large Walker
  - Order Walker

- The Order Walker is a cheater, it follows only a linear order.

The Walkers (3/3)

- For each kind of walkers/courses, you can choose:
  - a responsible course
  - a safe course

An example (1/2)



- accessible states = {0, 1, 2, 3, 4, 5}
- co-accessible states = {0, 1, 2, 3, 6}

An example (2/2)

Methods Definition of the GetAccessible functor

```
// prototype of () operator
{
course_depth(automaton, *this, security::safe_t(1));
return intersection(accessible, co_accessible);
}
// prototype of prefix method
{
accessible.insert(info.actual_state);
if (info.automaton.is_final(info.actual_state))
co_accessible = get_co_accessible(info.automaton, info.actual_state);
}
// prototype of delta method
{
info.automaton.deltac(...); }
```

Methods Definition of the GetCoAccessible functor

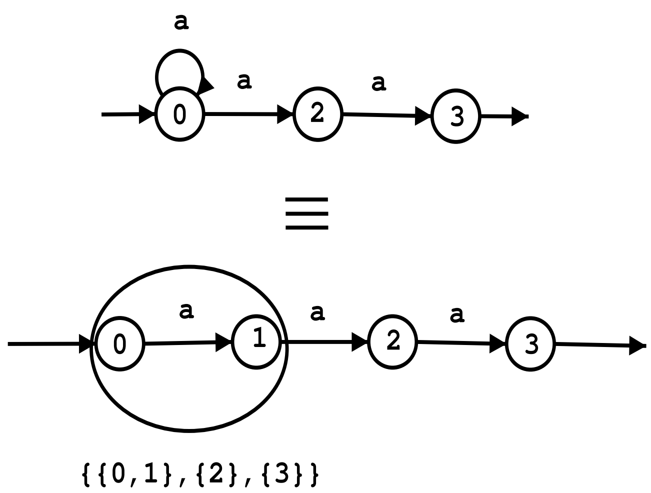
```
// prototype of () operator
{
course_depth(automaton, *this, security::safe_t(1));
return co_accessible;
}
// prototype of prefix method
{
co_accessible.insert(info.actual_state);
}
// prototype of delta method
{
info.automaton.rdeltac(...);
}
```

The Framework (1/3)

- We only give samples to our Framework.
- The Framework enables the manipulation of our search space.
- The Framework is bounded by the kind of hyper-specialized automaton/ search space:
  - MCA
  - PTA
  - PTA in a half superior lattice (lattice of deterministic automata)
- For each kind of Framework, different implementations are chosen through a Vaucanson-like mechanism.

The Framework (2/3)

- It represents our desire of mixed automaton/partition.
- Mechanism use: a View of an lattice's element from the hyper-specialized automaton (MCA/PTA) and a partition of its states.



The Framework (3/3)

- Although we can work on this view like on an automaton, we can generate a real automaton from the current lattice's element.
- Task to apply
- Ideal for incremental algorithm (history, ...)
- Thanks to the Framework, automata's Walkers become lattice's Walkers
  $\Rightarrow$ it's easier to implement algorithm in Mical

Conclusion

- No previous existing library for regular grammar inference $\Rightarrow$ Mical is the only one

- Many algorithms are written but we can complete Mical very easily thanks to the patterns.

Questions

- Q.: Why “Mical” ?
  A.: The priest Mical was contemporary to Jacques de Vaucanson...

  Q.: How the RPNI works ?
  Q.: It exists other way than the oracle to compute an “efficient” RIG ?