Generic Automata

Julian Verdurmen, Rui Barbosa, Vali Georgescul

June 26th 2009

Outline

- Introduction
- F automata
- Tree automata
- Conclusion

Introduction

- Two generalizations of automata :
  - Generalize shape of transition
    DFA : $a \to s \to s$
    NFA : $a \to s \to \wp\ s$
    to
    f Automata : $a \to s \to f\ s$
  - Generalize language (input) shape :
    from recognizing strings to recognizing trees

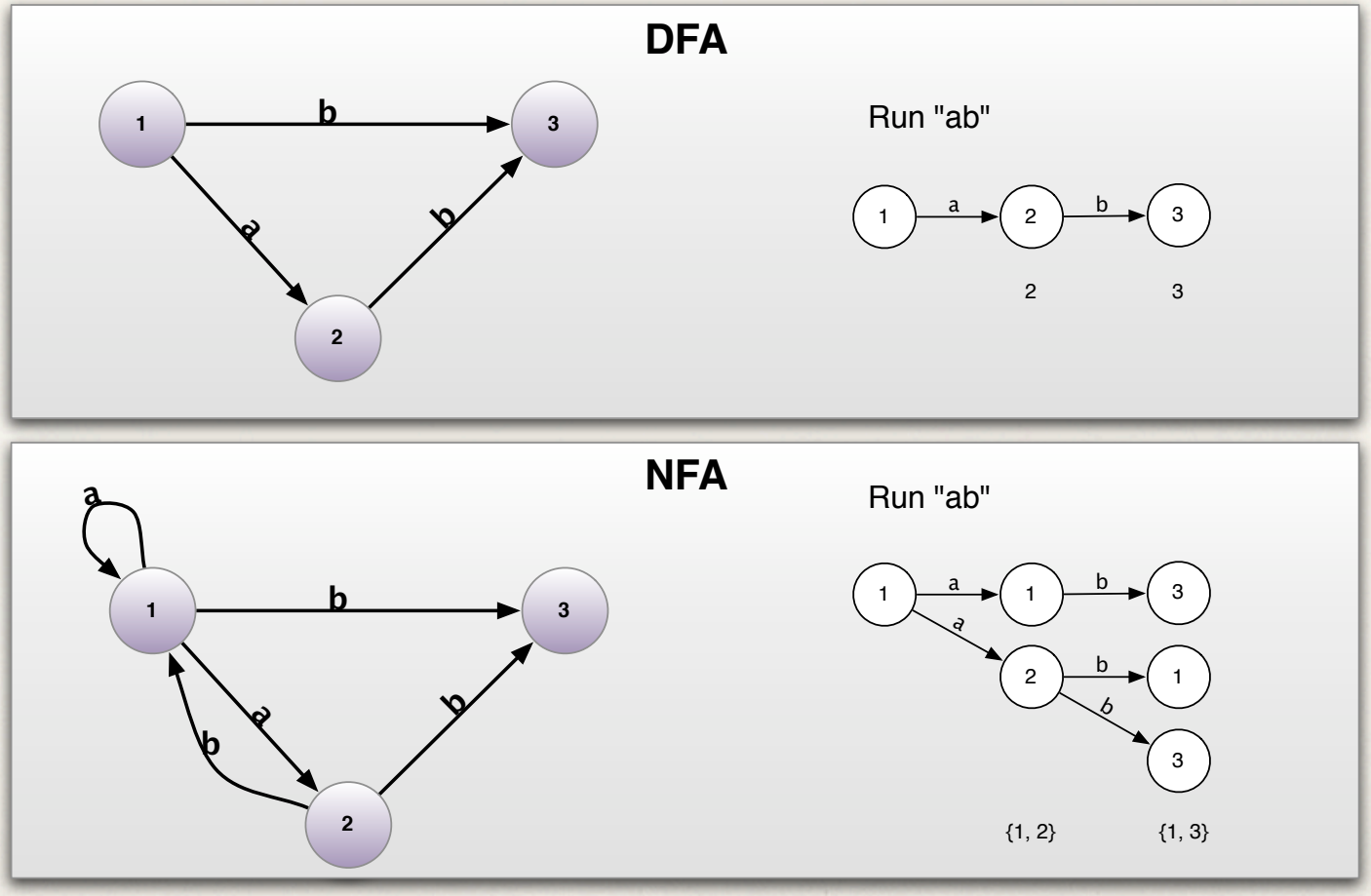
F Automata - examples

- Classical automata
  CDFA : a → s → s
  DFA : a → s → 1 + s (= Maybe s)
  NFA : a → s → ℘ s (= Set s)

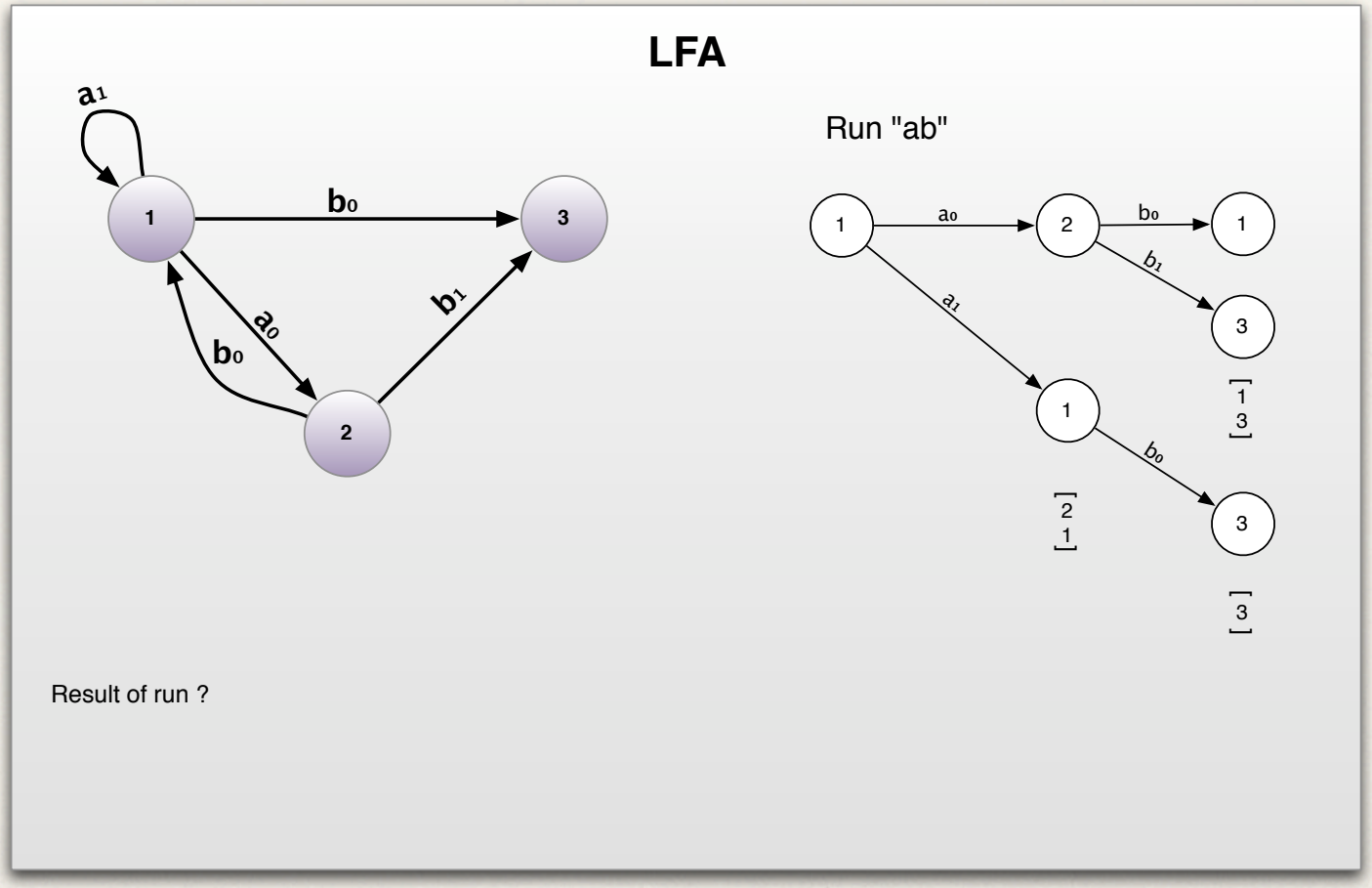
- LFA : a → s → [s] (if we want order)

- F Automata
  A F : a → s → F a

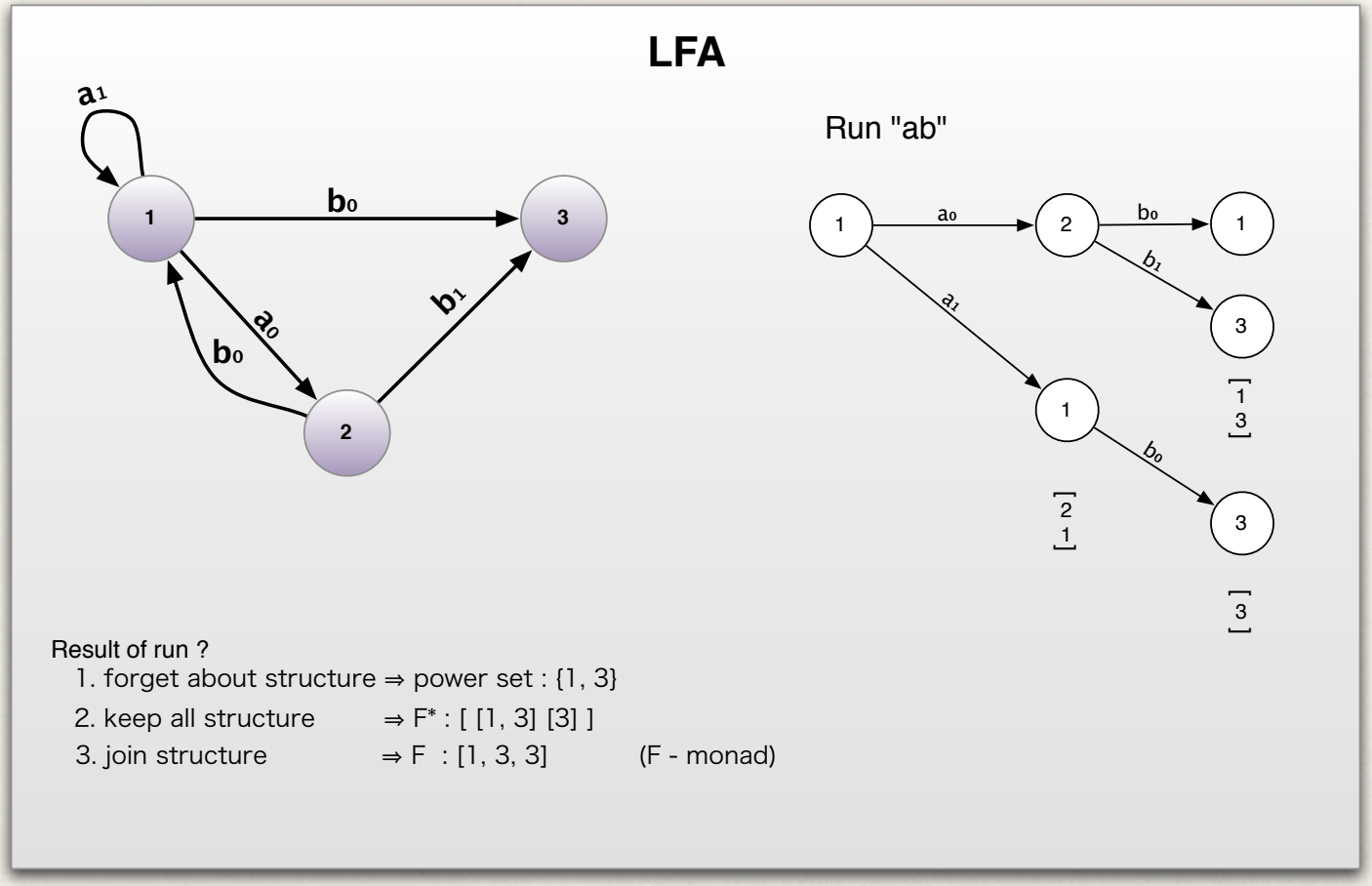
run


DFA
1
b
3
a
b
2
Run "ab"
1
a
2
b
3
2
3
NFA
a
1
b
3
a
b
2
Run "ab"
1
a
1
b
3
a
2
b
1
b
3
{1, 2}
{1, 3}

run


LFA
a1
1
b0
3
a0
b1
b0
2
Run "ab"
1
a0
2
b0
1
b1
3
a1
1
b0
3
[1 3]
[2 1]
[3]
Result of run ?

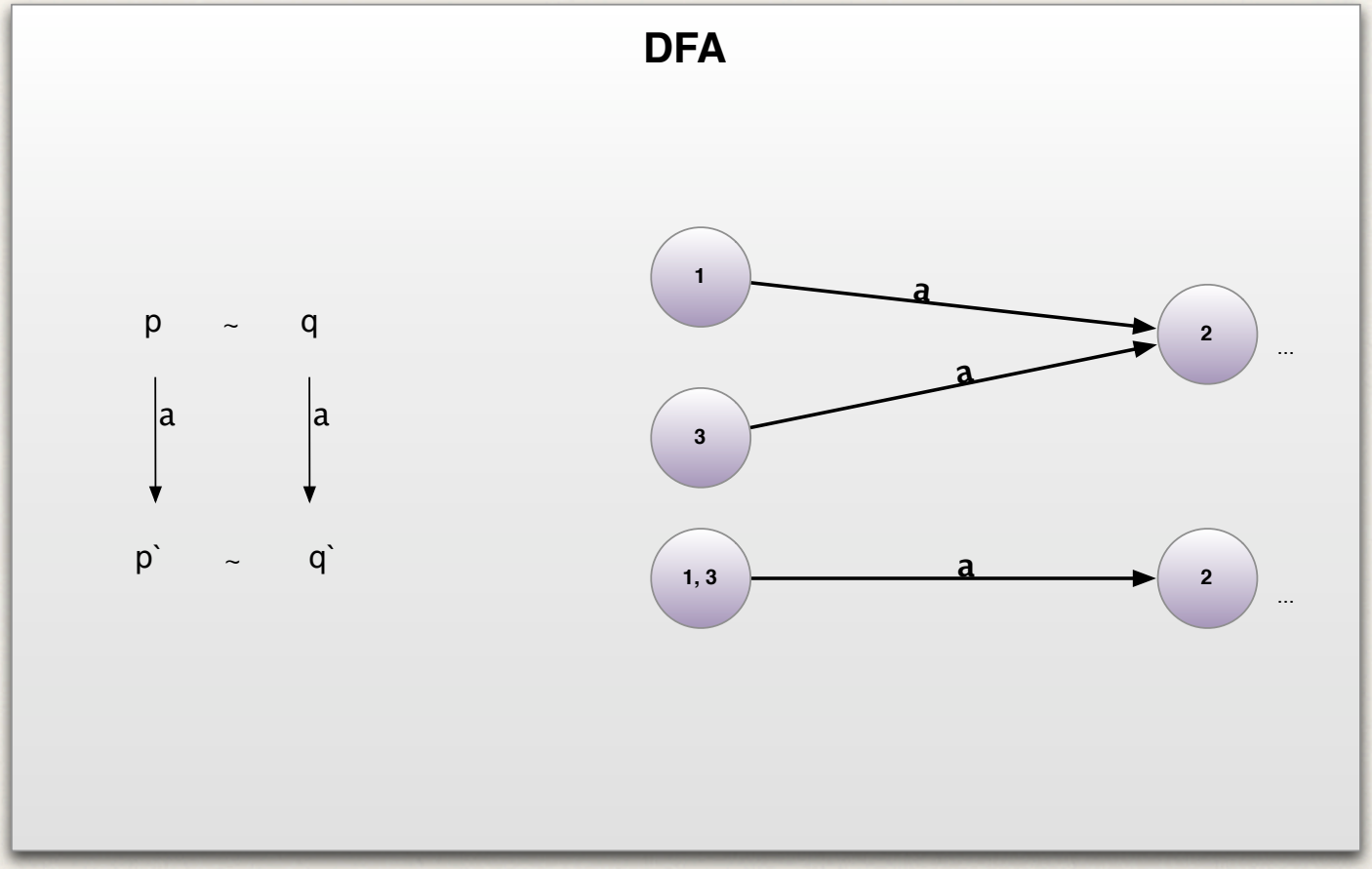
run



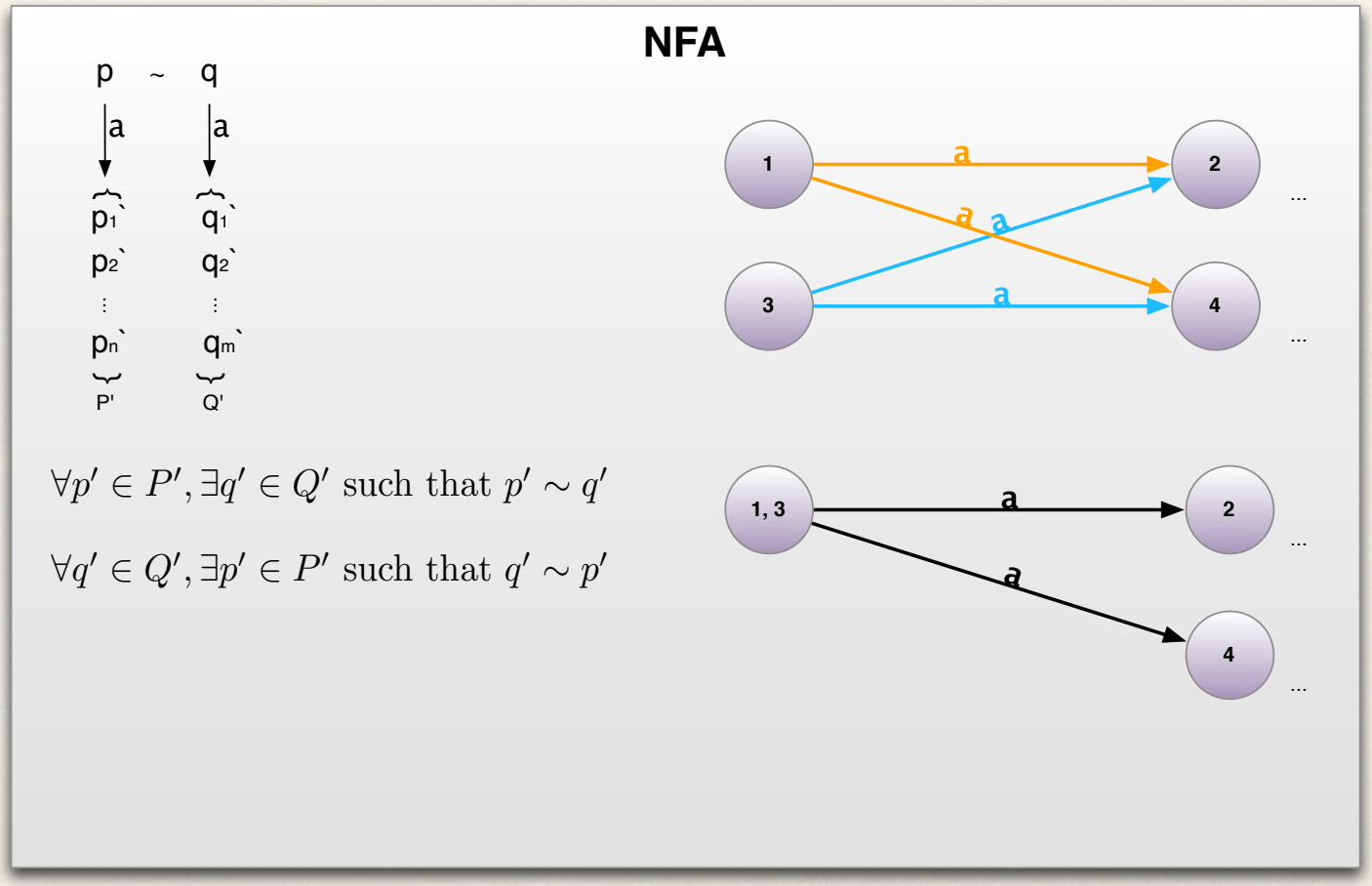
Result of run ?

1. forget about structure ⇒ power set : {1, 3}
2. keep all structure ⇒ F* : [[1, 3] [3]]
3. join structure ⇒ F : [1, 3, 3] (F - monad)

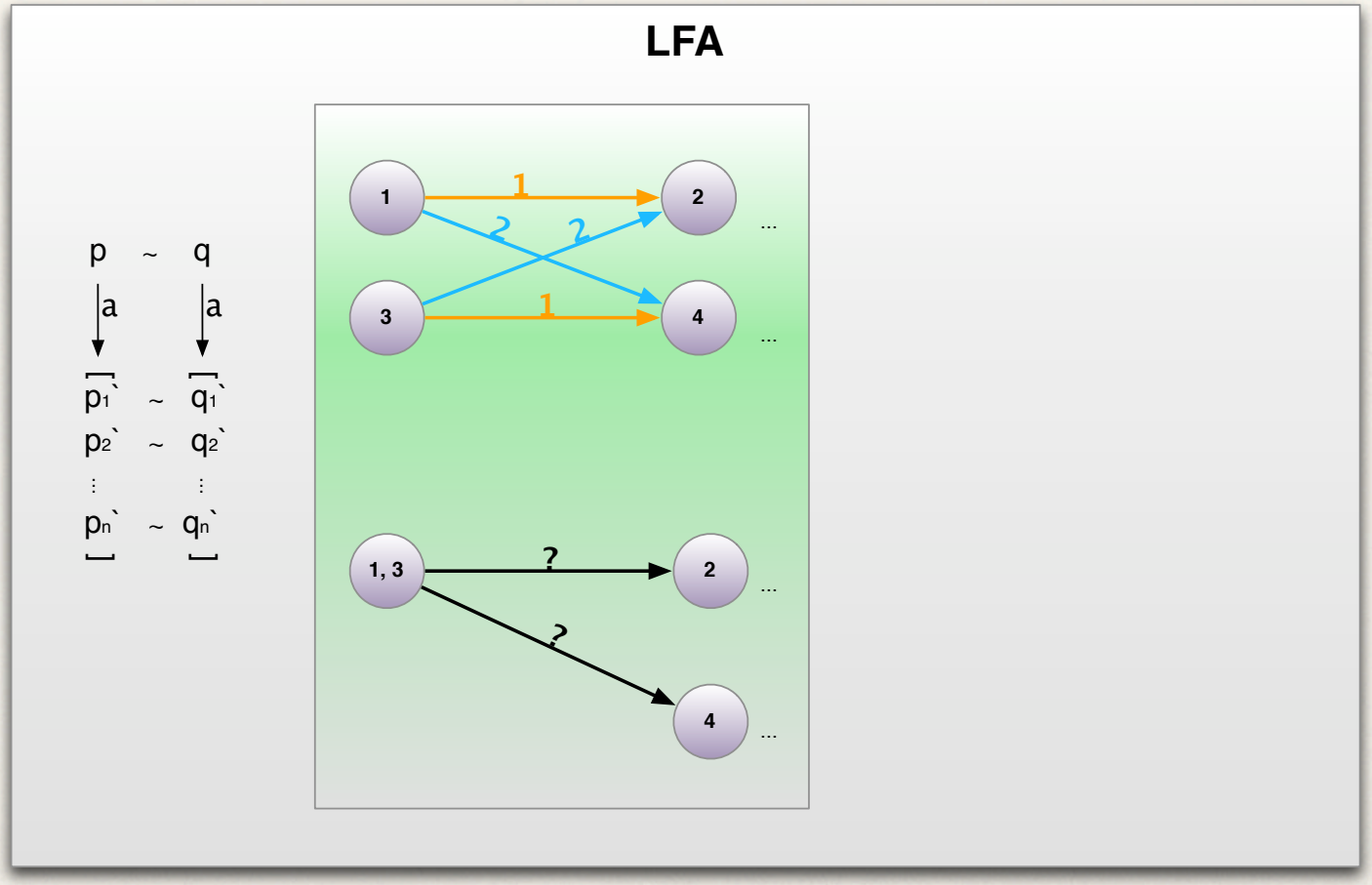
Bi-similarity - DFA



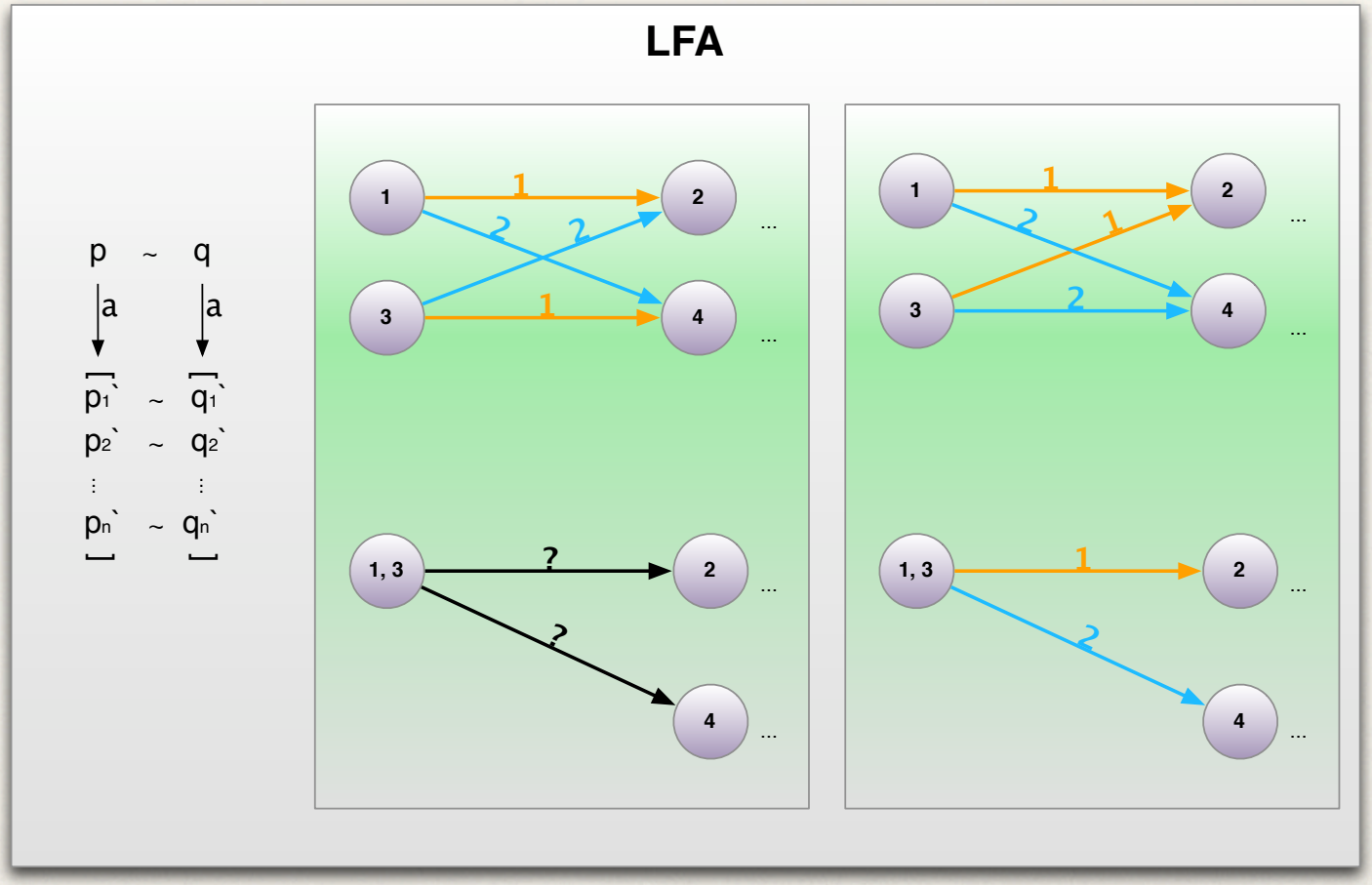
Bi-similarity - NFA



Bi-similarity - F automaton



Bi-similarity - F automaton



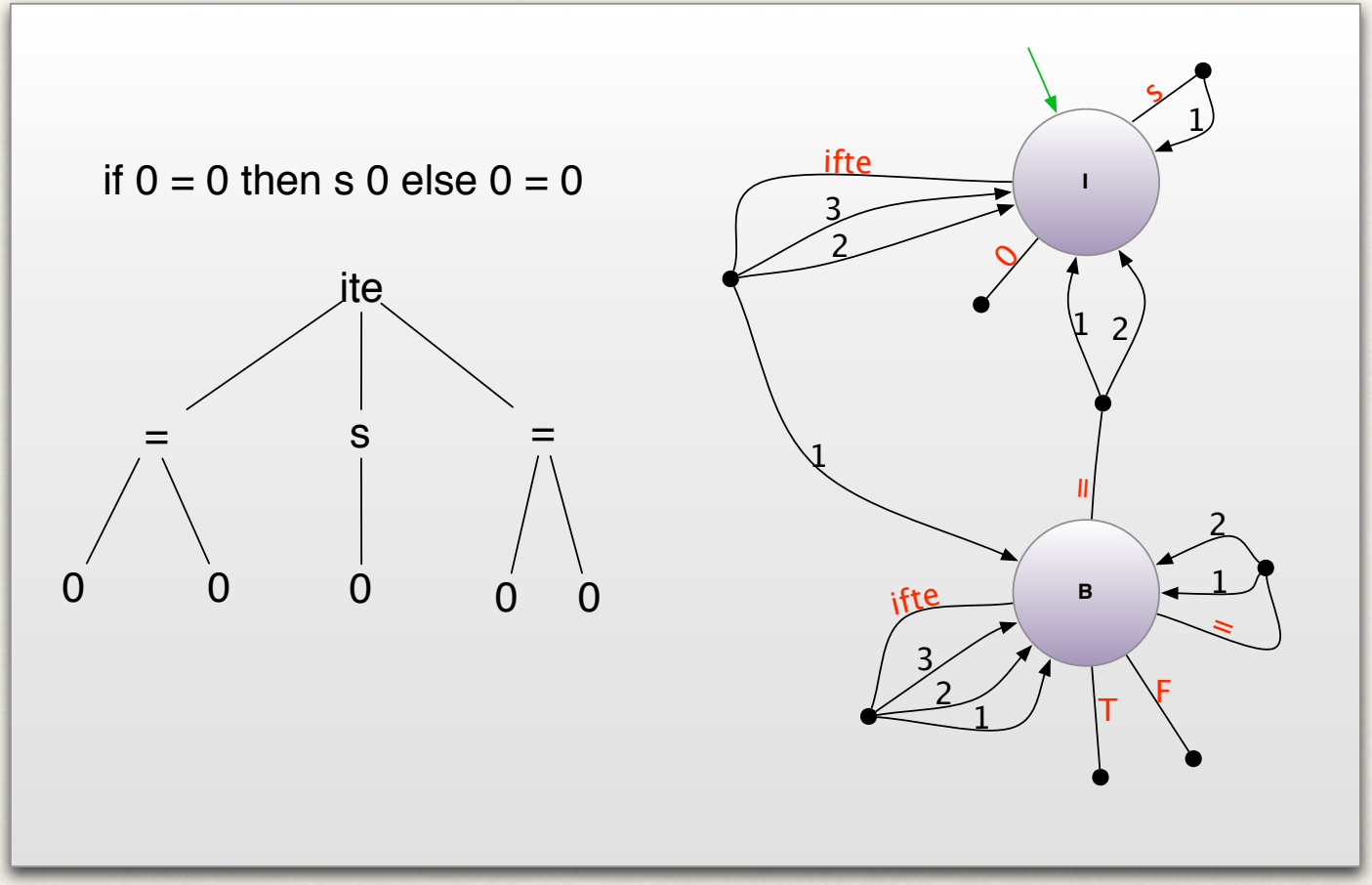
Implementation

Tree automata - Type checking expressions

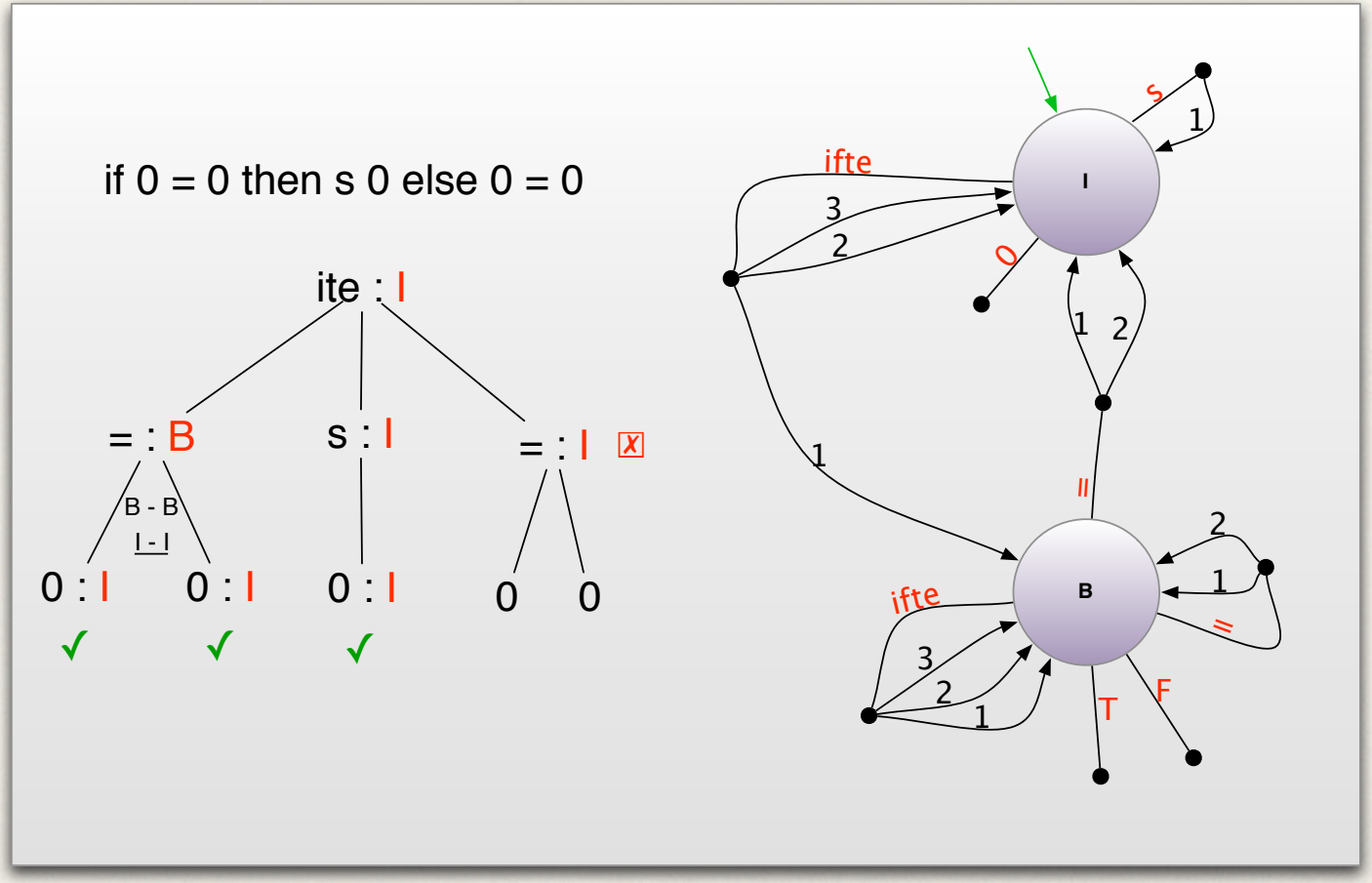
- data E = E + E
  | E ∧ E
  | ¬ E
  | if E then E else E
  | F
  | T
  | 0
  | s E
  | E = E

symbol	arity
+	2
∧	2
¬	1
ifte	3
F	0
T	0
s	1
0	0
=	2

Tree automata - Type checking expressions



Tree automata - Type checking expressions



Implementation - Definitions

- put here page1, then page 2 and then page 3

Implementation

- Used Generic Combinators (map, crush, zips)
- Available in Several Libraries
- Used Regular
  (we wanted a functorial view)
- Problem: access type parameters

Conclusion