Techniques for Unambiguous Systems

Wojciech Czerwiński

GandALF 2022

Plan

Plan

- unambiguity

Plan

- unambiguity
- equivalence of FA

Plan

- unambiguity
- equivalence of FA
  - weighted automata (Schutzenberger `61)

Plan

- unambiguity
- equivalence of FA
  - weighted automata (Schutzenberger `61)
- equivalence of VASSes

Plan

- unambiguity
- equivalence of FA
  - weighted automata (Schutzenberger `61)
- equivalence of VASSes
  - lookahead technique (Cz., Hofman `22)

Plan

- unambiguity
- equivalence of FA
  - weighted automata (Schutzenberger `61)
- equivalence of VASSes
  - lookahead technique (Cz., Hofman `22)
- connections to other fields

Unambiguity

Unambiguity

for each word there is at most one accepting run

Unambiguity

for each word there is at most one accepting run

more expressive than deterministic

Unambiguity

for each word there is at most one accepting run

more expressive than deterministic

many problems become simpler

Unambiguity

for each word there is at most one accepting run

more expressive than deterministic

many problems become simpler

mathematically interesting

Unambiguity

for each word there is at most one accepting run

more expressive than deterministic

many problems become simpler

mathematically interesting

recently a lot of research

Simplicity

Simplicity

many problems become simpler:

Simplicity

many problems become simpler:

universality for UFA (PTime vs PSpace)

Simplicity

many problems become simpler:

universality for UFA (PTime vs PSpace)

equivalence for UFA (PTime vs PSpace)

Simplicity

many problems become simpler:

universality for UFA (PTime vs PSpace)

equivalence for UFA (PTime vs PSpace)

equivalence for URA (ExpTime vs undec)

Simplicity

many problems become simpler:

universality for UFA (PTime vs PSpace)

equivalence for UFA (PTime vs PSpace)

equivalence for URA (ExpTime vs undec)

equivalence for UVASS (Ackermann vs undec)

UFA equivalence

UFA equivalence

reduction to weighted automata

UFA equivalence

reduction to weighted automata

reduction to zeroness problem

UFA equivalence

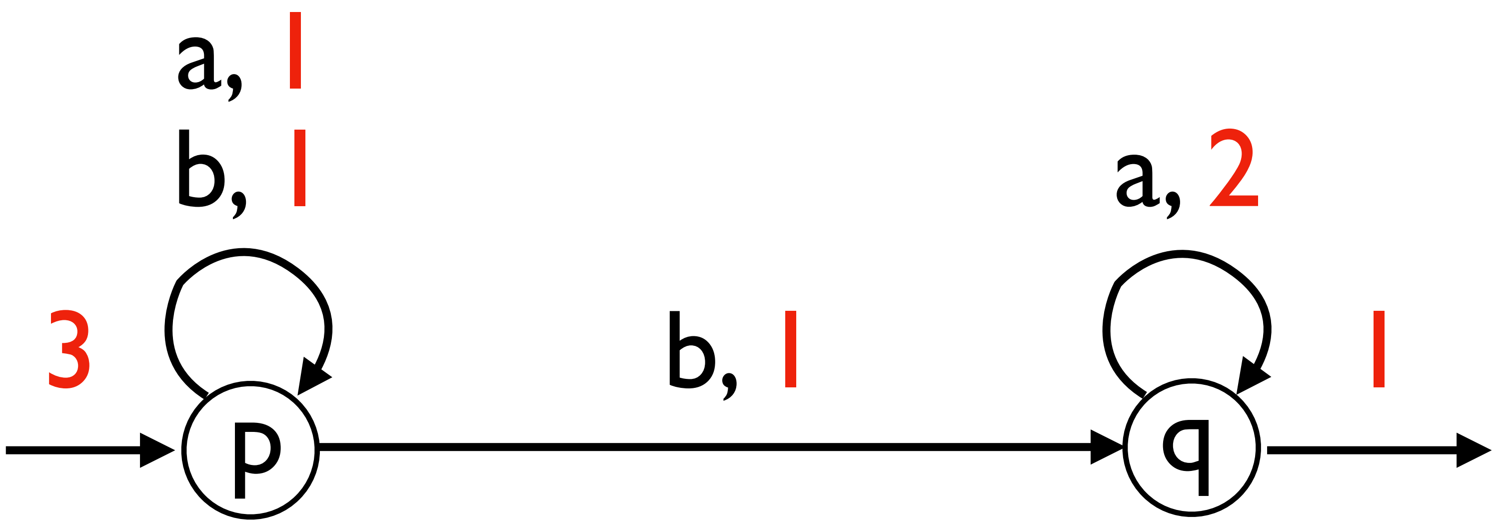
reduction to weighted automata

reduction to zeroness problem

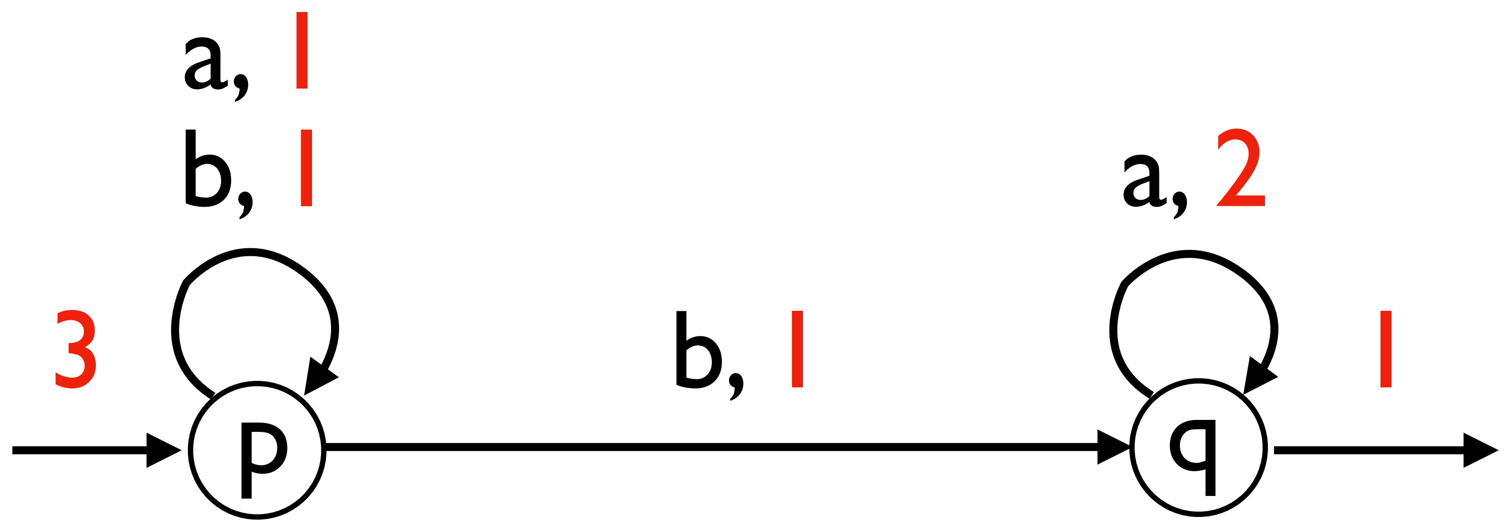
solving zeroness

Weighted automata

Weighted automata

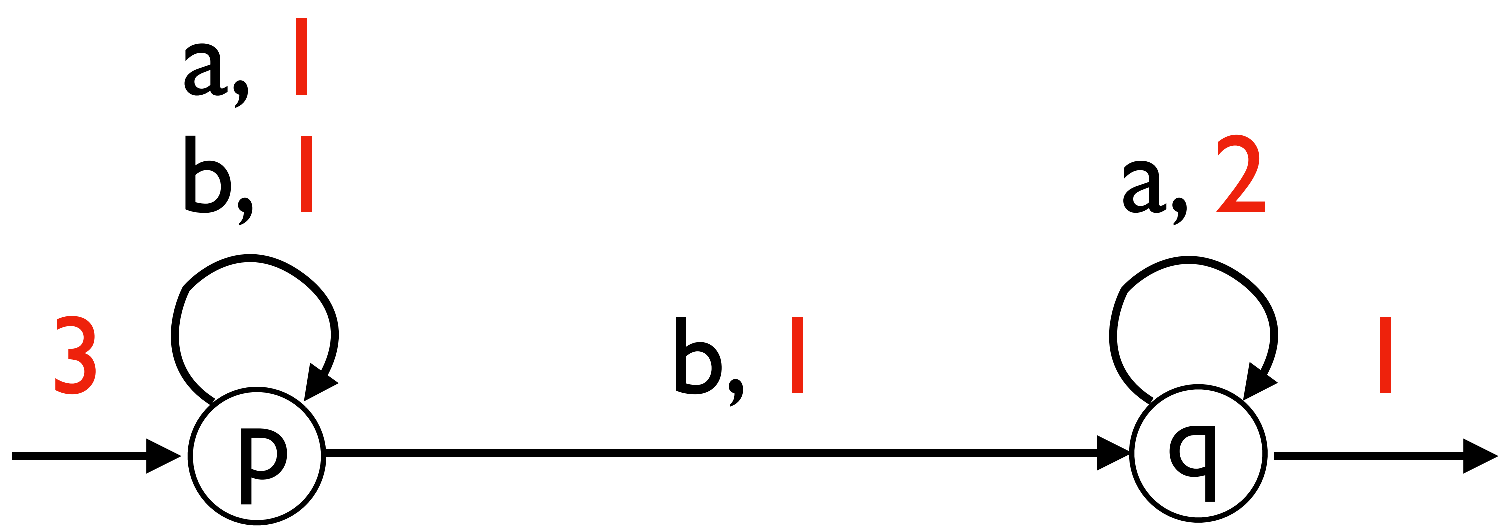


Weighted automata



weight of a run = product of transition weights

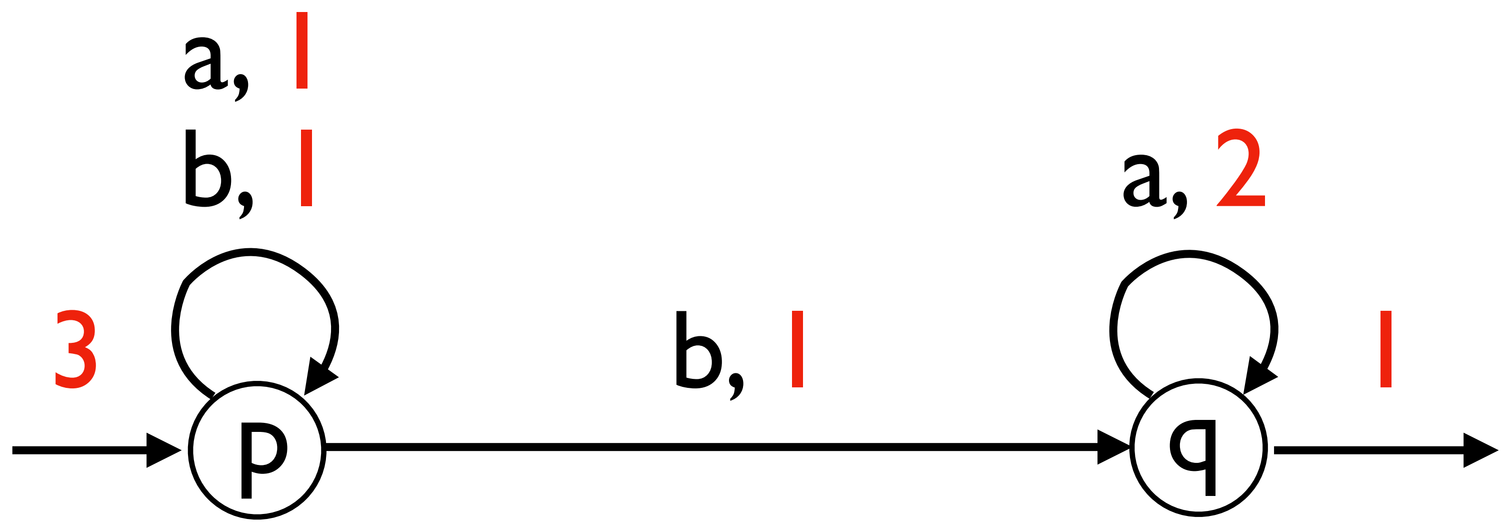
Weighted automata



weight of a run = product of transition weights

weight of a word = sum of run weights

Weighted automata



weight of a run = product of transition weights

weight of a word = sum of run weights

w(abbaabaaa) = 24

UFA to WA

UFA to WA

L(A) = L(B) ⇔ A' and B' equivalent

UFA to WA

$L(A) = L(B) \Leftrightarrow A'$ and B' equivalent

each transition has weight 1

UFA to WA

$L(A) = L(B) \Leftrightarrow A'$ and B' equivalent

each transition has weight 1

input weights of initial states are 1

UFA to WA

L(A) = L(B) ⇔ A' and B' equivalent

each transition has weight 1

input weights of initial states are 1

output weights of final states are 1

Zeroness

Zeroness

A’ and B’ equivalent $\Leftrightarrow$ A’(w)-B’(w) is constantly zero

Zeroness

A’ and B’ equivalent ⇔ A’(w)-B’(w) is constantly zero

construct C such that C(w) = A’(w)-B’(w)

Zeroness

A' and B' equivalent ⇔ A'(w)-B'(w) is constantly zero

construct C such that C(w) = A'(w)-B'(w)

enough to check zeroness for WA!

Zeroness

A’ and B’ equivalent ⇔ A’(w)-B’(w) is constantly zero

construct C such that C(w) = A’(w)-B’(w)

enough to check zeroness for WA!

solves also multiplicity equivalence!

Solving zeroness

Solving zeroness

Fix A with n states

Solving zeroness

Fix A with n states

Consider vec(w) in $\mathbb{Z}^n$

Solving zeroness

Fix A with n states

Consider $\mathrm{vec}(w)$ in $\mathbb{Z}^n$

Let V_k be spanned by $\mathrm{vec}(w)$ for words of length at most k

Solving zeroness

Fix A with n states

Consider $\mathrm{vec}(w)$ in $\mathbb{Z}^n$

Let V_k be spanned by $\mathrm{vec}(w)$ for words of length at most k

$$\mathrm{weight}(w) = \mathrm{out}(\mathrm{vec}(w))$$

Solving zeroness

Fix A with n states

Consider $\mathrm{vec}(w)$ in $\mathbb{Z}^n$

Let V_k be spanned by $\mathrm{vec}(w)$
for words of length at most k

$$\mathrm{weight}(w) = \mathrm{out}(\mathrm{vec}(w))$$

$$A \text{ is zero} \Leftrightarrow \text{all } V_k \subseteq \ker(\mathrm{out})$$

Solving zeroness

Fix A with n states

Consider vec(w) in $\mathbb{Z}^n$

Let V_k be spanned by vec(w)
for words of length at most k

weight(w) = out(vec(w))

A is zero $\Leftrightarrow$ all $V_k \subseteq$ ker(out)

Enough to compute V_k

Solving zeroness

Solving zeroness

A is zero $\Leftrightarrow$ all $V_k \subseteq \ker(\text{out})$

Solving zeroness

A is zero $\Leftrightarrow$ all $V_k \subseteq \ker(\mathrm{out})$

V_0 is spanned by $\mathrm{vec}(\varepsilon)$

Solving zeroness

A is zero $\Leftrightarrow$ all $V_k \subseteq \ker(\mathrm{out})$

V_0 is spanned by $\mathrm{vec}(\varepsilon)$

compute V_{k+1} from V_k

Solving zeroness

A is zero $\Leftrightarrow$ all $V_k \subseteq \ker(\mathrm{out})$

V_0 is spanned by $\mathrm{vec}(\varepsilon)$

compute V_{k+1} from V_k

V_{k+1} is generated by

Solving zeroness

A is zero $\Leftrightarrow$ all $V_k \subseteq \ker(\mathrm{out})$

V_0 is spanned by $\mathrm{vec}(\varepsilon)$

compute V_{k+1} from V_k

V_{k+1} is generated by generators of V_k after transitions

Solving zeroness

A is zero $\Leftrightarrow$ all $V_k \subseteq \ker(\text{out})$

V_0 is spanned by $\text{vec}(\varepsilon)$

compute V_{k+1} from V_k

V_{k+1} is generated by generators of V_k after transitions

and generators of V_k

Solving zeroness

Solving zeroness

A is zero $\Leftrightarrow$ all $V_k \subseteq \ker(\text{out})$

Solving zeroness

A is zero $\Leftrightarrow$ all $V_k \subseteq \ker(\mathrm{out})$

V_0 is spanned by $\mathrm{vec}(\varepsilon)$

Solving zeroness

A is zero $\Leftrightarrow$ all $V_k \subseteq \ker(\text{out})$

V_0 is spanned by $\text{vec}(\varepsilon)$

either $\dim(V_{k+1}) > \dim(V_k)$

Solving zeroness

A is zero $\Leftrightarrow$ all $V_k \subseteq \ker(\mathrm{out})$

V_0 is spanned by $\mathrm{vec}(\varepsilon)$

either $\dim(V_{k+1}) > \dim(V_k)$

or $V_{k+1} = V_k$

Solving zeroness

A is zero $\Leftrightarrow$ all $V_k \subseteq \ker(\mathrm{out})$

V_0 is spanned by $\mathrm{vec}(\varepsilon)$

either $\dim(V_{k+1}) > \dim(V_k)$

or $V_{k+1} = V_k$

algorithm:

Solving zeroness

A is zero $\Leftrightarrow$ all $V_k \subseteq \ker(\text{out})$

V_0 is spanned by $\text{vec}(\varepsilon)$

either $\dim(V_{k+1}) > \dim(V_k)$

or $V_{k+1} = V_k$

algorithm: compute V_k until stabilisation

Solving zeroness

A is zero ⇔ all $V_k \subseteq \ker(out)$

V_0 is spanned by $vec(\varepsilon)$

either $\dim(V_{k+1}) > \dim(V_k)$

or $V_{k+1} = V_k$

algorithm: compute V_k until stabilisation

check if included in $\ker(out)$

Solving zeroness

A is zero $\Leftrightarrow$ all $V_k \subseteq \ker(\text{out})$

V_0 is spanned by $\text{vec}(\varepsilon)$

either $\dim(V_{k+1}) > \dim(V_k)$ or $V_{k+1} = V_k$

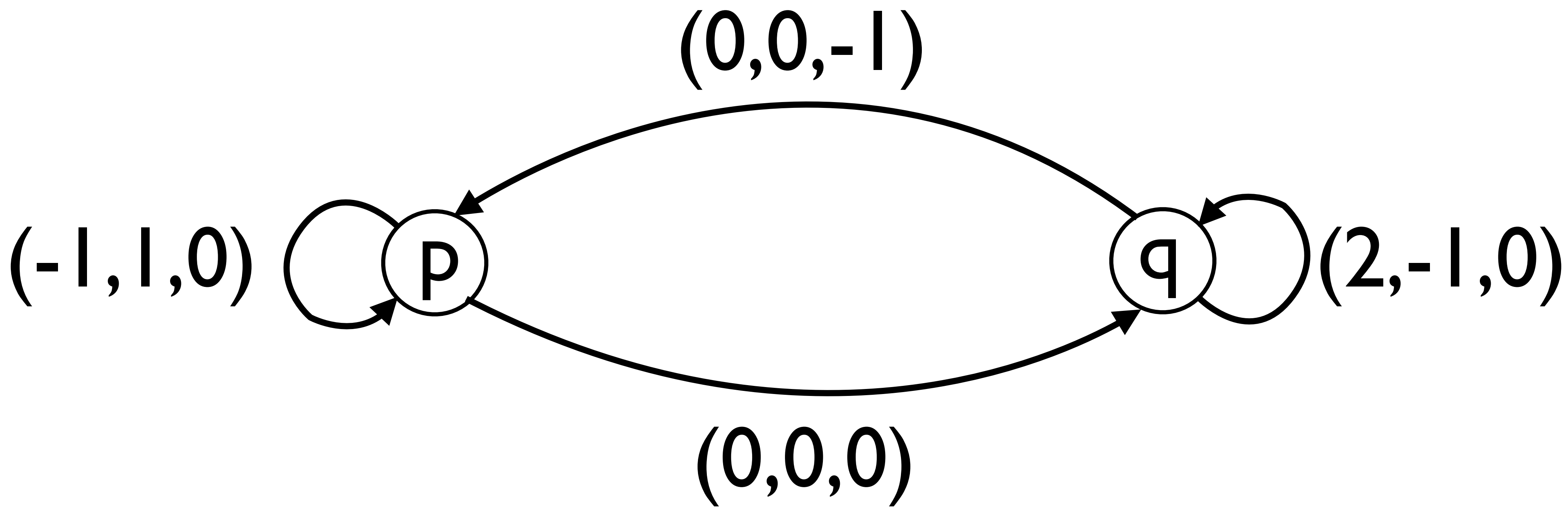
algorithm: compute V_k until stabilisation

check if included in $\ker(\text{out})$

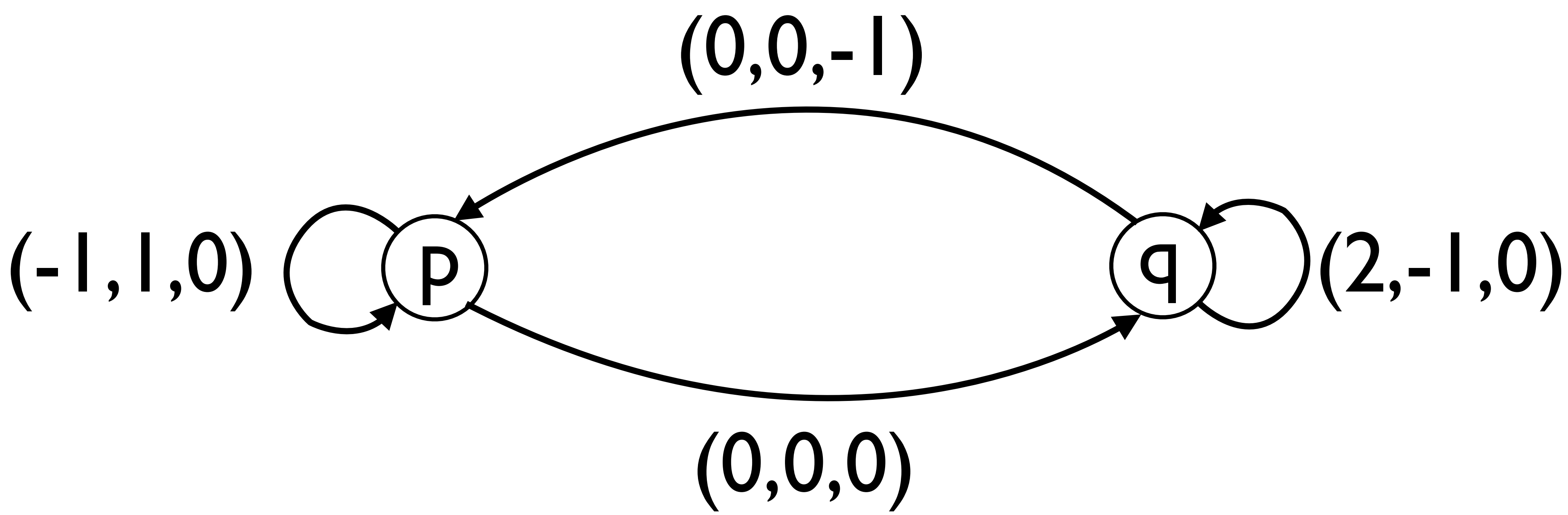
In PTime!

Vector Addition Systems with States

Vector Addition Systems with States

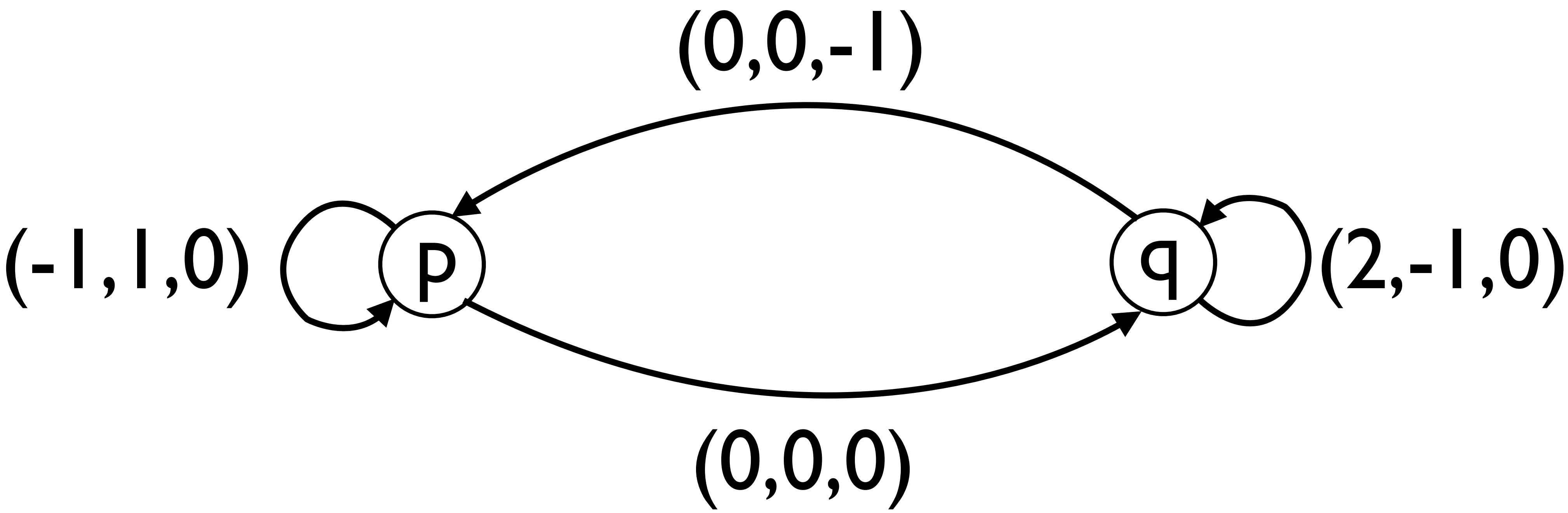


Vector Addition Systems with States



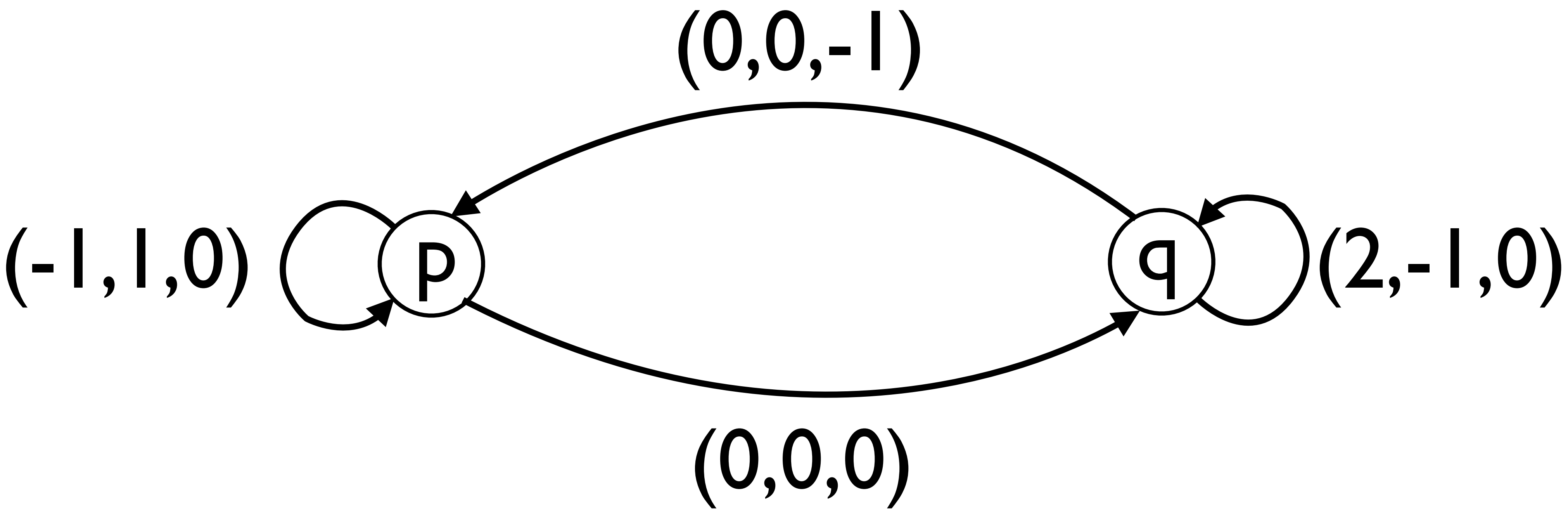
p(2,0,7)

Vector Addition Systems with States



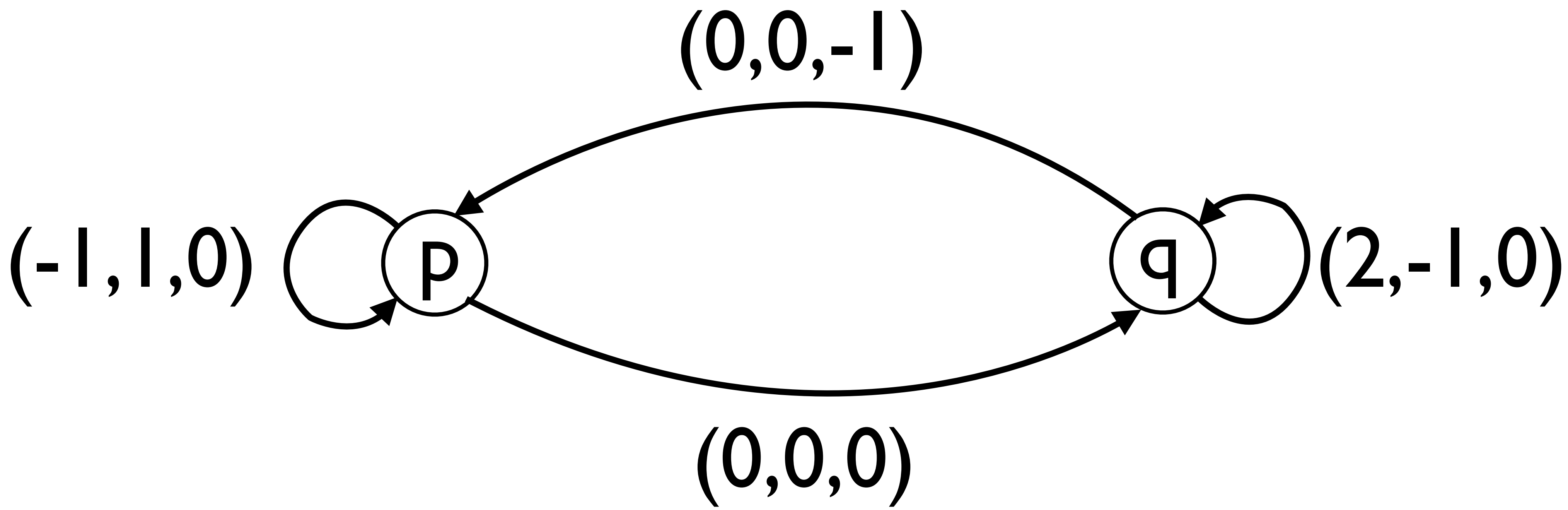
$p(2,0,7) \longrightarrow p(1,1,7)$

Vector Addition Systems with States



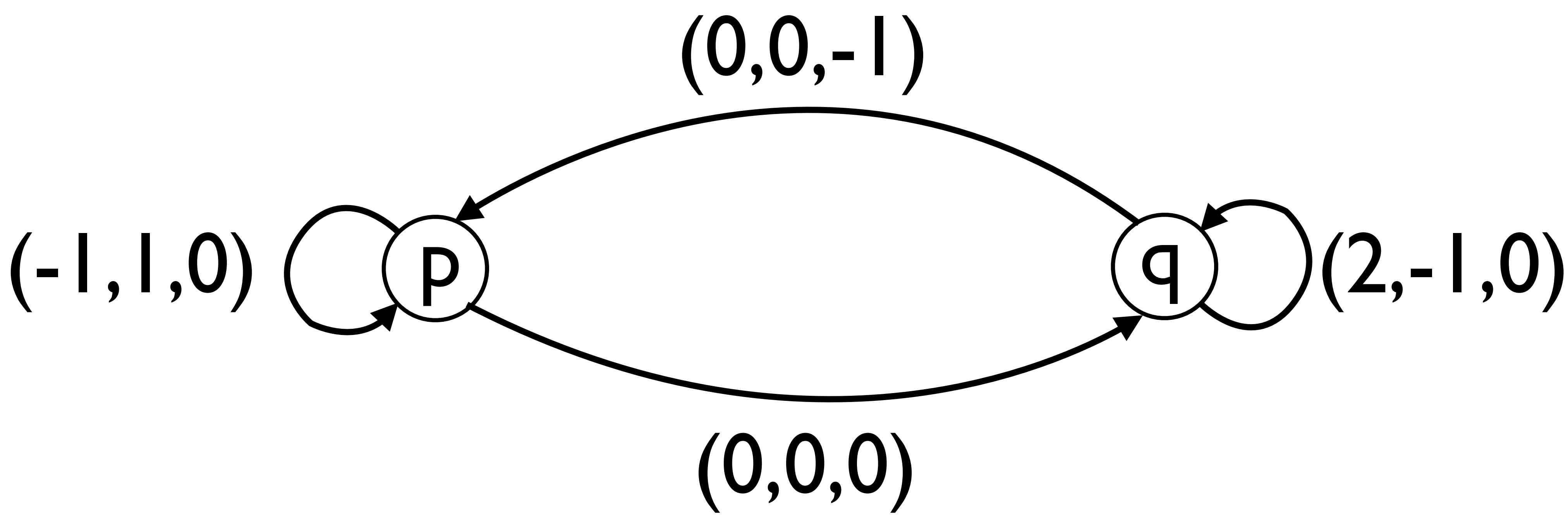
$$p(2,0,7) \longrightarrow p(1,1,7) \longrightarrow p(0,2,7)$$

Vector Addition Systems with States



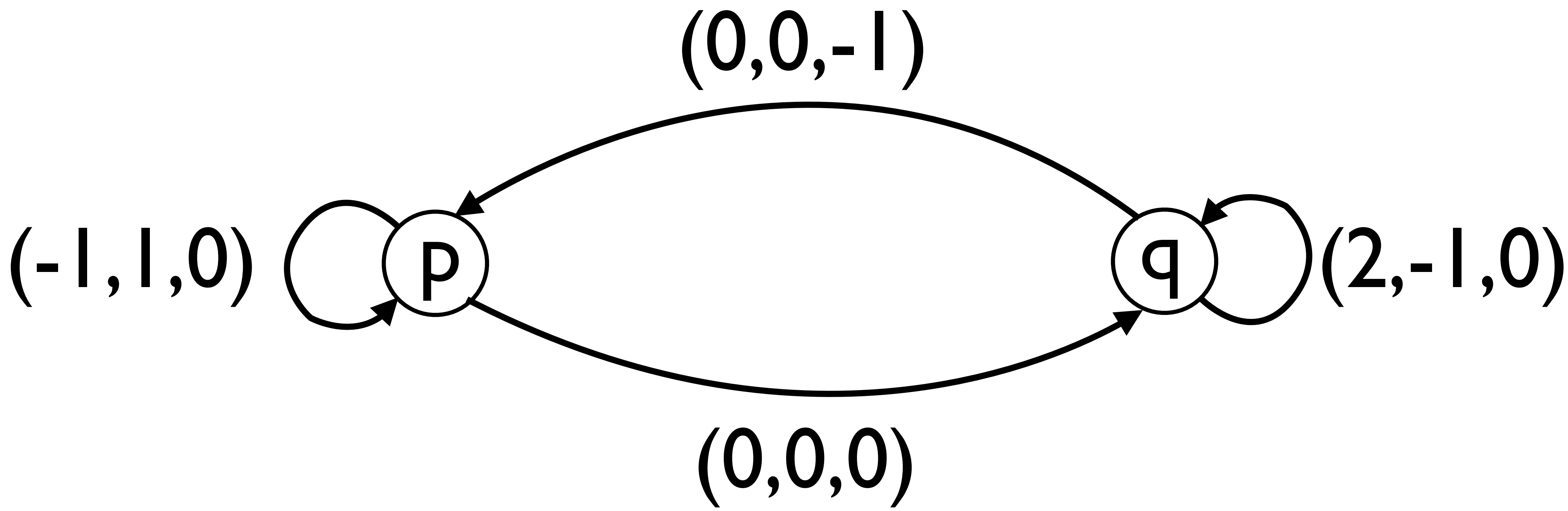
$$p(2,0,7) \longrightarrow p(1,1,7) \longrightarrow p(0,2,7) \longrightarrow q(0,2,7)$$

Vector Addition Systems with States



$$p(2,0,7) \longrightarrow p(1,1,7) \longrightarrow p(0,2,7) \longrightarrow q(0,2,7) \longrightarrow q(2,1,7)$$

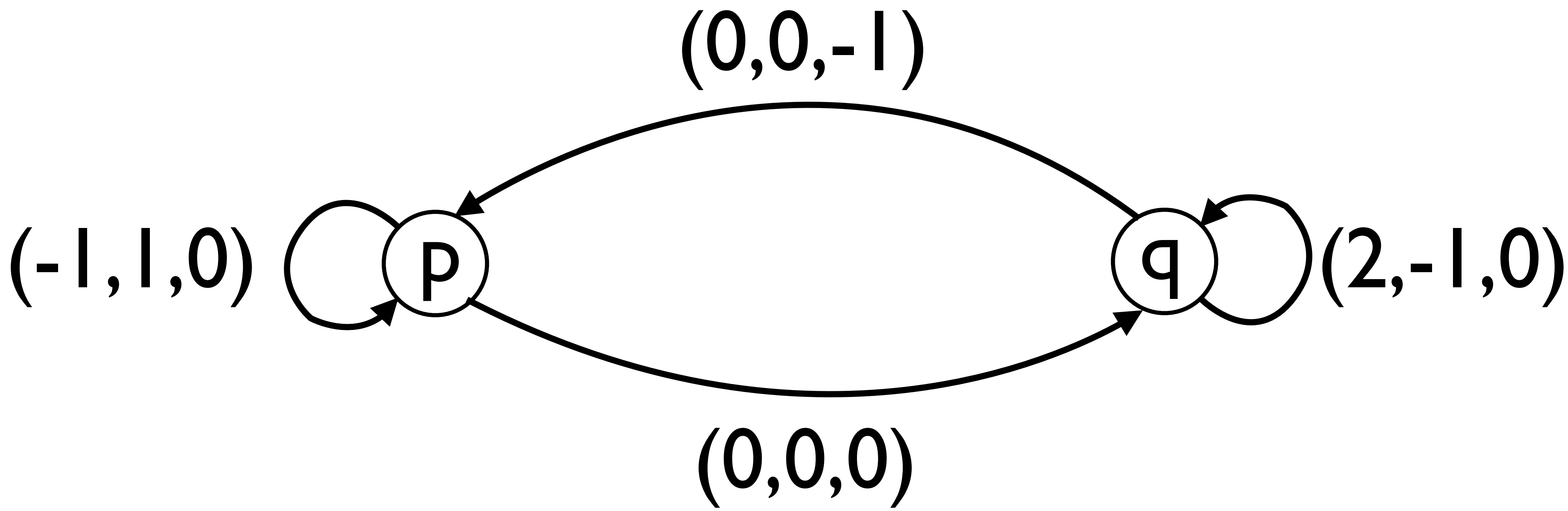
Vector Addition Systems with States



$$p(2,0,7) \longrightarrow p(1,1,7) \longrightarrow p(0,2,7) \longrightarrow q(0,2,7) \longrightarrow q(2,1,7)$$

$$\longrightarrow q(4,0,7)$$

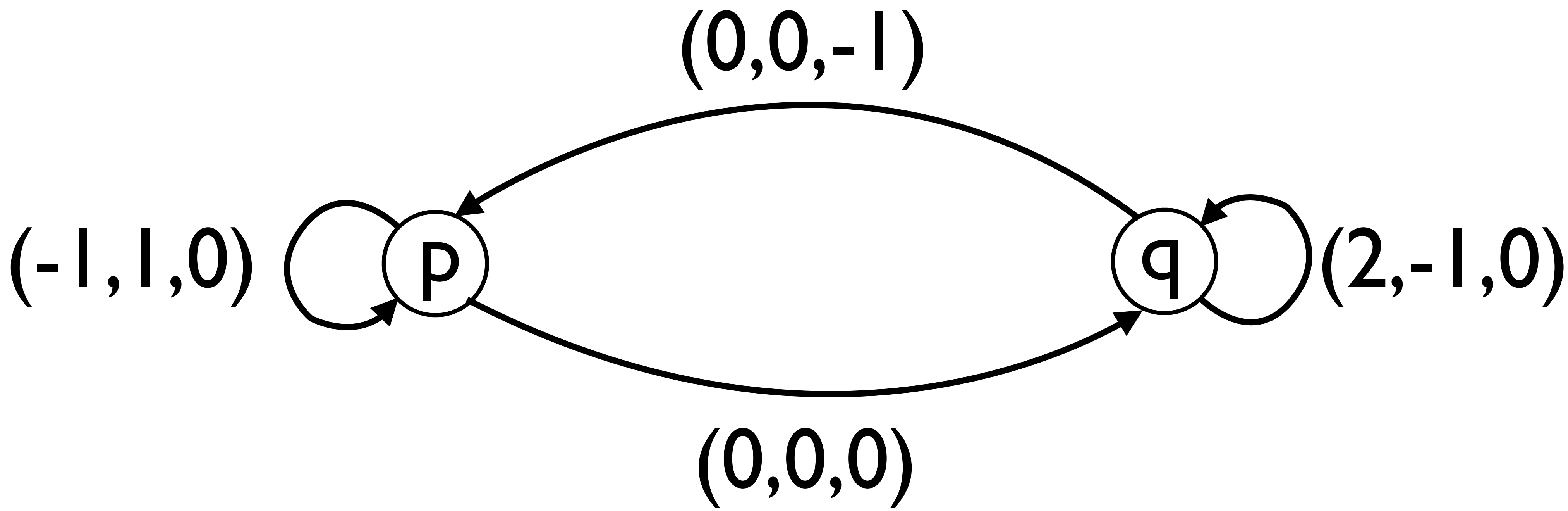
Vector Addition Systems with States



$$p(2,0,7) \longrightarrow p(1,1,7) \longrightarrow p(0,2,7) \longrightarrow q(0,2,7) \longrightarrow q(2,1,7)$$

$$\longrightarrow q(4,0,7) \longrightarrow p(4,0,6)$$

Vector Addition Systems with States



$$p(2,0,7) \longrightarrow p(1,1,7) \longrightarrow p(0,2,7) \longrightarrow q(0,2,7) \longrightarrow q(2,1,7)$$

$$\longrightarrow q(4,0,7) \longrightarrow p(4,0,6)$$

Petri nets

Language of a VASS

Language of a VASS

VASS = vector addition system with states

Language of a VASS

VASS = vector addition system with states

initial configuration, acceptance by states

Language of a VASS

VASS = vector addition system with states

initial configuration, acceptance by states

detVASS = each reachable configuration is deterministic

Equivalence for VASS

Equivalence for VASS

for 2-dimensional VASSes - undecidable

Equivalence for VASS

for 2-dimensional VASSes - undecidable

for deterministic VASSes - decidable

Equivalence for VASS

for 2-dimensional VASSes - undecidable

for deterministic VASSes - decidable

unambiguous?

Equivalence for VASS

for 2-dimensional VASSes - undecidable

for deterministic VASSes - decidable

unambiguous?

decidable, Ackermann-complete

Equivalence for VASS

for 2-dimensional VASSes - undecidable

for deterministic VASSes - decidable

unambiguous?

decidable, Ackermann-complete   CONCUR 2022

Zeroness?

Zeroness?

zeroness decidable?

Zeroness?

zeroness decidable?

multiplicity equivalence undecidable for VASSes

Zeroness?

zeroness decidable?

multiplicity equivalence undecidable for VASSes

Petr Jancar 2001
Nonprimitive recursive complexity and undecidability for Petri net equivalences

Zeroness?

zeroness decidable?

multiplicity equivalence undecidable for VASSes

Petr Jancar 2001
Nonprimitive recursive complexity and undecidability for Petri net equivalences

reduction to a deterministic case!

Idea

Idea

add information to get determinism

Idea

add information to get determinism lookahead

Idea

add information to get determinism lookahead

does the suffix belong to several fixed regular languages?

Idea

add information to get determinism lookahead

does the suffix belong to several fixed regular languages?

best captured by a monoid

Idea

add information to get determinism   lookahead

does the suffix belong to several fixed regular languages?

best captured by a monoid $h: \Sigma^* \to M$

Idea

add information to get determinism lookahead

does the suffix belong to several fixed regular languages?

best captured by a monoid $h: \Sigma^* \to M$

For each regular L over Σ
there are $h: \Sigma^* \to M$ and $F \subseteq M$
such that $L = h^{-1}(F)$.

Reduction

Reduction

Let h: $\Sigma^* \rightarrow M$

Reduction

Let h: $\Sigma^* \to M$ Then map_h: $\Sigma^* \to (\Sigma \times M)^*$

Reduction

Let $h: \Sigma^* \rightarrow M$ Then $map_h: \Sigma^* \rightarrow (\Sigma \times M)^*$

$map_h(a\ b\ c) = (*, h(abc))\ (a, h(bc))\ (b, h(c))\ (c, h(\varepsilon))$

Reduction

Let $h\colon \Sigma^* \to M$ Then $map_h\colon \Sigma^* \to (\Sigma \times M)^*$

$$map_h(a\ b\ c) = (*, h(abc))\ (a, h(bc))\ (b, h(c))\ (c, h(\varepsilon))$$

Claim 1: For each $h\colon \Sigma^* \to M$ it holds

$$K \subseteq L \Leftrightarrow map_h(K) \subseteq map_h(L)$$

Reduction

Let $h: \Sigma^* \to M$ Then $map_h: \Sigma^* \to (\Sigma \times M)^*$

$map_h(a\ b\ c) = (*, h(abc))\ (a, h(bc))\ (b, h(c))\ (c, h(\varepsilon))$

Claim 1: For each $h: \Sigma^* \to M$ it holds
$K \subseteq L \Leftrightarrow map_h(K) \subseteq map_h(L)$

Claim 2: For each $h: \Sigma^* \to M$ and each VASS V one can construct another VASS V_h accepting $map_h(L(V))$

Reduction

Let $h\colon \Sigma^* \to M$ Then $map_h\colon \Sigma^* \to (\Sigma \times M)^*$

$$map_h(a\ b\ c) = (*, h(abc))\ (a, h(bc))\ (b, h(c))\ (c, h(\varepsilon))$$

Claim 1: For each $h\colon \Sigma^* \to M$ it holds

$$K \subseteq L \Leftrightarrow map_h(K) \subseteq map_h(L)$$

Claim 2: For each $h\colon \Sigma^* \to M$ and each VASS V one can construct another VASS V_h accepting $map_h(L(V))$

Goal: choose h such that both V_h are deterministic

Reduction

Reduction

Claim 2: For each $h: \Sigma^* \to M$ and each VASS V one can construct another VASS V_h accepting $map_h(L(V))$

Reduction

Claim 2: For each $h: \Sigma^* \to M$ and each VASS V one can construct another VASS V_h accepting $map_h(L(V))$

States are pairs in
$Q \times M \cup \{(init, \$)\}$

Reduction

Claim 2: For each $h: \Sigma^* \to M$ and each VASS V one can construct another VASS V_h accepting $map_h(L(V))$

States are pairs in $Q \times M \cup \{(init, \$)\}$

initial is $(init, \$)$

Reduction

Claim 2: For each $h: \Sigma^* \to M$ and each VASS V one can construct another VASS V_h accepting $\mathrm{map}_h(L(V))$

States are pairs in
$Q \times M \cup \{(\mathrm{init}, \$)\}$

initial is $(\mathrm{init}, \$)$

final are $F \times \{h(\varepsilon)\}$

Reduction

Claim 2: For each $h: \Sigma^* \rightarrow M$ and each VASS V one can construct another VASS V_h accepting $map_h(L(V))$

States are pairs in $Q \times M \cup \{(init, \$)\}$

initial is $(init, \$)$

final are $F \times \{h(\varepsilon)\}$

For a transition

Reduction

Claim 2: For each $h: \Sigma^* \to M$ and each VASS V one can construct another VASS V_h accepting $map_h(L(V))$

States are pairs in $Q \times M \cup \{(init, \$)\}$

initial is $(init, \$)$

final are $F \times \{h(\varepsilon)\}$

For a transition $p \xrightarrow[v]{a} q$

Reduction

Claim 2: For each $h: \Sigma^* \to M$ and each VASS V one can construct another VASS V_h accepting $map_h(L(V))$

States are pairs in $Q \times M \cup \{(init, \$)\}$

initial is $(init, \$)$

final are $F \times \{h(\varepsilon)\}$

For a transition $p \xrightarrow[v]{a} q$

We create

Reduction

Claim 2: For each $h: \Sigma^* \rightarrow M$ and each VASS V one can construct another VASS V_h accepting $map_h(L(V))$

States are pairs in $Q \times M \cup \{(init, \$)\}$

initial is $(init, \$)$

final are $F \times \{h(\varepsilon)\}$

For a transition $p \xrightarrow[v]{a} q$

We create $(p,m) \xrightarrow[v]{(a,m')} (q,m')$

Reduction

Claim 2: For each $h: \Sigma^* \to M$ and each VASS V one can construct another VASS V_h accepting $map_h(L(V))$

States are pairs in $Q \times M \cup \{(init, \$)\}$

initial is $(init, \$)$

final are $F \times \{h(\varepsilon)\}$

For a transition $p \xrightarrow[v]{a} q$

We create $(p,m) \xrightarrow[v]{(a,m')} (q,m')$

if $m = h(a)\ m'$ or $m = \$, a = *$

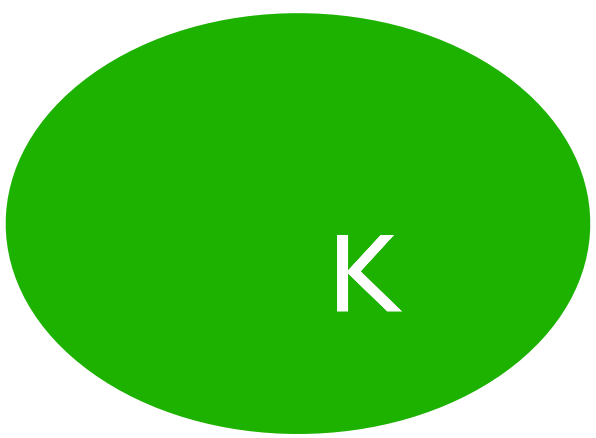
Separability

Separability

Languages K and L are regular-separable
if there is a regular S such that
$K \subseteq S$ and $L \cap S = \varnothing$.

Separability

Languages K and L are regular-separable
if there is a regular S such that
$K \subseteq S$ and $L \cap S = \varnothing$.

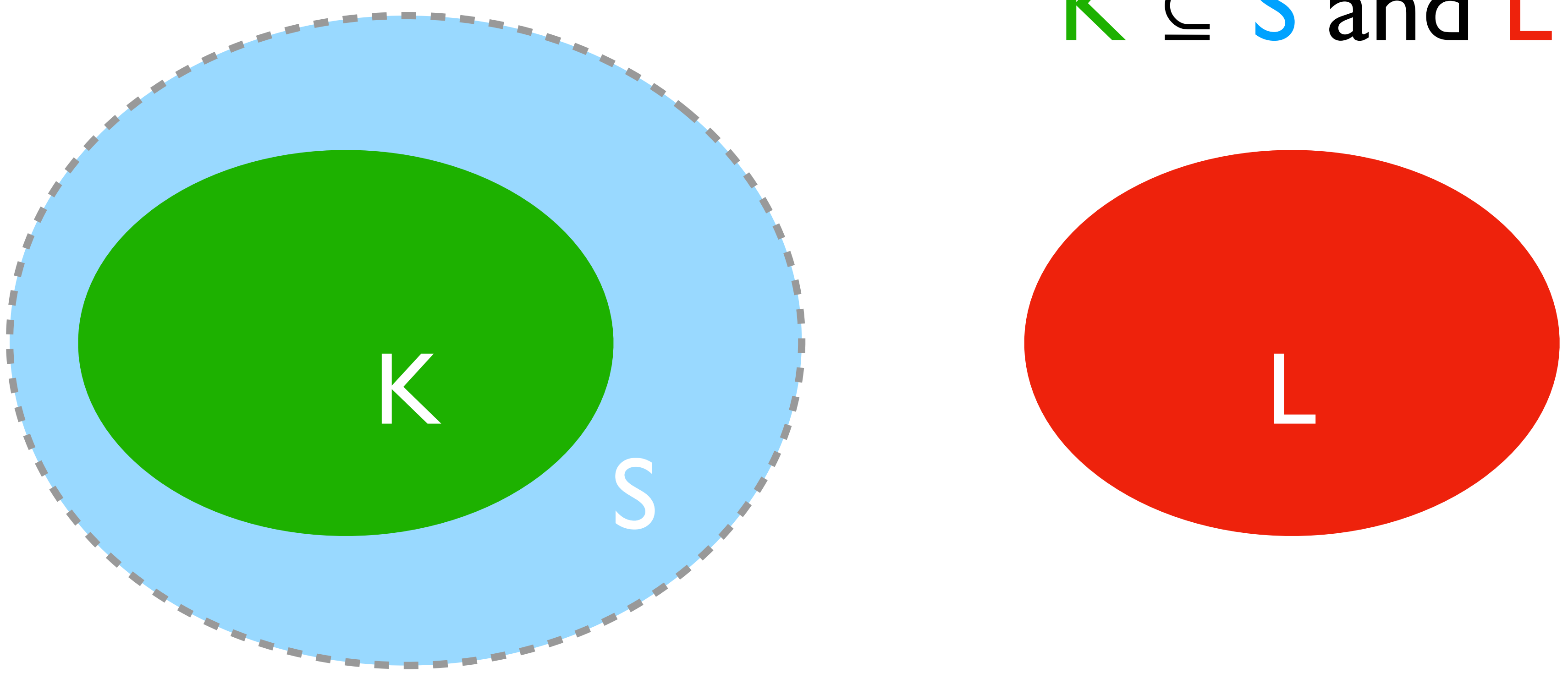


Separability

Languages K and L are regular-separable
if there is a regular S such that
$K \subseteq S$ and $L \cap S = \emptyset$.

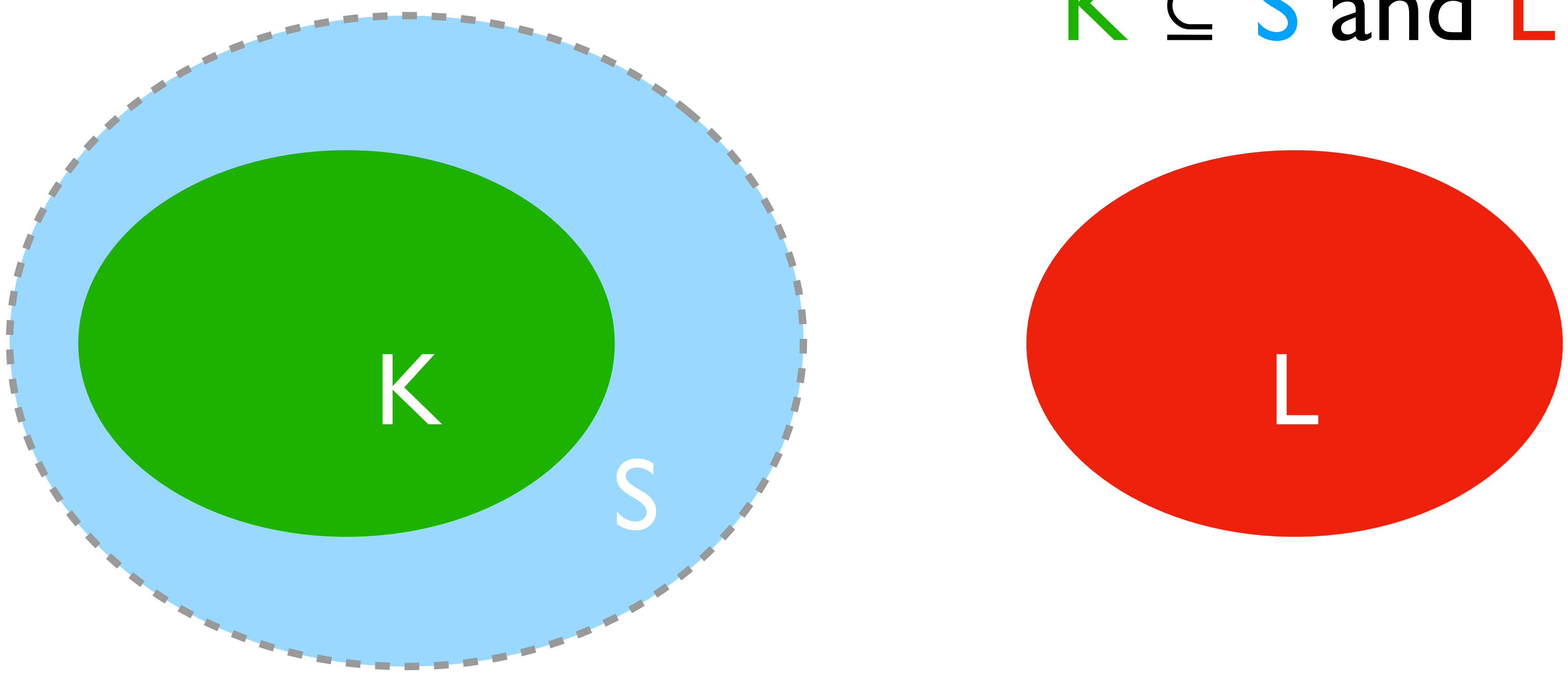
Separability

Languages K and L are regular-separable
if there is a regular S such that
$K \subseteq S$ and $L \cap S = \varnothing$.



Separability

Languages K and L are regular-separable
if there is a regular S such that
$K \subseteq S$ and $L \cap S = \varnothing$.



Theorem [Cz., Lasota, Meyer, Muskalla, Kumar, Saivasan]
Each two disjoint VASS languages are regular-separable.

Separability

Separability

Theorem For each VASS V there is a finite family of regular languages $\mathcal{F}_V$ such that if $L(c_1)$ and $L(c_2)$ are disjointed then they are separable by some language from $\mathcal{F}_V$.

Separability

Theorem For each VASS V there is
a finite family of regular languages $\mathcal{F}_V$
such that if $L(c_1)$ and $L(c_2)$ are disjointed then
they are separable by some language from $\mathcal{F}_V$.

Claim 3: For each unambiguous VASS V and h: $\Sigma^* \to M$
recognising all the languages from $\mathcal{F}_V$
the extended VASS V'_h is deterministic and computable.

Separability

Theorem For each VASS V there is
a finite family of regular languages $\mathcal{F}_V$
such that if $L(c_1)$ and $L(c_2)$ are disjointed then
they are separable by some language from $\mathcal{F}_V$.

Claim 3: For each unambiguous VASS V and h: $\Sigma^* \to M$
recognising all the languages from $\mathcal{F}_V$
the extended VASS V'_h is deterministic and computable.

So language equivalence is decidable for UVASSes

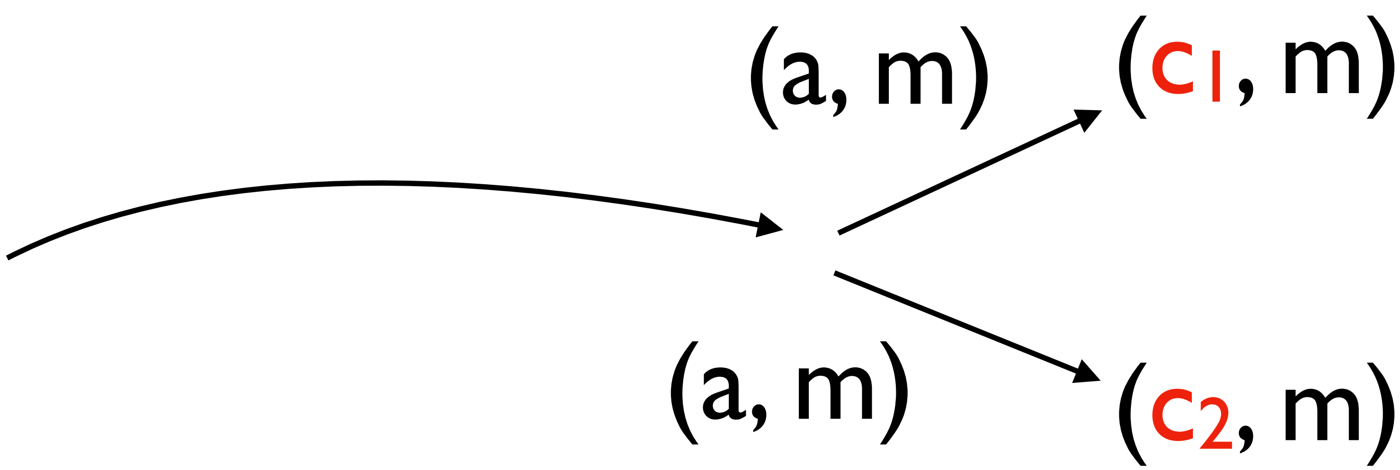
Determinism

Determinism

Claim 3: For each unambiguous VASS V and $h: \Sigma^* \to M$ recognising all the languages from $\mathcal{F}_V$ the extended VASS V'_h is deterministic and computable.

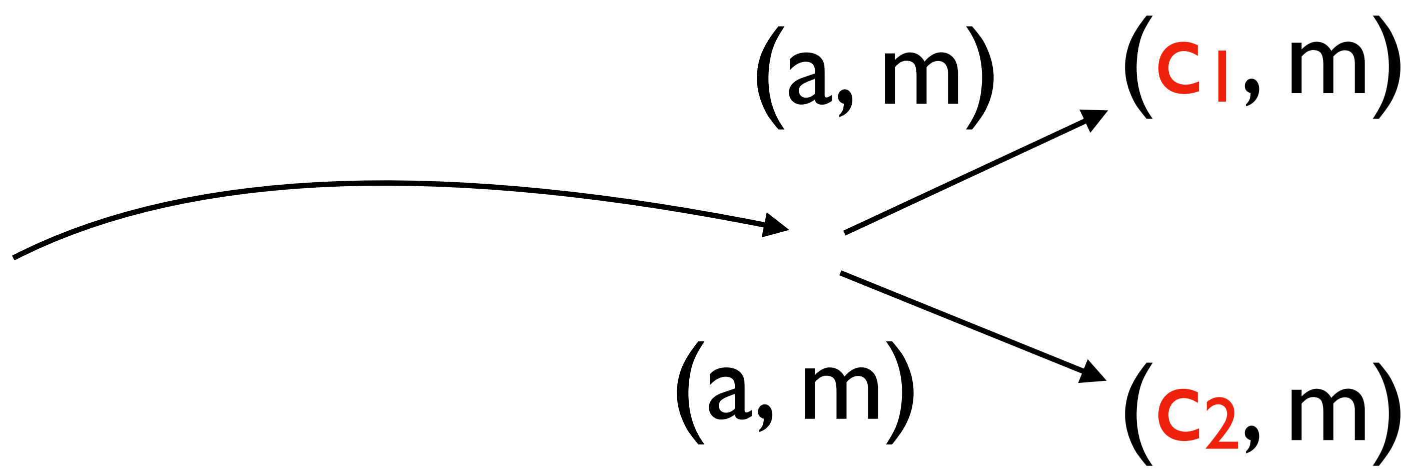
Determinism

Claim 3: For each unambiguous VASS V and $h: \Sigma^* \to M$ recognising all the languages from $\mathcal{F}_V$ the extended VASS V'_h is deterministic and computable.



Determinism

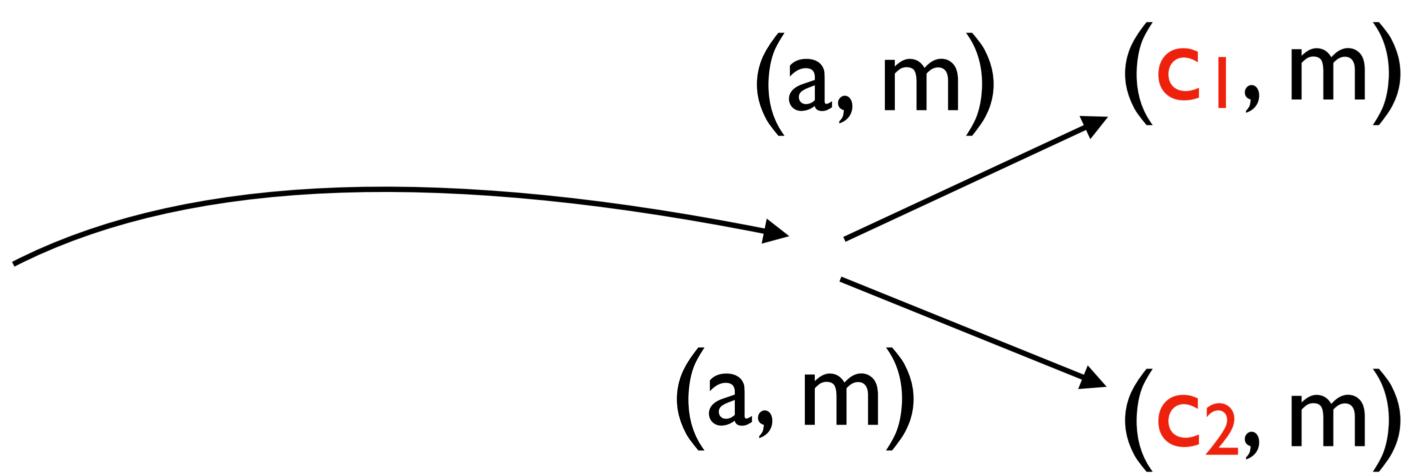
Claim 3: For each unambiguous VASS V and $h: \Sigma^* \to M$ recognising all the languages from $\mathcal{F}_V$ the extended VASS V'_h is deterministic and computable.



if $L(c_1)$ and $L(c_2)$ both nonempty then they intersect

Determinism

Claim 3: For each unambiguous VASS V and $h: \Sigma^* \to M$ recognising all the languages from $\mathcal{F}_V$ the extended VASS V'_h is deterministic and computable.

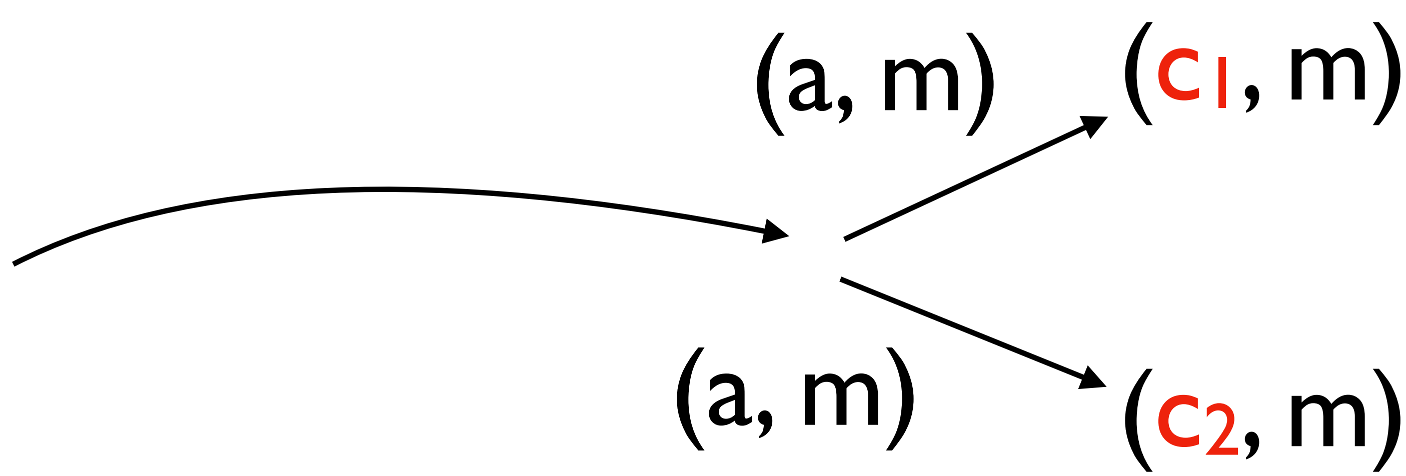


if $L(c_1)$ and $L(c_2)$ both nonempty then they intersect

this contradicts unambiguity!

Determinism

Claim 3: For each unambiguous VASS V and $h: \Sigma^* \to M$ recognising all the languages from $\mathcal{F}_V$ the extended VASS V'_h is deterministic and computable.



if $L(c_1)$ and $L(c_2)$ both nonempty then they intersect

this contradicts unambiguity!

so V_h is deterministic after removing c with $L(c) = \varnothing$

Multiplicity equivalence

Multiplicity equivalence

finite automata: in PTime

Multiplicity equivalence

finite automata: in PTime

context-free grammars: decidability is a big open problem

Multiplicity equivalence

finite automata: in PTime

context-free grammars: decidability is a big open problem

maybe a better semantics?

Multiplicity equivalence

finite automata: in PTime

context-free grammars: decidability is a big open problem

maybe a better semantics?

$\mathbb{Z}$-VASSes, one letter: decidable, holonomic sequences
(Bostan et al., ICALP `2020)

Multiplicity equivalence

finite automata: in PTime

context-free grammars: decidability is a big open problem

maybe a better semantics?

$\mathbb{Z}$-VASSes, one letter: decidable, holonomic sequences (Bostan et al., ICALP `2020)

one $\mathbb{Z}$-counter: connections to complex analysis

Future work

Future work

What makes unambiguous system easier?

Future work

What makes unambiguous system easier?

Is the lookahead trick more universal?

Future work

What makes unambiguous system easier?

Is the lookahead trick more universal?

Connections between unambiguity and separability?

Future work

What makes unambiguous system easier?

Is the lookahead trick more universal?

Connections between unambiguity and separability?

Exploring multiplicity equivalence

Future work

What makes unambiguous system easier?

Is the lookahead trick more universal?

Connections between unambiguity and separability?

Exploring multiplicity equivalence

Thank you!