

Automata and program analysis

Thomas Colcombet

FCT

Bordeaux 13 September 2017

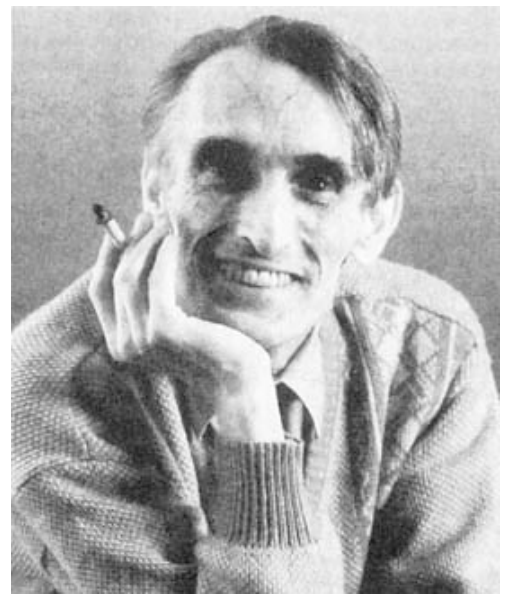
based on joint work with **Laure Daviaud** et **Florian Zuleger**



Weighted automata and tropical automata

Weighted automata

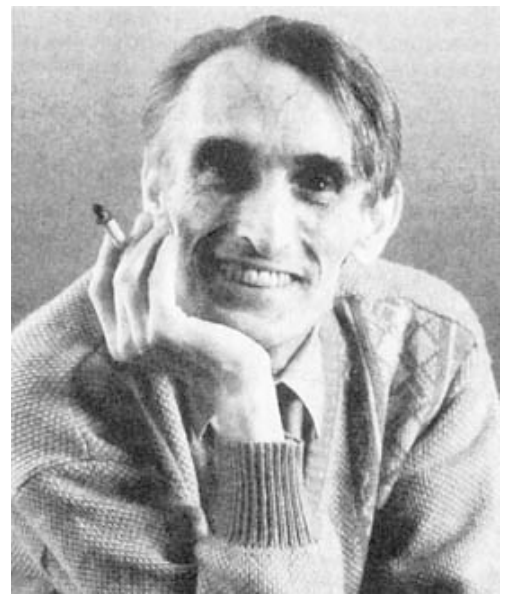
[Schützenberger 61]



Weighted automata

[Schützenberger 61]

Consider a non-deterministic automaton (A, Q, I, F, Δ) .

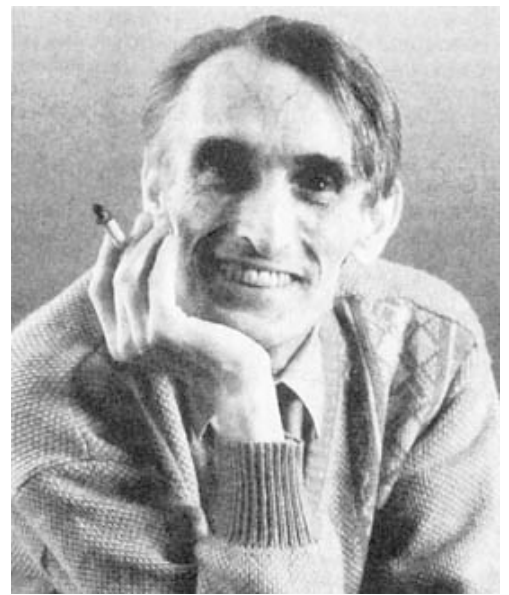


Weighted automata

[Schützenberger 61]

Consider a **non-deterministic automaton** (A, Q, I, F, Δ) .

It computes a language $L: A^* \rightarrow \{0, 1\}$

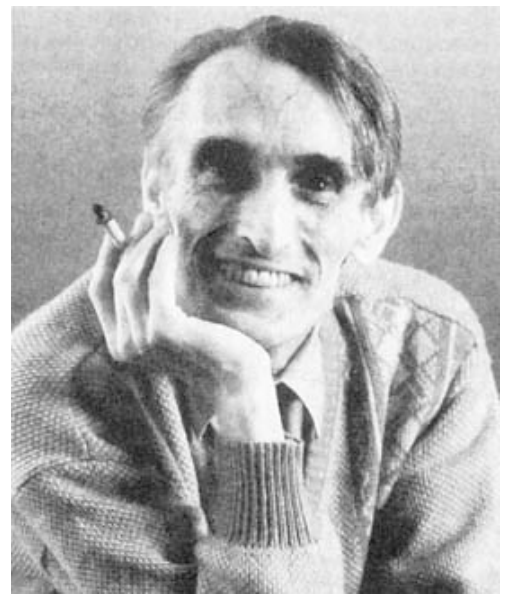


Weighted automata

[Schützenberger 61]

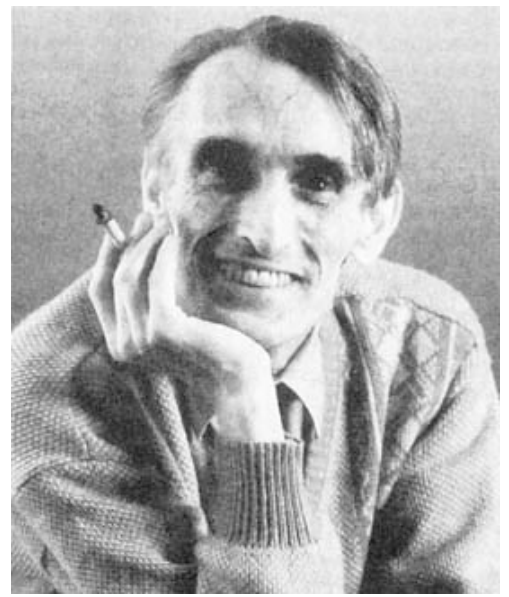
Consider a **non-deterministic automaton** (A, Q, I, F, Δ) .

It computes a language $L: A^* \rightarrow \{0, 1\}$ accepted
not accepted



Weighted automata

[Schützenberger 61]



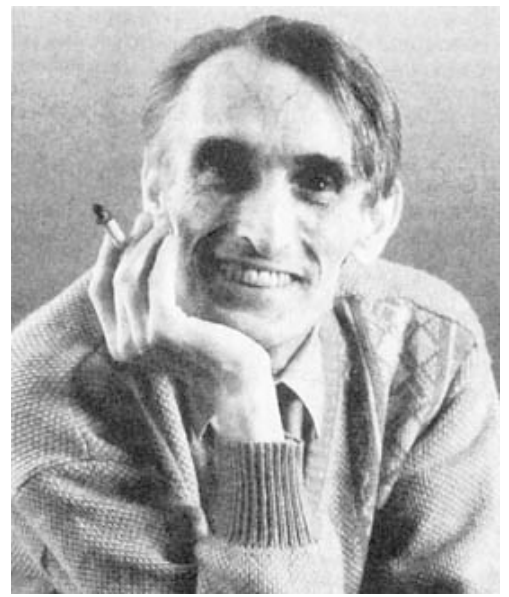
Consider a **non-deterministic automaton** (A, Q, I, F, Δ) .

It computes a language $L: A^* \rightarrow \{0,1\}$ accepted
not accepted

Q states, **initial** $I: Q \rightarrow \{0,1\}$, **final** $F: Q \rightarrow \{0,1\}$, **weights** $\Delta: Q \times A \times Q \rightarrow \{0,1\}$

Weighted automata

[Schützenberger 61]



Consider a **non-deterministic automaton** (A, Q, I, F, Δ) .

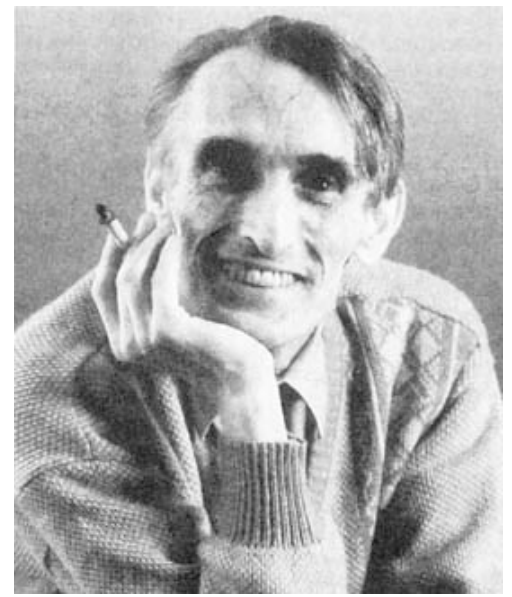
It computes a language $L: A^* \rightarrow \{0,1\}$ accepted
not accepted

Q states, **initial** $I: Q \rightarrow \{0,1\}$, **final** $F: Q \rightarrow \{0,1\}$, **weights** $\Delta: Q \times A \times Q \rightarrow \{0,1\}$

Definition: $u = a_1, a_2, \dots, a_n \in L$ iff there exists an accepting run over it.

Weighted automata

[Schützenberger 61]



Consider a **non-deterministic automaton** (A, Q, I, F, Δ) .

It computes a language $L: A^* \rightarrow \{0,1\}$ accepted
not accepted

Q states, **initial** $I: Q \rightarrow \{0,1\}$, **final** $F: Q \rightarrow \{0,1\}$, **weights** $\Delta: Q \times A \times Q \rightarrow \{0,1\}$

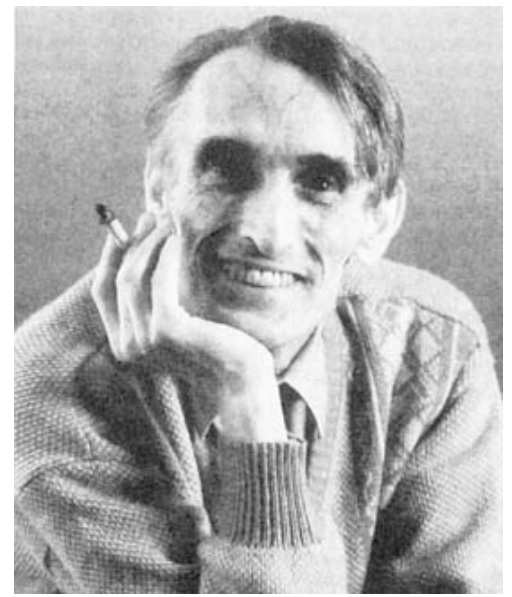
Definition: $u = a_1, a_2, \dots, a_n \in L$ iff there exists an accepting run over it.

Logically, there exist p_0, p_1, \dots, p_n such that

$$I(q_0) \wedge \Delta(q_0, a_1, q_1) \wedge \Delta(q_1, a_2, q_2) \wedge \dots \wedge \Delta(q_{n-1}, a_n, q_n) \wedge F(q_n)$$

Weighted automata

[Schützenberger 61]



Consider a **non-deterministic automaton** (A, Q, I, F, Δ) .

It computes a language $L: A^* \rightarrow \{0, 1\}$ accepted
not accepted

Q states, **initial** $I: Q \rightarrow \{0, 1\}$, **final** $F: Q \rightarrow \{0, 1\}$, **weights** $\Delta: Q \times A \times Q \rightarrow \{0, 1\}$

Definition: $u = a_1, a_2, \dots, a_n \in L$ iff there exists an accepting run over it.

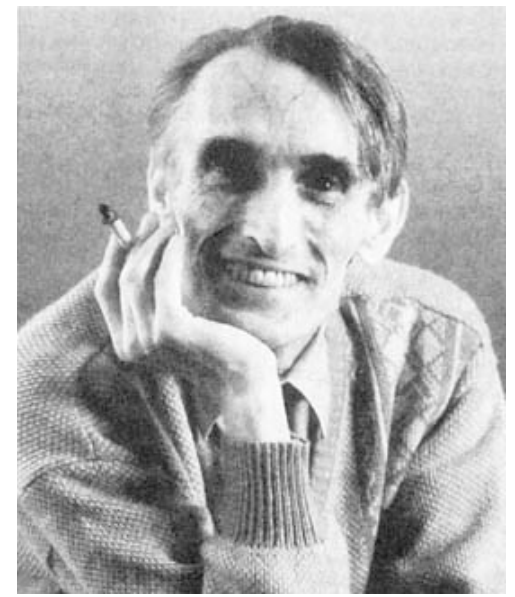
Logically, there exist p_0, p_1, \dots, p_n such that

$$I(q_0) \wedge \Delta(q_0, a_1, q_1) \wedge \Delta(q_1, a_2, q_2) \wedge \dots \wedge \Delta(q_{n-1}, a_n, q_n) \wedge F(q_n)$$

[Schützenberger 61] disjunction and conjunction can be replaced by the operation over an arbitrary semiring $(S, \oplus, \otimes, 0, 1)$.

Weighted automata

[Schützenberger 61]



Consider a **non-deterministic automaton** (A, Q, I, F, Δ) .

It computes a language $L: A^* \rightarrow \{0, 1\}$ accepted
not accepted

Q states, **initial** $I: Q \rightarrow \{0, 1\}$, **final** $F: Q \rightarrow \{0, 1\}$, **weights** $\Delta: Q \times A \times Q \rightarrow \{0, 1\}$

Definition: $u = a_1, a_2, \dots, a_n \in L$ iff there exists an accepting run over it.

Logically, there exist p_0, p_1, \dots, p_n such that

$$I(q_0) \wedge \Delta(q_0, a_1, q_1) \wedge \Delta(q_1, a_2, q_2) \wedge \dots \wedge \Delta(q_{n-1}, a_n, q_n) \wedge F(q_n)$$

[Schützenberger 61] disjunction and conjunction can be replaced by the operation over an arbitrary semiring $(S, \oplus, \otimes, 0, 1)$.

An **automaton** (A, Q, I, F, Δ) with $I: Q \rightarrow S$, $F: Q \rightarrow S$, and $\Delta: Q \times A \times Q$, computes a map $L: A^* \rightarrow S$ defined as

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} I(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

Example of weighted automata

Example of weighted automata

A **semiring** $(S, \oplus, \otimes, 0, 1)$ is such that:

- (R, \oplus) is a **commutative monoid with identity element 0**:
 $(a \oplus b) \oplus c = a \oplus (b \oplus c) ; 0 \oplus a = a \oplus 0 = a ; a \oplus b = b \oplus a$
- (R, \otimes) is a **monoid with identity element 1**:
 $(a \otimes b) \otimes c = a \otimes (b \otimes c) ; 1 \otimes a = a \otimes 1 = a$
- Multiplication **left and right distributes** over addition:
 $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) ; (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
- Multiplication by **0 annihilates S**:
 $0 \otimes a = a \otimes 0 = 0$

Example of weighted automata

A **semiring** $(S, \oplus, \otimes, 0, 1)$ is such that:

- (R, \oplus) is a **commutative monoid with identity element 0**:

$$(a \oplus b) \oplus c = a \oplus (b \oplus c) ; 0 \oplus a = a \oplus 0 = a ; a \oplus b = b \oplus a$$

- (R, \otimes) is a **monoid with identity element 1**:

$$(a \otimes b) \cdot c = a \otimes (b \otimes c) ; 1 \otimes a = a \otimes 1 = a$$

- Multiplication **left and right distributes** over addition:

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) ; (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$$

- Multiplication by **0 annihilates S**:

$$0 \otimes a = a \otimes 0 = 0$$

multiplication
addition

Example of weighted automata

A **semiring** $(S, \oplus, \otimes, 0, 1)$ is such that:

- (R, \oplus) is a **commutative monoid with identity element 0**:

$$(a \oplus b) \oplus c = a \oplus (b \oplus c) ; 0 \oplus a = a \oplus 0 = a ; a \oplus b = b \oplus a$$

- (R, \otimes) is a **monoid with identity element 1**:

$$(a \otimes b) \cdot c = a \otimes (b \otimes c) ; 1 \otimes a = a \otimes 1 = a$$

- Multiplication **left and right distributes** over addition:

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) ; (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$$

- Multiplication by **0 annihilates S**:

$$0 \otimes a = a \otimes 0 = 0$$

multiplication
addition

Gives rise to
product of S
valued matrices
that form a
monoid.

Example of weighted automata

A **semiring** $(S, \oplus, \otimes, 0, 1)$ is such that:

- (R, \oplus) is a **commutative monoid with identity element 0**:
 $(a \oplus b) \oplus c = a \oplus (b \oplus c) ; 0 \oplus a = a \oplus 0 = a ; a \oplus b = b \oplus a$
- (R, \otimes) is a **monoid with identity element 1**:
 $(a \otimes b) \cdot c = a \otimes (b \otimes c) ; 1 \otimes a = a \otimes 1 = a$
- Multiplication **left and right distributes** over addition:
 $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) ; (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
- Multiplication by **0 annihilates S**:
 $0 \otimes a = a \otimes 0 = 0$

multiplication
addition

Gives rise to
product of S
valued matrices
that form a
monoid.

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} I(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

Example of weighted automata

A **semiring** $(S, \oplus, \otimes, 0, 1)$ is such that:

- (R, \oplus) is a **commutative monoid with identity element 0**:
 $(a \oplus b) \oplus c = a \oplus (b \oplus c) ; 0 \oplus a = a \oplus 0 = a ; a \oplus b = b \oplus a$
- (R, \otimes) is a **monoid with identity element 1**:
 $(a \otimes b) \cdot c = a \otimes (b \otimes c) ; 1 \otimes a = a \otimes 1 = a$
- Multiplication **left and right distributes** over addition:
 $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) ; (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
- Multiplication by **0 annihilates S**:
 $0 \otimes a = a \otimes 0 = 0$

multiplication
addition

Gives rise to
product of S
valued matrices
that form a
monoid.

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} I(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

Boolean semiring: $(\{0, 1\}, \vee, \wedge, 0, 1)$

Example of weighted automata

A **semiring** $(S, \oplus, \otimes, 0, 1)$ is such that:

- (R, \oplus) is a **commutative monoid with identity element 0**:
 $(a \oplus b) \oplus c = a \oplus (b \oplus c) ; 0 \oplus a = a \oplus 0 = a ; a \oplus b = b \oplus a$
- (R, \otimes) is a **monoid with identity element 1**:
 $(a \otimes b) \cdot c = a \otimes (b \otimes c) ; 1 \otimes a = a \otimes 1 = a$
- Multiplication **left and right distributes** over addition:
 $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) ; (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
- Multiplication by **0 annihilates S**:
 $0 \otimes a = a \otimes 0 = 0$

multiplication
addition

Gives rise to
product of S
valued matrices
that form a
monoid.

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} I(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

Boolean semiring: $(\{0, 1\}, \vee, \wedge, 0, 1)$

Non-deterministic automata

Example of weighted automata

- A **semiring** $(S, \oplus, \otimes, 0, 1)$ is such that:
- (R, \oplus) is a **commutative monoid with identity element 0**:
 $(a \oplus b) \oplus c = a \oplus (b \oplus c) ; 0 \oplus a = a \oplus 0 = a ; a \oplus b = b \oplus a$
 - (R, \otimes) is a **monoid with identity element 1**:
 $(a \otimes b) \cdot c = a \otimes (b \otimes c) ; 1 \otimes a = a \otimes 1 = a$
 - Multiplication **left and right distributes** over addition:
 $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) ; (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
 - Multiplication by **0 annihilates S**:
 $0 \otimes a = a \otimes 0 = 0$
- multiplication
addition
- Gives rise to product of S valued matrices that form a monoid.

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} I(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

Boolean semiring: $(\{0, 1\}, \vee, \wedge, 0, 1)$

Non-deterministic automata

Reals/Integers/Rationals/Natural numbers: $(\mathbf{R}, +, \times, 0, 1)$

Example of weighted automata

- A **semiring** $(S, \oplus, \otimes, 0, 1)$ is such that:
- (R, \oplus) is a **commutative monoid with identity element 0**:
 $(a \oplus b) \oplus c = a \oplus (b \oplus c) ; 0 \oplus a = a \oplus 0 = a ; a \oplus b = b \oplus a$
 - (R, \otimes) is a **monoid with identity element 1**:
 $(a \otimes b) \cdot c = a \otimes (b \otimes c) ; 1 \otimes a = a \otimes 1 = a$
 - Multiplication **left and right distributes** over addition:
 $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) ; (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
 - Multiplication by **0 annihilates S**:
 $0 \otimes a = a \otimes 0 = 0$
- multiplication
addition
- Gives rise to product of S valued matrices that form a monoid.

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} I(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

Boolean semiring: $(\{0, 1\}, \vee, \wedge, 0, 1)$

Non-deterministic automata

Reals/Integers/Rationals/Natural numbers: $(\mathbf{R}, +, \times, 0, 1)$ Computes the number of runs of the NDA

Example of weighted automata

- A **semiring** $(S, \oplus, \otimes, 0, 1)$ is such that:
- (R, \oplus) is a **commutative monoid with identity element 0**:
 $(a \oplus b) \oplus c = a \oplus (b \oplus c) ; 0 \oplus a = a \oplus 0 = a ; a \oplus b = b \oplus a$
 - (R, \otimes) is a **monoid with identity element 1**:
 $(a \otimes b) \cdot c = a \otimes (b \otimes c) ; 1 \otimes a = a \otimes 1 = a$
 - Multiplication **left and right distributes** over addition:
 $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) ; (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
 - Multiplication by **0 annihilates S**:
 $0 \otimes a = a \otimes 0 = 0$
- multiplication
addition
- Gives rise to product of S valued matrices that form a monoid.

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} I(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

Boolean semiring: $(\{0, 1\}, \vee, \wedge, 0, 1)$

Non-deterministic automata

Reals/Integers/Rationals/Natural numbers: $(\mathbf{R}, +, \times, 0, 1)$ Computes the number of runs of the NDA

« Rat semiring »: $(\text{Rat}(A), \cup, \cdot, \emptyset, \{\varepsilon\})$

Example of weighted automata

- A **semiring** $(S, \oplus, \otimes, 0, 1)$ is such that:
- (R, \oplus) is a **commutative monoid with identity element 0**:
 $(a \oplus b) \oplus c = a \oplus (b \oplus c) ; 0 \oplus a = a \oplus 0 = a ; a \oplus b = b \oplus a$
 - (R, \otimes) is a **monoid with identity element 1**:
 $(a \otimes b) \cdot c = a \otimes (b \otimes c) ; 1 \otimes a = a \otimes 1 = a$
 - Multiplication **left and right distributes** over addition:
 $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) ; (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
 - Multiplication by **0 annihilates S**:
 $0 \otimes a = a \otimes 0 = 0$
- multiplication
addition
- Gives rise to product of S valued matrices that form a monoid.

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} I(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

Boolean semiring: $(\{0, 1\}, \vee, \wedge, 0, 1)$

Non-deterministic automata

Reals/Integers/Rationals/Natural numbers: $(\mathbf{R}, +, \times, 0, 1)$ Computes the number of runs of the NDA

« Rat semiring »: $(\text{Rat}(A), \cup, \cdot, \emptyset, \{\varepsilon\})$

Rational transducers

Example of weighted automata

- A **semiring** $(S, \oplus, \otimes, 0, 1)$ is such that:
- (R, \oplus) is a **commutative monoid with identity element 0**:
 $(a \oplus b) \oplus c = a \oplus (b \oplus c) ; 0 \oplus a = a \oplus 0 = a ; a \oplus b = b \oplus a$
 - (R, \otimes) is a **monoid with identity element 1**:
 $(a \otimes b) \cdot c = a \otimes (b \otimes c) ; 1 \otimes a = a \otimes 1 = a$
 - Multiplication **left and right distributes** over addition:
 $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) ; (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
 - Multiplication by **0 annihilates S**:
 $0 \otimes a = a \otimes 0 = 0$
- multiplication
addition
- Gives rise to product of S valued matrices that form a monoid.

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} I(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

Boolean semiring: $(\{0, 1\}, \vee, \wedge, 0, 1)$

Non-deterministic automata

Reals/Integers/Rationals/Natural numbers: $(\mathbf{R}, +, \times, 0, 1)$ Computes the number of runs of the NDA

« Rat semiring »: $(\text{Rat}(A), \cup, \cdot, \emptyset, \{\varepsilon\})$

Rational transducers

Tropical semiring: $(\mathbf{R} \cup \{-\infty\}, \max, +, -\infty, 0)$

$(\mathbf{R} \cup \{+\infty\}, \min, +, +\infty, 0), (\mathbf{N} \cup \{-\infty\}, \max, +, -\infty, 0), (\mathbf{N} \cup \{+\infty\}, \min, +, +\infty, 0)$

Example of weighted automata

- A **semiring** $(S, \oplus, \otimes, 0, 1)$ is such that:
- (R, \oplus) is a **commutative monoid with identity element 0**:
 $(a \oplus b) \oplus c = a \oplus (b \oplus c) ; 0 \oplus a = a \oplus 0 = a ; a \oplus b = b \oplus a$
 - (R, \otimes) is a **monoid with identity element 1**:
 $(a \otimes b) \cdot c = a \otimes (b \otimes c) ; 1 \otimes a = a \otimes 1 = a$
 - Multiplication **left and right distributes** over addition:
 $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) ; (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
 - Multiplication by **0 annihilates S**:
 $0 \otimes a = a \otimes 0 = 0$
- multiplication
addition
- Gives rise to product of S valued matrices that form a monoid.

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} I(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

Boolean semiring: $(\{0, 1\}, \vee, \wedge, 0, 1)$

Non-deterministic automata

Reals/Integers/Rationals/Natural numbers: $(\mathbf{R}, +, \times, 0, 1)$ Computes the number of runs of the NDA

« Rat semiring »: $(\text{Rat}(A), \cup, \cdot, \emptyset, \{\varepsilon\})$

Rational transducers

Tropical semiring: $(\mathbf{R} \cup \{-\infty\}, \max, +, -\infty, 0)$

Tropical automata

$(\mathbf{R} \cup \{+\infty\}, \min, +, +\infty, 0), (\mathbf{N} \cup \{-\infty\}, \max, +, -\infty, 0), (\mathbf{N} \cup \{+\infty\}, \min, +, +\infty, 0)$

Example of weighted automata

- A **semiring** $(S, \oplus, \otimes, 0, 1)$ is such that:
- (R, \oplus) is a **commutative monoid with identity element 0**:
 $(a \oplus b) \oplus c = a \oplus (b \oplus c) ; 0 \oplus a = a \oplus 0 = a ; a \oplus b = b \oplus a$
 - (R, \otimes) is a **monoid with identity element 1**:
 $(a \otimes b) \otimes c = a \otimes (b \otimes c) ; 1 \otimes a = a \otimes 1 = a$
 - Multiplication **left and right distributes** over addition:
 $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) ; (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
 - Multiplication by **0 annihilates S**:
 $0 \otimes a = a \otimes 0 = 0$
- multiplication
addition
- Gives rise to product of S valued matrices that form a monoid.

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} I(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

Boolean semiring: $(\{0, 1\}, \vee, \wedge, 0, 1)$

Non-deterministic automata

Reals/Integers/Rationals/Natural numbers: $(\mathbf{R}, +, \times, 0, 1)$ Computes the number of runs of the NDA

« Rat semiring »: $(\text{Rat}(A), \cup, \cdot, \emptyset, \{\varepsilon\})$

Rational transducers

Tropical semiring: $(\mathbf{R} \cup \{-\infty\}, \max, +, -\infty, 0)$

Tropical automata

$(\mathbf{R} \cup \{+\infty\}, \min, +, +\infty, 0), (\mathbf{N} \cup \{-\infty\}, \max, +, -\infty, 0), (\mathbf{N} \cup \{+\infty\}, \min, +, +\infty, 0)$



Tropical automata

Tropical automata

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} I(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

Tropical automata

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} I(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

$(\mathbf{N} \cup \{-\infty\}, \max, +, -\infty, 0)$

$L(u) \geq n$ if and only if $(\exists \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

Tropical automata

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} l(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

(**N**_U{ $-\infty$ }, max, +, $-\infty$, 0)

$L(u) \geq n$ if and only if $(\exists \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

(**N**_U{ ∞ }, min, +, ∞ , 0)

$L(u) \geq n$ if and only if $(\forall \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

Tropical automata

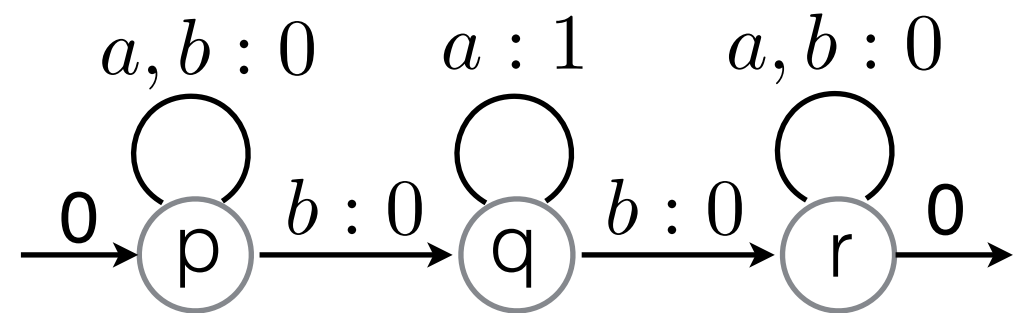
$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} l(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

(**N**_U{ $-\infty$ }, max, +, $-\infty$, 0)

$L(u) \geq n$ if and only if $(\exists \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

(**N**_U{ ∞ }, min, +, ∞ , 0)

$L(u) \geq n$ if and only if $(\forall \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$



Tropical automata

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} l(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

(**N**_U{ $-\infty$ }, max, +, $-\infty$, 0)

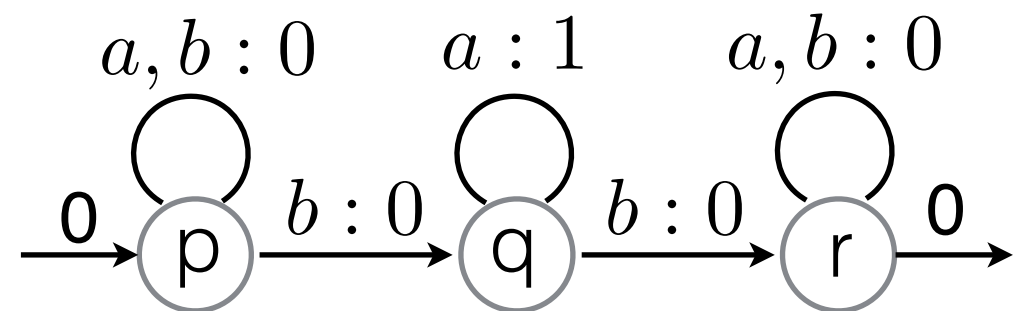
$L(u) \geq n$ if and only if $(\exists \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

(**N**_U{ ∞ }, min, +, ∞ , 0)

$L(u) \geq n$ if and only if $(\forall \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

by convention zero-transitions ($-\infty/+\infty$)
are not displayed

(neutral for \otimes and absorbing for \oplus)



Tropical automata

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} l(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

(**N**_U{ $-\infty$ }, max, +, $-\infty$, 0)

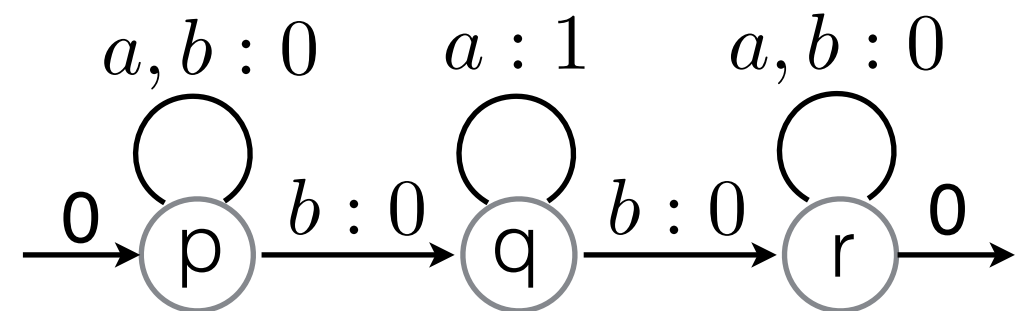
$L(u) \geq n$ if and only if $(\exists \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

(**N**_U{ ∞ }, min, +, ∞ , 0)

$L(u) \geq n$ if and only if $(\forall \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

by convention zero-transitions ($-\infty/+\infty$)
are not displayed

(neutral for \otimes and absorbing for \otimes)



The **max-plus automaton** computes:

Tropical automata

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} l(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

(**N**_U{-∞}, max, +, -∞, 0)

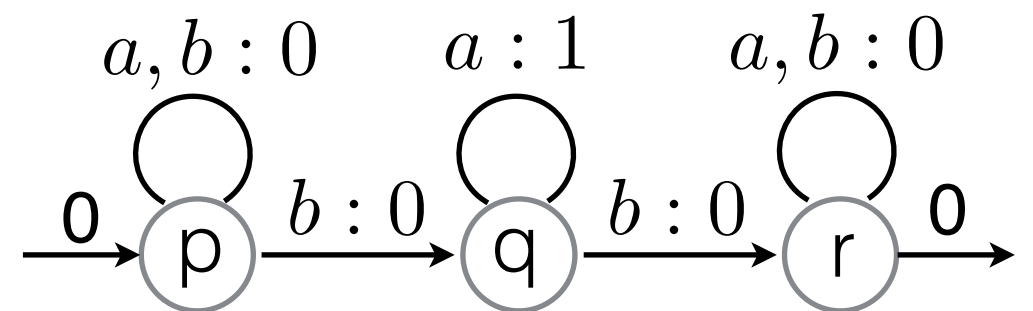
$L(u) \geq n$ if and only if $(\exists \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

(**N**_U{∞}, min, +, ∞, 0)

$L(u) \geq n$ if and only if $(\forall \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

by convention zero-transitions (-∞/+∞)
are not displayed

(neutral for \otimes and absorbing for \otimes)



The **max-plus automaton** computes:

$$L_A: A^* \rightarrow \mathbf{N}_U\{-\infty\}$$

$$u \mapsto$$

Tropical automata

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} l(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

($\mathbf{N}_{\mathbf{U}\{-\infty\}}$, max, +, $-\infty$, 0)

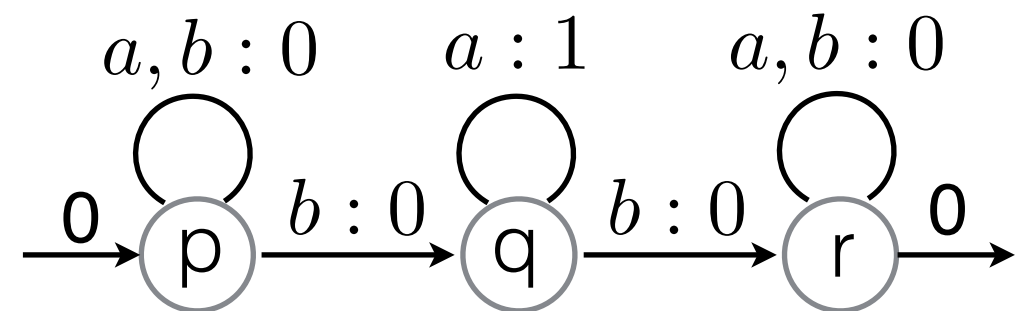
$L(u) \geq n$ if and only if $(\exists \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

($\mathbf{N}_{\mathbf{U}\{\infty\}}$, min, +, ∞ , 0)

$L(u) \geq n$ if and only if $(\forall \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

by convention zero-transitions ($-\infty/+\infty$)
are not displayed

(neutral for \otimes and absorbing for \otimes)



The **max-plus automaton** computes:

$$L_A: A^* \rightarrow \mathbf{N}_{\mathbf{U}\{-\infty\}}$$

$u \mapsto$ the size of the
longest block of
consecutive a's
surrounded by 2 b's

Tropical automata

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} l(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

($\mathbf{N}_{\mathbf{U}\{-\infty\}}$, max, +, $-\infty$, 0)

$L(u) \geq n$ if and only if $(\exists \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

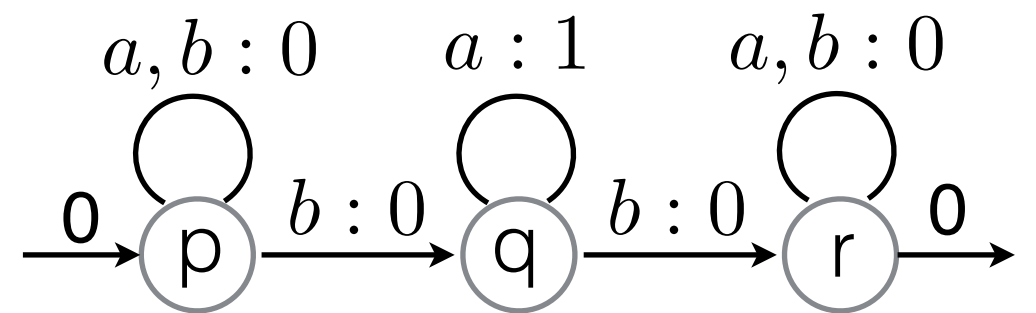
($\mathbf{N}_{\mathbf{U}\{\infty\}}$, min, +, ∞ , 0)

$L(u) \geq n$ if and only if $(\forall \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

by convention zero-transitions ($-\infty/+\infty$)
are not displayed

(neutral for \otimes and absorbing for \otimes)

[Krob 94] The equality of max-plus definable functions is undecidable.



The **max-plus automaton** computes:

$$L_A: A^* \rightarrow \mathbf{N}_{\mathbf{U}\{-\infty\}}$$

$u \mapsto$ the size of the
longest block of
consecutive a's
surrounded by 2 b's

Tropical automata

$$L(a_1 a_2 \dots a_n) = \bigoplus_{p_0, \dots, p_n} l(q_0) \otimes \left(\bigotimes_{i=1}^n \Delta(q_{i-1}, a_i, q_i) \right) \otimes F(q_n)$$

($\mathbf{N}_{\mathbf{U}\{-\infty\}}$, max, +, $-\infty$, 0)

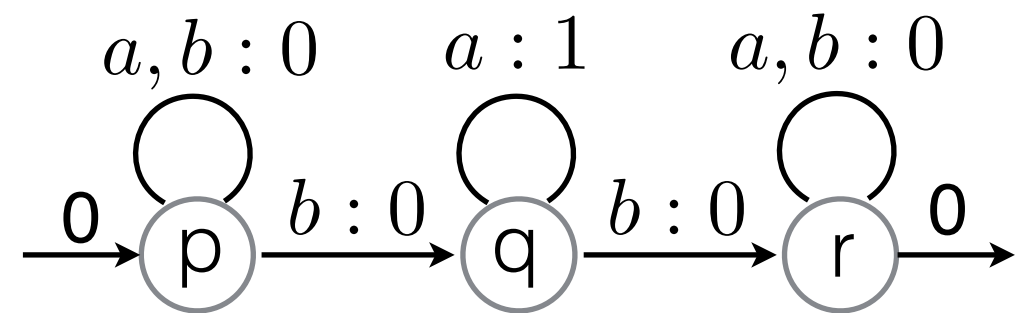
$L(u) \geq n$ if and only if $(\exists \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

($\mathbf{N}_{\mathbf{U}\{\infty\}}$, min, +, ∞ , 0)

$L(u) \geq n$ if and only if $(\forall \text{ run } \rho \text{ over } u) \text{ weight}(\rho) \geq n$

by convention zero-transitions ($-\infty/+\infty$)
are not displayed

(neutral for \otimes and absorbing for \otimes)



[Krob 94] The equality of max-plus definable functions is undecidable.

The **max-plus automaton** computes:

$$L_A: A^* \rightarrow \mathbf{N}_{\mathbf{U}\{-\infty\}}$$

$u \mapsto$ the size of the longest block of consecutive a's surrounded by 2 b's

[Hashiguchi 81] The boundedness of distance automata is decidable.

[Leung88] [Simon78,94] [Kirsten05]

[C. & Bojanczyk 06] [C. 09] [Bojanczyk15]

Alternation of quantifiers

Alternation of quantifiers

Emptiness of NDA ?

$(\exists \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

Alternation of quantifiers

Emptiness of NDA ?

$(\exists \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

NL-c

Alternation of quantifiers

Emptiness of NDA ?

NL-c

$(\exists \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

Universality of NDA ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

Alternation of quantifiers

Emptiness of NDA ?

$(\exists \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

NL-c

Universality of NDA ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

PSPACE-c
(powerset)

Alternation of quantifiers

Emptiness of NDA ?

$(\exists \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

NL-c

Universality of NDA ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

PSPACE-c
(powerset)

Is a $(\mathbb{Z}_{\cup\{\infty\}}, \max, +)$ automaton ≤ 0 ?

$(\forall \text{ word } w) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq 0$

Alternation of quantifiers

Emptiness of NDA ?

$(\exists \text{ word } w) (\exists \text{ run } \rho \text{ over } w) \quad \rho \text{ is accepting}$

NL-c

Universality of NDA ?

$(\forall \text{ word } w) (\exists \text{ run } \rho \text{ over } w) \quad \rho \text{ is accepting}$

PSPACE-c
(powerset)

Is a $(\mathbb{Z}_{\cup\{\infty\}}, \max, +)$ automaton ≤ 0 ?

$(\forall \text{ word } w) (\forall \text{ run } \rho \text{ over } w) \quad \text{weight}(\rho) \leq 0$

NL-c

Alternation of quantifiers

Emptiness of NDA ?

$(\exists \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

NL-c

Universality of NDA ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

PSPACE-c
(powerset)

Is a $(\mathbb{Z}_{\cup\{\infty\}, \max, +)$ automaton ≤ 0 ?

$(\forall \text{ word } w) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq 0$

NL-c

Is a $(\mathbb{Z}_{\cup\{\infty\}, \max, +)$ automaton ≥ 0 ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad \text{weight}(p) \geq 0$

Alternation of quantifiers

Emptiness of NDA ?

$(\exists \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

NL-c

Universality of NDA ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

PSPACE-c
(powerset)

Is a $(\mathbb{Z}_{\cup\{\infty\}}, \max, +)$ automaton ≤ 0 ?

$(\forall \text{ word } w) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq 0$

NL-c

Is a $(\mathbb{Z}_{\cup\{\infty\}}, \max, +)$ automaton ≥ 0 ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad \text{weight}(p) \geq 0$

undecidable

[Krob92, other form]

Alternation of quantifiers

Emptiness of NDA ?

$(\exists \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

NL-c

Universality of NDA ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

PSPACE-c
(powerset)

Is a $(\mathbf{Z}_{\cup\{\infty\}}, \text{max}, +)$ automaton ≤ 0 ?

$(\forall \text{ word } w) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq 0$

NL-c

Is a $(\mathbf{Z}_{\cup\{\infty\}}, \text{max}, +)$ automaton ≥ 0 ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad \text{weight}(p) \geq 0$

undecidable

[Krob92, other form]

Is a $(\mathbf{N}_{\cup\{-\infty\}}, \text{max}, +)$ automaton bounded?

$(\exists n \in \mathbf{N}) (\forall \text{ word } w) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq n$

Alternation of quantifiers

Emptiness of NDA ?

$(\exists \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

NL-c

Universality of NDA ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

PSPACE-c
(powerset)

Is a $(\mathbf{Z}_{\cup\{\infty\}}, \text{max}, +)$ automaton ≤ 0 ?

$(\forall \text{ word } w) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq 0$

NL-c

Is a $(\mathbf{Z}_{\cup\{\infty\}}, \text{max}, +)$ automaton ≥ 0 ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad \text{weight}(p) \geq 0$

undecidable
[Krob92, other form]

Is a $(\mathbf{N}_{\cup\{-\infty\}}, \text{max}, +)$ automaton bounded?

$(\exists n \in \mathbf{N}) (\forall \text{ word } w) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq n$

NL-c

Alternation of quantifiers

Emptiness of NDA ?

$(\exists \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

NL-c

Universality of NDA ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

PSPACE-c
(powerset)

Is a $(\mathbf{Z}_{\cup\{\infty\}}, \text{max}, +)$ automaton ≤ 0 ?

$(\forall \text{ word } w) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq 0$

NL-c

Is a $(\mathbf{Z}_{\cup\{\infty\}}, \text{max}, +)$ automaton ≥ 0 ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad \text{weight}(p) \geq 0$

undecidable
[Krob92, other form]

Is a $(\mathbf{N}_{\cup\{-\infty\}}, \text{max}, +)$ automaton bounded?

$(\exists n \in \mathbf{N}) (\forall \text{ word } w) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq n$

NL-c

Is a $(\mathbf{N}_{\cup\{\infty\}}, \text{min}, +)$ automaton bounded?

$(\exists n \in \mathbf{N}) (\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq n$

Alternation of quantifiers

Emptiness of NDA ?

$(\exists \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

NL-c

Universality of NDA ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

PSPACE-c
(powerset)

Is a $(\mathbf{Z}_{\cup\{\infty\}}, \text{max}, +)$ automaton ≤ 0 ?

$(\forall \text{ word } w) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq 0$

NL-c

Is a $(\mathbf{Z}_{\cup\{\infty\}}, \text{max}, +)$ automaton ≥ 0 ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad \text{weight}(p) \geq 0$

undecidable
[Krob92, other form]

Is a $(\mathbf{N}_{\cup\{-\infty\}}, \text{max}, +)$ automaton bounded?

$(\exists n \in \mathbf{N}) (\forall \text{ word } w) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq n$

NL-c

Is a $(\mathbf{N}_{\cup\{\infty\}}, \text{min}, +)$ automaton bounded?

$(\exists n \in \mathbf{N}) (\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq n$

PSPACE-c
[Hashiguchi81, Leung84]

Alternation of quantifiers

Emptiness of NDA ?

$(\exists \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

NL-c

Universality of NDA ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

PSPACE-c
(powerset)

Is a $(\mathbf{Z}_{\cup\{\infty\}}, \text{max}, +)$ automaton ≤ 0 ?

$(\forall \text{ word } w) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq 0$

NL-c

Is a $(\mathbf{Z}_{\cup\{\infty\}}, \text{max}, +)$ automaton ≥ 0 ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad \text{weight}(p) \geq 0$

undecidable
[Krob92, other form]

Is a $(\mathbf{N}_{\cup\{-\infty\}}, \text{max}, +)$ automaton bounded?

$(\exists n \in \mathbf{N}) (\forall \text{ word } w) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq n$

NL-c

Is a $(\mathbf{N}_{\cup\{\infty\}}, \text{min}, +)$ automaton bounded?

$(\exists n \in \mathbf{N}) (\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq n$ [Hashiguchi81, Leung84]

PSPACE-c

Given a $(\mathbf{N}_{\cup\{\infty\}}, \text{max}, +)$ automaton, find the least $\theta \in [0, 1]$ such that

$(\exists a) (\forall s \in \mathbf{N}) (\exists \text{ word } w, |w| \geq s) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq as^\theta$

Alternation of quantifiers

Emptiness of NDA ?

$(\exists \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

NL-c

Universality of NDA ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad p \text{ is accepting}$

PSPACE-c
(powerset)

Is a $(\mathbf{Z}_{\cup\{\infty\}}, \text{max}, +)$ automaton ≤ 0 ?

$(\forall \text{ word } w) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq 0$

NL-c

Is a $(\mathbf{Z}_{\cup\{\infty\}}, \text{max}, +)$ automaton ≥ 0 ?

$(\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad \text{weight}(p) \geq 0$

undecidable

[Krob92, other form]

Is a $(\mathbf{N}_{\cup\{-\infty\}}, \text{max}, +)$ automaton bounded?

$(\exists n \in \mathbf{N}) (\forall \text{ word } w) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq n$

NL-c

Is a $(\mathbf{N}_{\cup\{\infty\}}, \text{min}, +)$ automaton bounded?

$(\exists n \in \mathbf{N}) (\forall \text{ word } w) (\exists \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq n$ [Hashiguchi81, Leung84]

PSPACE-c

Given a $(\mathbf{N}_{\cup\{\infty\}}, \text{max}, +)$ automaton, find the least $\theta \in [0, 1]$ such that

$(\exists a) (\forall s \in \mathbf{N}) (\exists \text{ word } w, |w| \geq s) (\forall \text{ run } p \text{ over } w) \quad \text{weight}(p) \leq as^\theta$

[C., Daviaud, Zuleger 14] This θ exists and is rational.

Furthermore, it can be constructed in EXPSPACE, likely to be PSPACE-complete.

More on asymptotic analysis

More on asymptotic analysis

Given a $(\mathbf{N} \cup \{\infty\}, \max, +)$ automaton, find the least $\theta \in [0, 1]$ such that

$$(\exists a) (\forall s \in \mathbf{N}) (\exists \text{ word } w, |w| \geq s) (\forall \text{ run } \rho \text{ over } w) \text{ weight}(\rho) \leq as^\theta$$

More on asymptotic analysis

Given a $(\mathbf{N} \cup \{\infty\}, \max, +)$ automaton, find the least $\theta \in [0, 1]$ such that

$$(\exists a) (\forall s \in \mathbf{N}) (\exists \text{ word } w, |w| \geq s) (\forall \text{ run } \rho \text{ over } w) \text{ weight}(\rho) \leq as^\theta$$

[C., Daviaud, Zuleger 14] This θ exists and is rational.

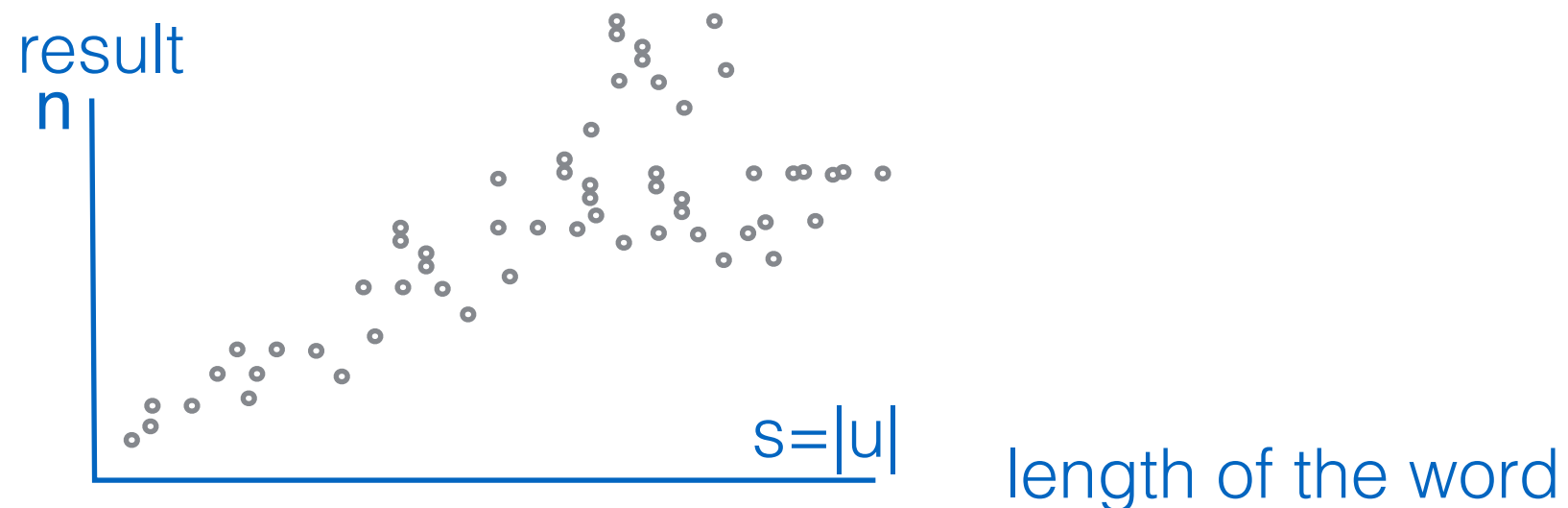
Furthermore, it can be constructed in EXPSPACE, likely to be PSPACE-complete.

More on asymptotic analysis

Given a $(\mathbf{N}_{\cup\{\infty\}}, \max, +)$ automaton, find the least $\theta \in [0, 1]$ such that
 $(\exists a) (\forall s \in \mathbf{N}) (\exists \text{ word } w, |w| \geq s) (\forall \text{ run } \rho \text{ over } w) \text{ weight}(\rho) \leq as^\theta$

[C., Daviaud, Zuleger 14] This θ exists and is rational.

Furthermore, it can be constructed in EXPSPACE, likely to be PSPACE-complete.

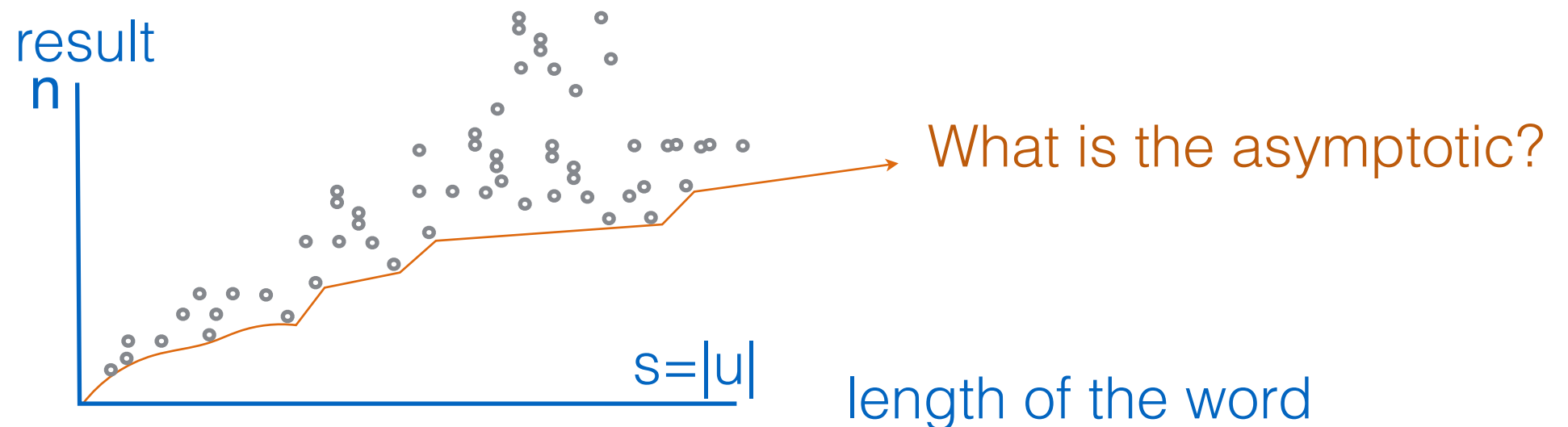


More on asymptotic analysis

Given a $(\mathbf{N}_{\cup\{\infty\}}, \max, +)$ automaton, find the least $\theta \in [0, 1]$ such that
 $(\exists a) (\forall s \in \mathbf{N}) (\exists \text{ word } w, |w| \geq s) (\forall \text{ run } \rho \text{ over } w) \text{ weight}(\rho) \leq as^\theta$

[C., Daviaud, Zuleger 14] This θ exists and is rational.

Furthermore, it can be constructed in EXPSPACE, likely to be PSPACE-complete.

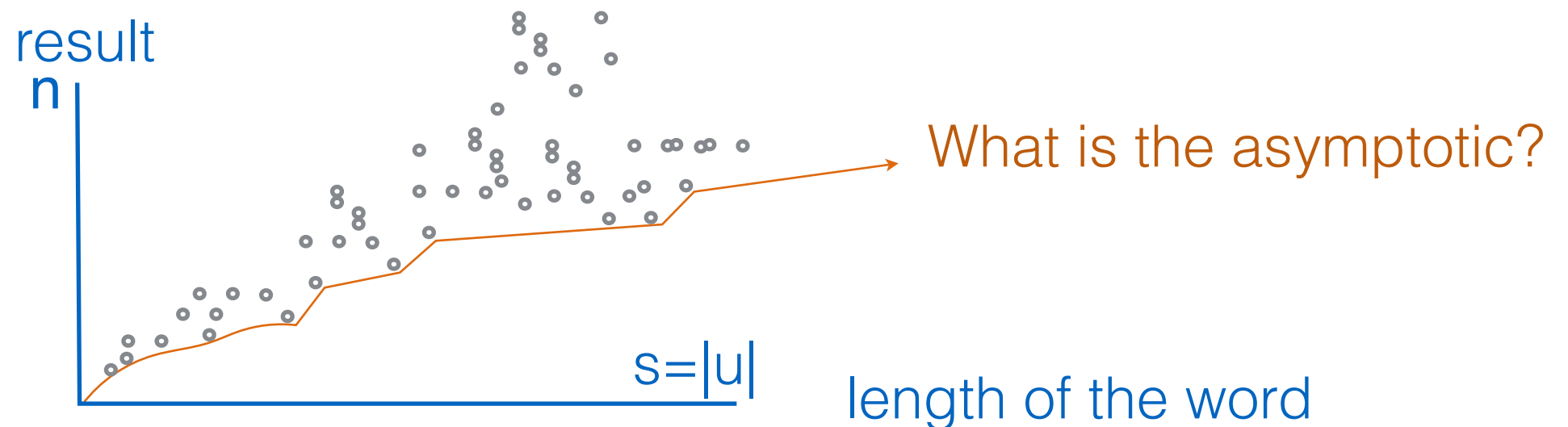


More on asymptotic analysis

Given a $(\mathbf{N} \cup \{\infty\}, \max, +)$ automaton, find the least $\theta \in [0, 1]$ such that
 $(\exists a) (\forall s \in \mathbf{N}) (\exists \text{ word } w, |w| \geq s) (\forall \text{ run } \rho \text{ over } w) \text{ weight}(\rho) \leq as^\theta$

[C., Daviaud, Zuleger 14] This θ exists and is rational.

Furthermore, it can be constructed in EXPSPACE, likely to be PSPACE-complete.



Compute:
$$\liminf_{u \in A^*} \frac{\log f(u)}{\log |u|} = \theta$$

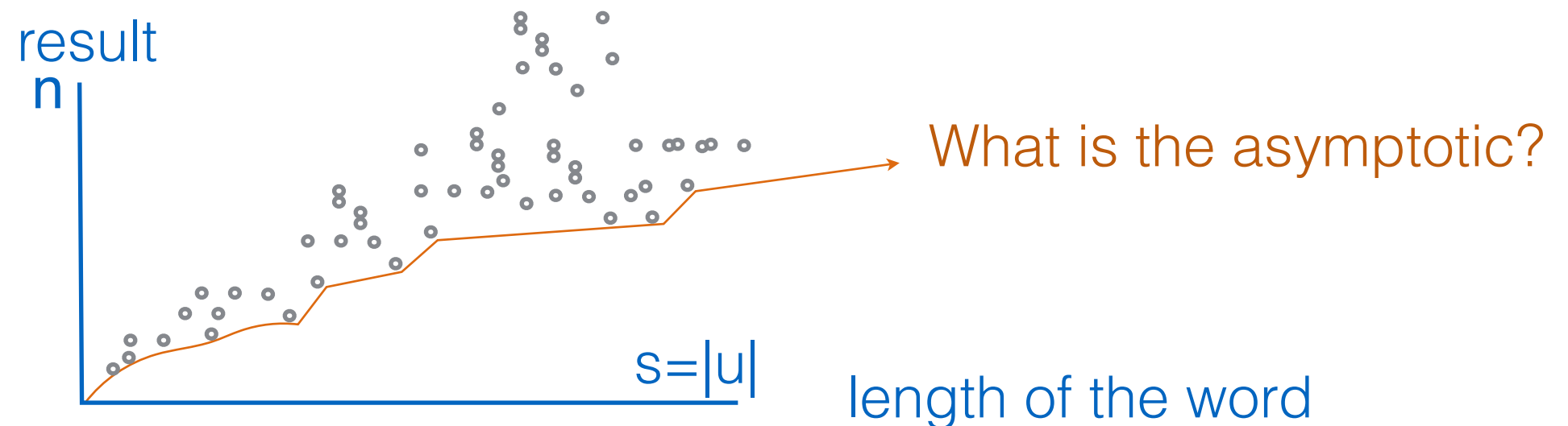
find the least value of a word
of length at least s

More on asymptotic analysis

Given a $(\mathbf{N} \cup \{\infty\}, \max, +)$ automaton, find the least $\theta \in [0, 1]$ such that
 $(\exists a) (\forall s \in \mathbf{N}) (\exists \text{ word } w, |w| \geq s) (\forall \text{ run } \rho \text{ over } w) \text{ weight}(\rho) \leq as^\theta$

[C., Daviaud, Zuleger 14] This θ exists and is rational.

Furthermore, it can be constructed in EXPSPACE, likely to be PSPACE-complete.



Compute: $\liminf_{u \in A^*} \frac{\log f(u)}{\log |u|} = \theta \iff \limsup_{u \in A^*} \frac{\log |u|}{\log f(|u|)} = \frac{1}{\theta}$

find the least value of a word of length at least s find the longest size of a word of value at most n

Ingredients of the proof

Ingredients of the proof

Ingredient 1.

Given a set of words W , collect an information $I(W)$ sufficient for understanding its behavior in any context.

Ingredients of the proof

Ingredient 1.

Given a set of words W , collect an information $I(W)$ sufficient for understanding its behavior in any context.

e.g. for universality $I(W) = \{P \subseteq Q : P = \text{Reach}(I, u) \text{ for some } u \in W\}$

Ingredients of the proof

Ingredient 1.

Given a set of words W , collect an information $I(W)$ sufficient for understanding its behavior in any context.

e.g. for universality $I(W) = \{P \subseteq Q : P = \text{Reach}(I, u) \text{ for some } u \in W\}$

In our case,

$$I(W) = \{ f: Q \times Q \rightarrow \mathbf{N} : \text{there is a run that displays this behavior} \} \subseteq P(\mathbf{N}^{Q \times Q})$$

Ingredients of the proof

Ingredient 1.

Given a set of words W , collect an information $I(W)$ sufficient for understanding its behavior in any context.

e.g. for universality $I(W) = \{P \subseteq Q : P = \text{Reach}(I, u) \text{ for some } u \in W\}$

In our case,

$$I(W) = \{ f: Q \times Q \rightarrow \mathbf{N} : \text{there is a run that displays this behavior} \} \subseteq P(\mathbf{N}^{Q \times Q})$$

Ingredient 2.

Give a notion of **approximation** for such sets: Hausdorff-like keeping asymptotes.

Ingredients of the proof

Ingredient 1.

Given a set of words W , collect an information $I(W)$ sufficient for understanding its behavior in any context.

e.g. for universality $I(W) = \{P \subseteq Q : P = \text{Reach}(I, u) \text{ for some } u \in W\}$

In our case,

$$I(W) = \{ f: Q \times Q \rightarrow \mathbf{N} : \text{there is a run that displays this behavior} \} \subseteq P(\mathbf{N}^{Q \times Q})$$

Ingredient 2.

Give a notion of **approximation** for such sets: Hausdorff-like keeping asymptotes.

Ingredient 3.

Define **presentable sets** families of such sets of maps that are nicely behaved (that can be algorithmically handled). In our case **unions of convex polytopes in $\mathbf{R}^{Q \times Q}$** representing simultaneous asymptotic behaviors.

Ingredients of the proof

Ingredient 1.

Given a set of words W , collect an information $I(W)$ sufficient for understanding its behavior in any context.

e.g. for universality $I(W) = \{P \subseteq Q : P = \text{Reach}(I, u) \text{ for some } u \in W\}$

In our case,

$$I(W) = \{ f: Q \times Q \rightarrow \mathbf{N} : \text{there is a run that displays this behavior} \} \subseteq P(\mathbf{N}^{Q \times Q})$$

Ingredient 2.

Give a notion of **approximation** for such sets: Hausdorff-like keeping asymptotes.

Ingredient 3.

Define **presentable sets** families of such sets of maps that are nicely behaved (that can be algorithmically handled). In our case **unions of convex polytopes in $\mathbf{R}^{Q \times Q}$** representing simultaneous asymptotic behaviors.

Step 4.

Compute a **presentable equivalent** (up to approximation) of $I(A^*)$

Ingredients of the proof

Ingredient 1.

Given a set of words W , collect an information $I(W)$ sufficient for understanding its behavior in any context.

e.g. for universality $I(W) = \{P \subseteq Q : P = \text{Reach}(I, u) \text{ for some } u \in W\}$

In our case,

$$I(W) = \{ f: Q \times Q \rightarrow \mathbf{N} : \text{there is a run that displays this behavior} \} \subseteq P(\mathbf{N}^{Q \times Q})$$

Ingredient 2.

Give a notion of **approximation** for such sets: Hausdorff-like keeping asymptotes.

Ingredient 3.

Define **presentable sets** families of such sets of maps that are nicely behaved (that can be algorithmically handled). In our case **unions of convex polytopes in $\mathbf{R}^{Q \times Q}$** representing simultaneous asymptotic behaviors.

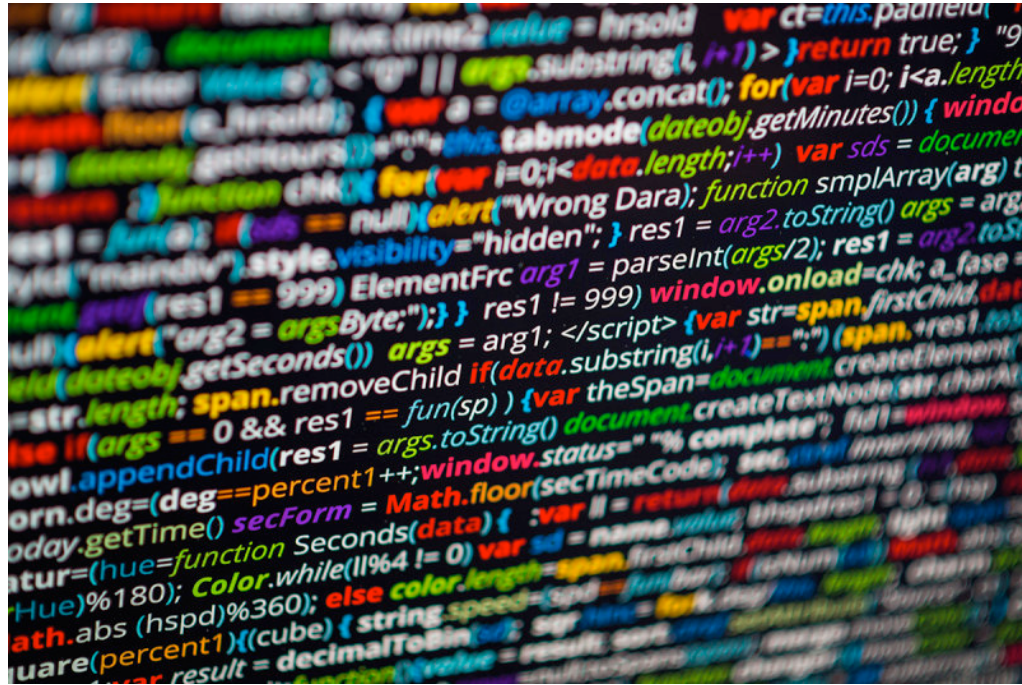
Step 4.

Compute a **presentable equivalent** (up to approximation) of $I(A^*)$

This is done by induction of the **factorisation forest height** [Simon].

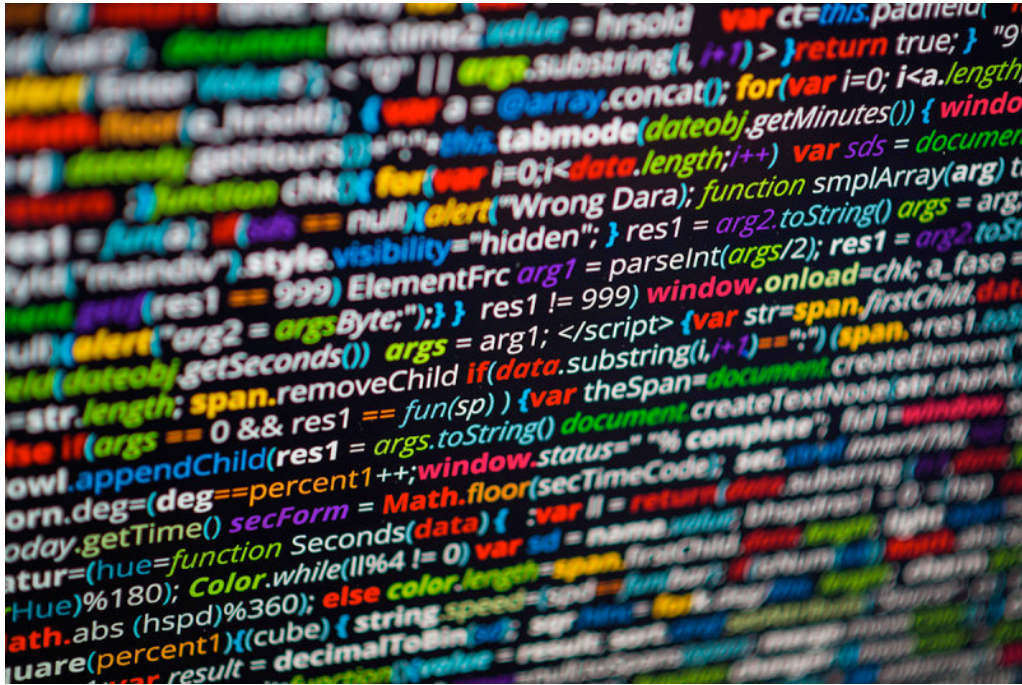
Program analysis and the size-change abstraction

Program analysis



- Given an input program/piece of program:
- Does it perform a zero division?
 - Does it access a non-allocated memory area?
 - Is there a dynamic type problem?
 - Does it comply to the specification?
 - Is there a memory leakage?
 - Does it terminate?
 - What is its running time?

Program analysis

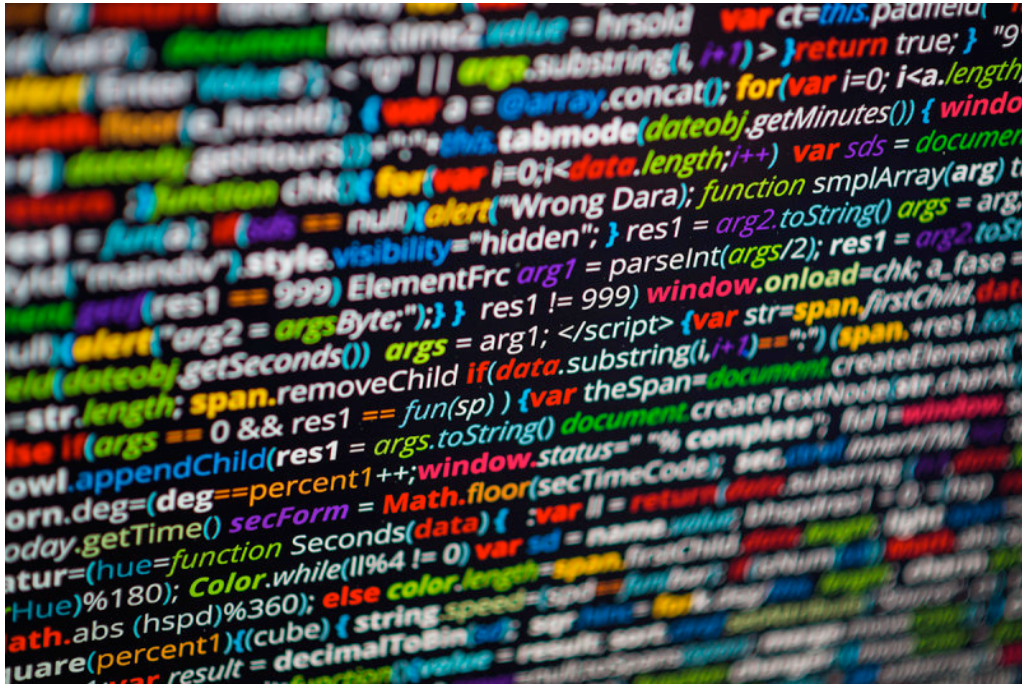


Given an input program/piece of program:

- Does it perform a zero division?
- Does it access a non-allocated memory area?
- Is there a dynamic type problem?
- Does it comply to the specification?
- Is there a memory leakage?
- Does it terminate?
- What is its running time?

[Rice-like] Essentially, all these questions are undecidable.

Program analysis



Given an input program/piece of program:

- Does it perform a zero division?
- Does it access a non-allocated memory area?
- Is there a dynamic type problem?
- Does it comply to the specification?
- Is there a memory leakage?
- Does it terminate?
- What is its running time?

[Rice-like] Essentially, all these questions are undecidable.

Solution here: in this talk, we use the size-change abstract model

(**[Ben-Amram, Chin Soon Lee, Neil D. Jones 01]**).



Example

```
void main() {  
    uint x,y;  
    x = read_input();  
    y = read_input();  
    while (x > 0) {  
        if (y > 0)  
            { y--; }  
        else  
            { y = read_input();  
              x--; }  
    }  
}
```

Example

```
void main() {  
    uint x,y;  
    x = read_input();  
    y = read_input();  
    while (x > 0) {  
        if (y > 0)  
            { y--; }  
        else  
            { y = read_input();  
              x--; }  
    }  
}
```

these variables remain non-negative.

Example

```
void main() {  
  uint x,y;  
  x = read_input();  
  y = read_input();  
  while (x > 0) {  
    if (y > 0)  
      { y--; }  
    else  
      { y = read_input();  
        x--; }  
  }  
}
```

these variables remain non-negative.

are initialized with an uncontrolled value

Example

```
void main() {  
    uint x,y;  
    x = read_input();  
    y = read_input();  
    while (x > 0) {  
        if (y > 0)  
            { y--; }  
        else  
            { y = read_input();  
              x--; }  
    }  
}
```

these variables remain non-negative.

are initialized with an uncontrolled value

either y decreases

Example

```
void main() {  
    uint x,y;  
    x = read_input();  
    y = read_input();  
    while (x > 0) {  
        if (y > 0)  
            { y--; }  
        else  
            { y = read_input();  
              x--; }  
    }  
}
```

these variables remain non-negative.

are initialized with an uncontrolled value

either y decreases

or x decreases,
and y gets an uncontrolled value

Example

```
void main() {  
  uint x,y;  
  x = read_input();  
  y = read_input();  
  while (x > 0) {  
    if (y > 0)  
      { y--; }  
    else  
      { y = read_input();  
        x--; }  
  }  
}
```

these variables remain non-negative.

are initialized with an uncontrolled value

either y decreases

or x decreases,
and y gets an uncontrolled value

Remark: This program terminates.

Example

```
void main() {  
  uint x,y;  
  x = read_input();  
  y = read_input();  
  while (x > 0) {  
    if (y > 0)  
      { y--; }  
    else  
      { y = read_input();  
        x--; }  
  }  
}
```

these variables remain non-negative.

are initialized with an uncontrolled value

either y decreases

or x decreases,
and y gets an uncontrolled value

Remark: This program terminates.

Question: what method can automatically establish it ?

Principle of abstraction

Principle of abstraction

Principle: replace the program by an abstraction:

- Information that is lost is replaced by non-determinism.
This includes:
 - + The dynamic information resulting from the interactions with the environment.
 - + All the tests and computations that cannot be abstracted in the restricted model of the abstraction.
- The resulting abstraction can be analyzed: it can be decided whether the resulting abstraction stops on all its executions.
- If the abstraction stops on all its executions, then the original program stops on all its executions.

Principle of abstraction

Principle: replace the program by an abstraction:

- Information that is lost is replaced by non-determinism.
This includes:
 - + The dynamic information resulting from the interactions with the environment.
 - + All the tests and computations that cannot be abstracted in the restricted model of the abstraction.
- The resulting abstraction can be analyzed: it can be decided whether the resulting abstraction stops on all its executions.
- If the abstraction stops on all its executions, then the original programs stops on all its executions.

Remark: Of course, this is a compromise between the efficiency of the decision problem, and the loss of information during the abstraction.

Principle of abstraction

Principle: replace the program by an abstraction:

- Information that is lost is replaced by non-determinism.
This includes:
 - + The dynamic information resulting from the interactions with the environment.
 - + All the tests and computations that cannot be abstracted in the restricted model of the abstraction.
- The resulting abstraction can be analyzed: it can be decided whether the resulting abstraction stops on all its executions.
- If the abstraction stops on all its executions, then the original programs stops on all its executions.

Remark: Of course, this is a compromise between the efficiency of the decision problem, and the loss of information during the abstraction.

⇒ In this talk, we use the model of size-change abstraction.

Size-change abstraction

Size-change abstraction

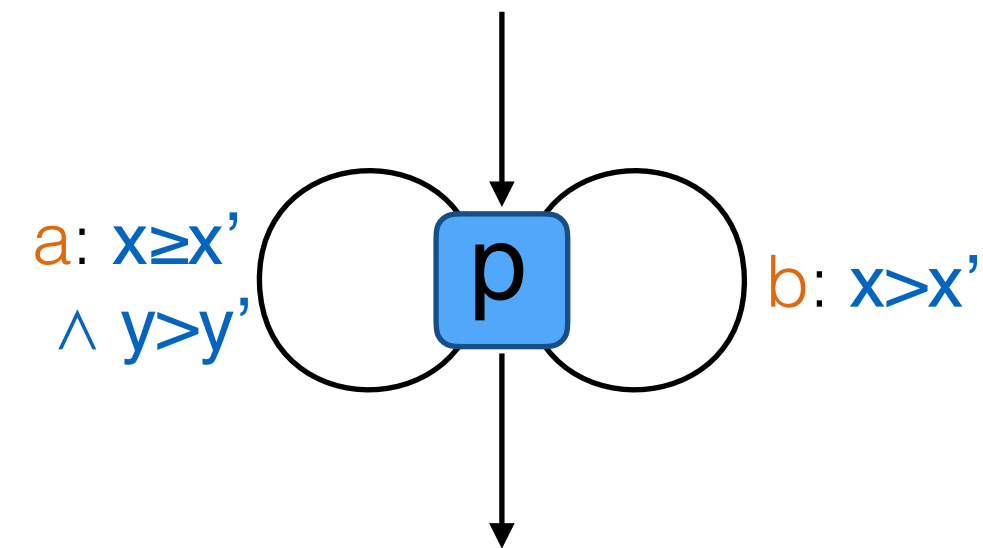
[Ben-Amram et al. 01] A **size-change abstraction (SCA)**:

- this is a non-deterministic finite state machine
- that uses a finite set **variables** (**x**, **y**, **z**...) ranging over **non-negative integers**
- during each **transition**, a guards relate the variables before and after:
 - $x \geq y$** meaning « **val of x before the transition** \geq **val of y after the transition** »
 - $x > y$** meaning « **val of x before the transition** $>$ **val of y after the transition** »

Size-change abstraction

[Ben-Amram et al. 01] A **size-change abstraction (SCA)**:

- this is a non-deterministic finite state machine
- that uses a finite set **variables** (x, y, z, \dots) ranging over **non-negative integers**
- during each **transition**, a guards relate the variables before and after:
 - $x \geq y'$ meaning « **val of x before the transition** \geq **val of y after the transition** »
 - $x > y'$ meaning « **val of x before the transition** $>$ **val of y after the transition** »

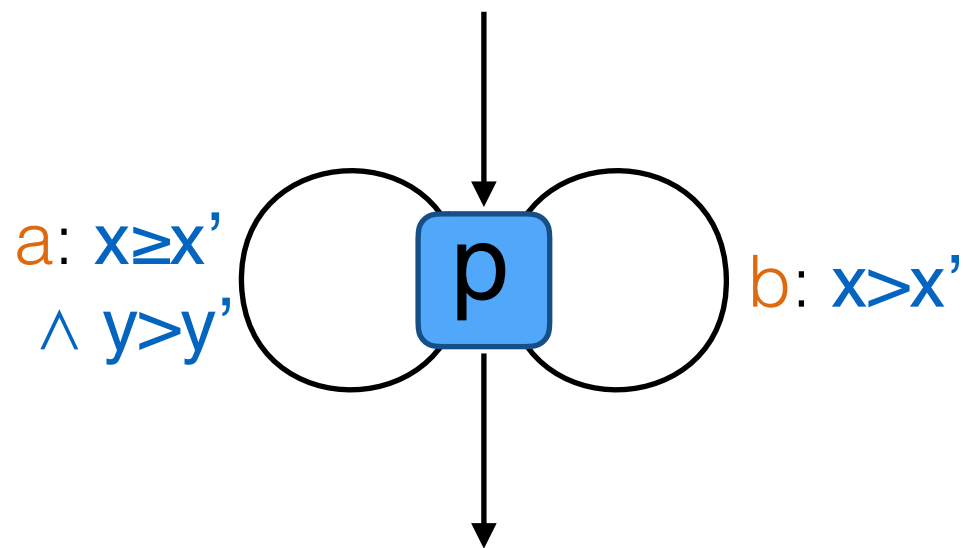


Size-change abstraction

[Ben-Amram et al. 01] A **size-change abstraction (SCA)**:

- this is a non-deterministic finite state machine
- that uses a finite set **variables** (x, y, z, \dots) ranging over **non-negative integers**
- during each **transition**, a guards relate the variables before and after:
 $x \geq y'$ meaning « **val of x before the transition** \geq **val of y after the transition** »
 $x > y'$ meaning « **val of x before the transition** $>$ **val of y after the transition** »

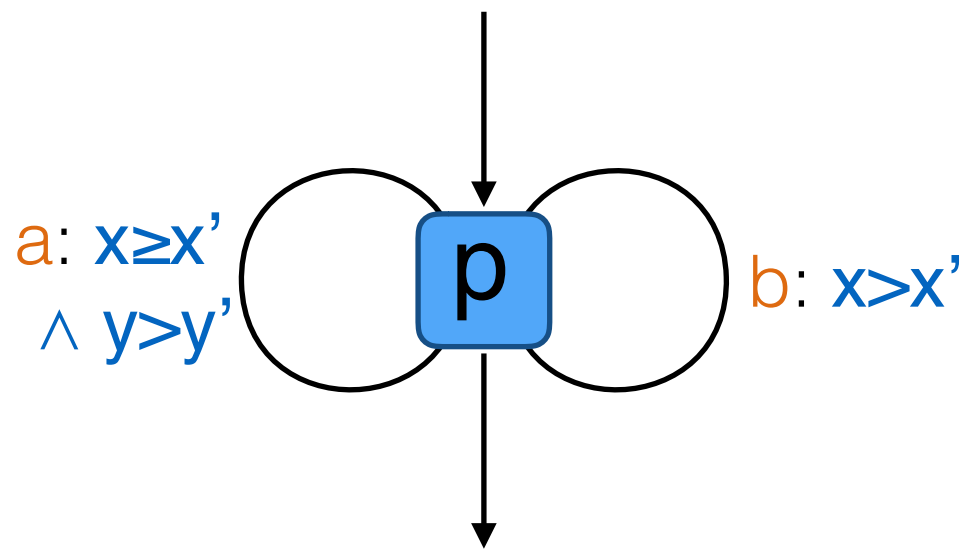
A **configuration** is a **state** together with a non-negative integer value for each of the **variables**.



Size-change abstraction

[Ben-Amram et al. 01] A **size-change abstraction (SCA)**:

- this is a non-deterministic finite state machine
- that uses a finite set **variables** (x, y, z, \dots) ranging over **non-negative integers**
- during each **transition**, a guards relate the variables before and after:
 $x \geq y'$ meaning « **val of x before the transition** \geq **val of y after the transition** »
 $x > y'$ meaning « **val of x before the transition** $>$ **val of y after the transition** »



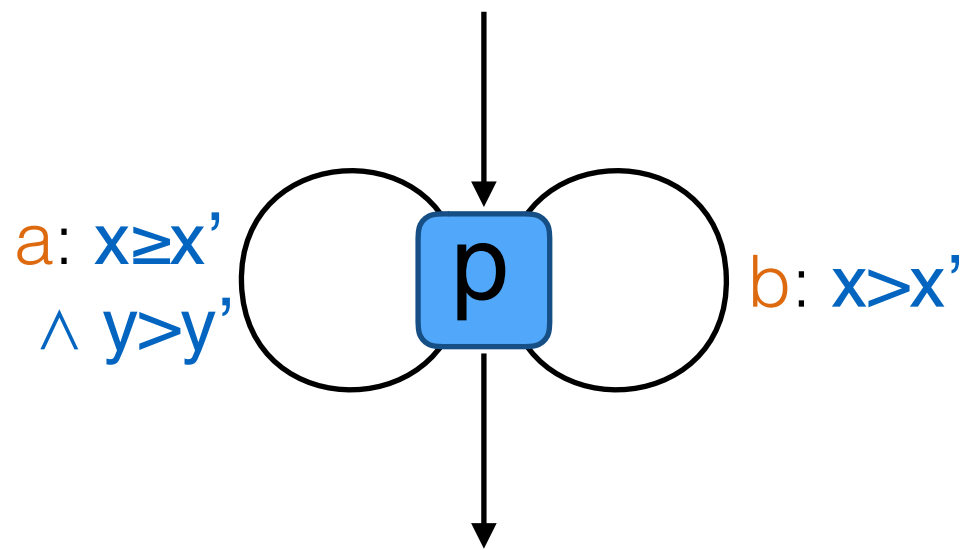
A **configuration** is a **state** together with a non-negative integer value for each of the **variables**.

A **run of the SCA** is a sequence of **configurations** that starts in an initial configuration, ends in a final one, and each consecutive configurations satisfy the guard of some possible transition.

Size-change abstraction

[Ben-Amram et al. 01] A **size-change abstraction (SCA)**:

- this is a non-deterministic finite state machine
- that uses a finite set **variables** (x, y, z, \dots) ranging over **non-negative integers**
- during each **transition**, a guards relate the variables before and after:
 $x \geq y'$ meaning « **val of x before the transition** \geq **val of y after the transition** »
 $x > y'$ meaning « **val of x before the transition** $>$ **val of y after the transition** »



A **configuration** is a **state** together with a non-negative integer value for each of the **variables**.

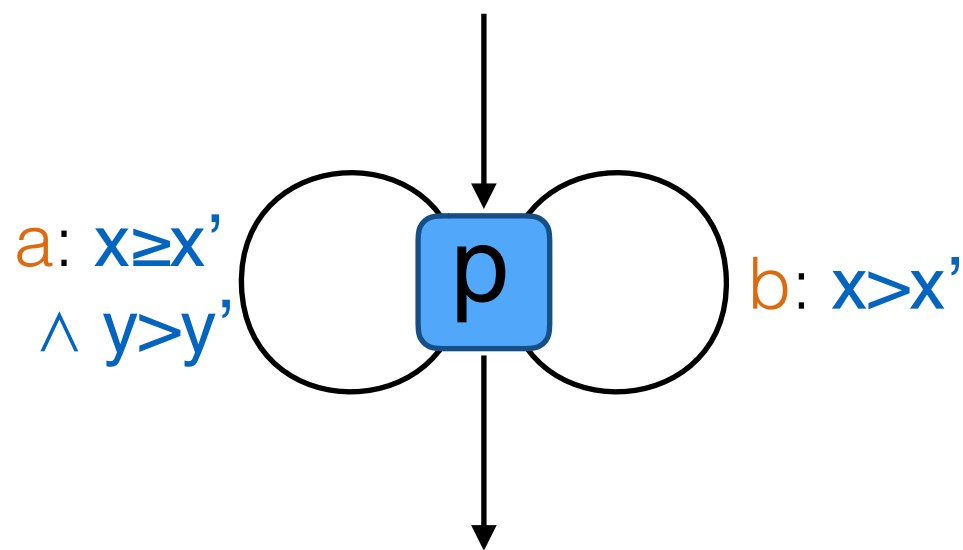
A **run of the SCA** is a sequence of **configurations** that starts in an initial configuration, ends in a final one, and each consecutive configurations satisfy the guard of some possible transition.

(p,2,2) (p,2,1) (p,2,0) (p,1,2) (p,1,1) (p,0,2) (p,0,1) (p,0,0)

Size-change abstraction

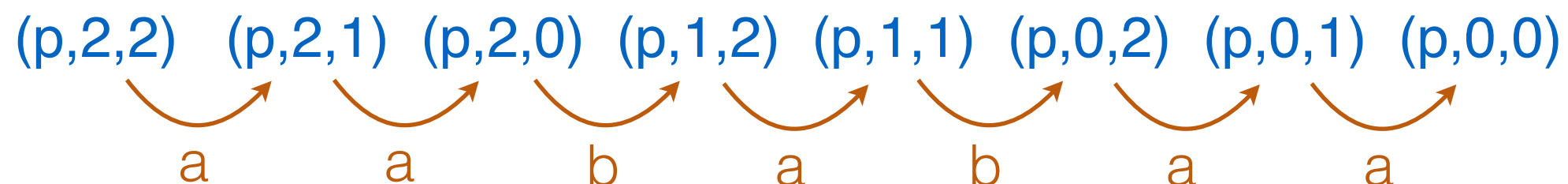
[Ben-Amram et al. 01] A **size-change abstraction (SCA)**:

- this is a non-deterministic finite state machine
- that uses a finite set **variables** (x, y, z, \dots) ranging over **non-negative integers**
- during each **transition**, a guards relate the variables before and after:
 $x \geq y'$ meaning « **val of x before the transition** \geq **val of y after the transition** »
 $x > y'$ meaning « **val of x before the transition** $>$ **val of y after the transition** »



A **configuration** is a **state** together with a non-negative integer value for each of the **variables**.

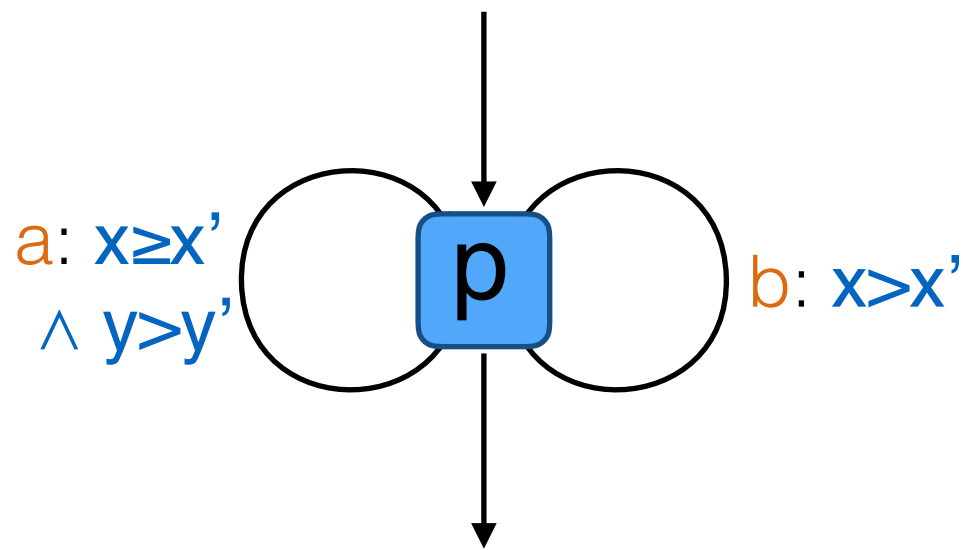
A **run of the SCA** is a sequence of **configurations** that starts in an initial configuration, ends in a final one, and each consecutive configurations satisfy the guard of some possible transition.



Size-change abstraction

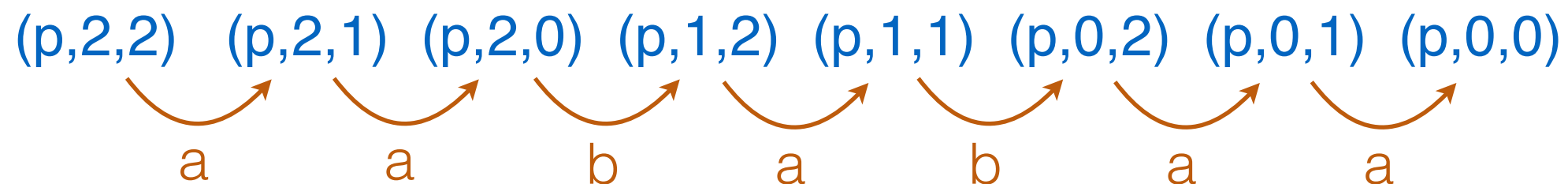
[Ben-Amram et al. 01] A **size-change abstraction (SCA)**:

- this is a non-deterministic finite state machine
- that uses a finite set **variables** (x, y, z, \dots) ranging over **non-negative integers**
- during each **transition**, a guards relate the variables before and after:
 $x \geq y'$ meaning « **val of x before the transition** \geq **val of y after the transition** »
 $x > y'$ meaning « **val of x before the transition** $>$ **val of y after the transition** »



A **configuration** is a **state** together with a non-negative integer value for each of the **variables**.

A **run of the SCA** is a sequence of **configurations** that starts in an initial configuration, ends in a final one, and each consecutive configurations satisfy the guard of some possible transition.

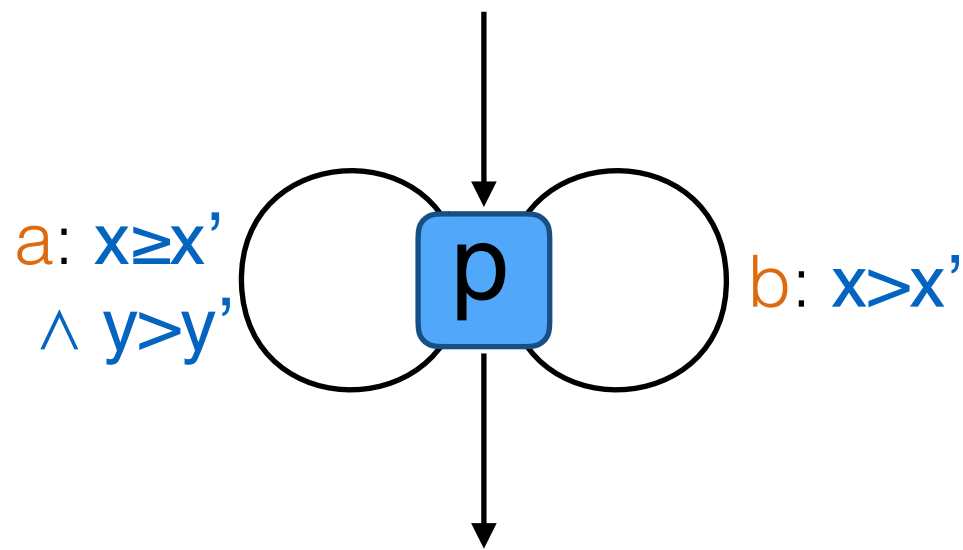


A **size-change abstraction terminates** if it has no infinite **run**.

Size-change abstraction

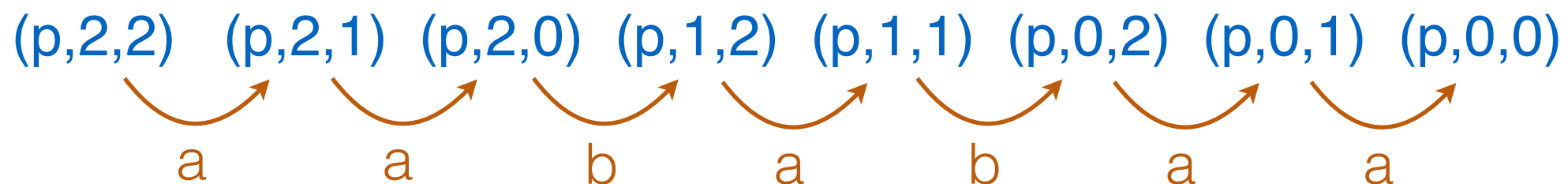
[Ben-Amram et al. 01] A **size-change abstraction (SCA)**:

- this is a non-deterministic finite state machine
- that uses a finite set **variables** (x, y, z, \dots) ranging over **non-negative integers**
- during each **transition**, a guards relate the variables before and after:
 $x \geq y'$ meaning « **val of x before the transition** \geq **val of y after the transition** »
 $x > y'$ meaning « **val of x before the transition** $>$ **val of y after the transition** »



A **configuration** is a **state** together with a non-negative integer value for each of the **variables**.

A **run of the SCA** is a sequence of **configurations** that starts in an initial configuration, ends in a final one, and each consecutive configurations satisfy the guard of some possible transition.



A **size-change abstraction terminates** if it has no infinite **run**.

[Ben-Aram et al. 01] Termination of size-change abstraction is PSPACE.

Abstracting

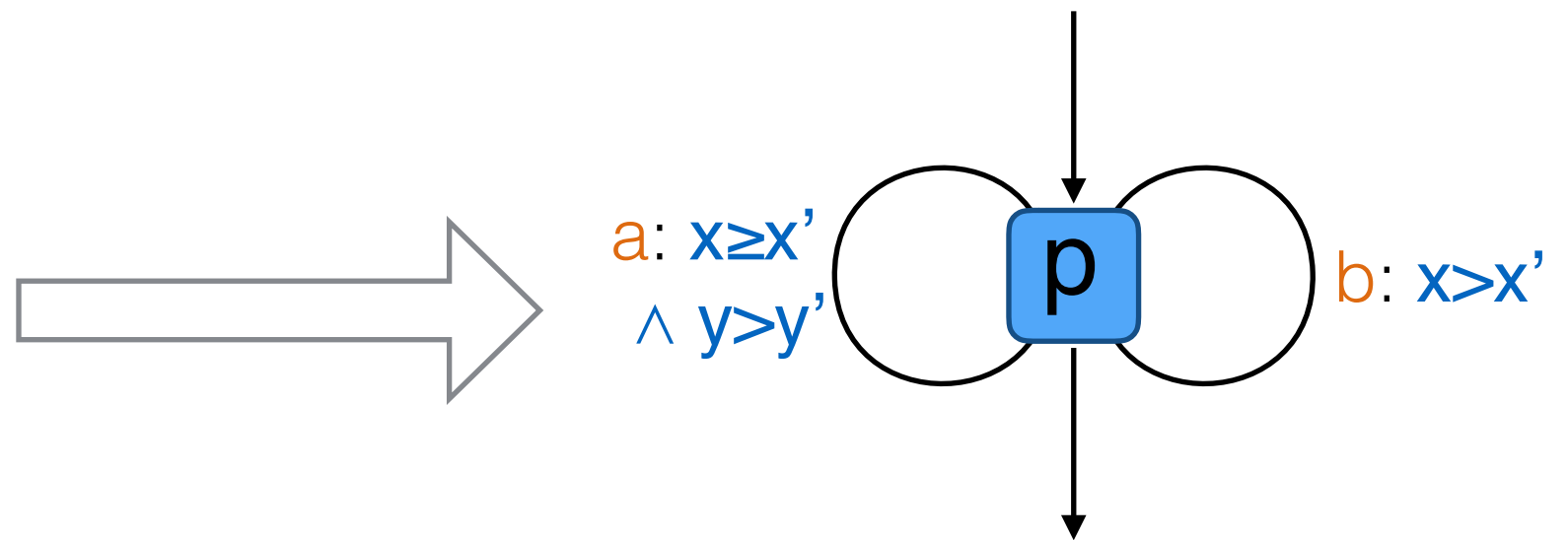
- fix quantities to keep track of, here **x,y** (can be other quantities)
- construct the control flow graph of the code
- use as guard the best ones you can infer

```
void main() {  
  uint x,y;  
  x = read_input();  
  y = read_input();  
  while (x > 0) {  
    if (y > 0)  
      { y--; }  
    else  
      { y = read_input();  
        x--; }  
  }  
}
```

Abstracting

- fix quantities to keep track of, here x, y (can be other quantities)
- construct the control flow graph of the code
- use as guard the best ones you can infer

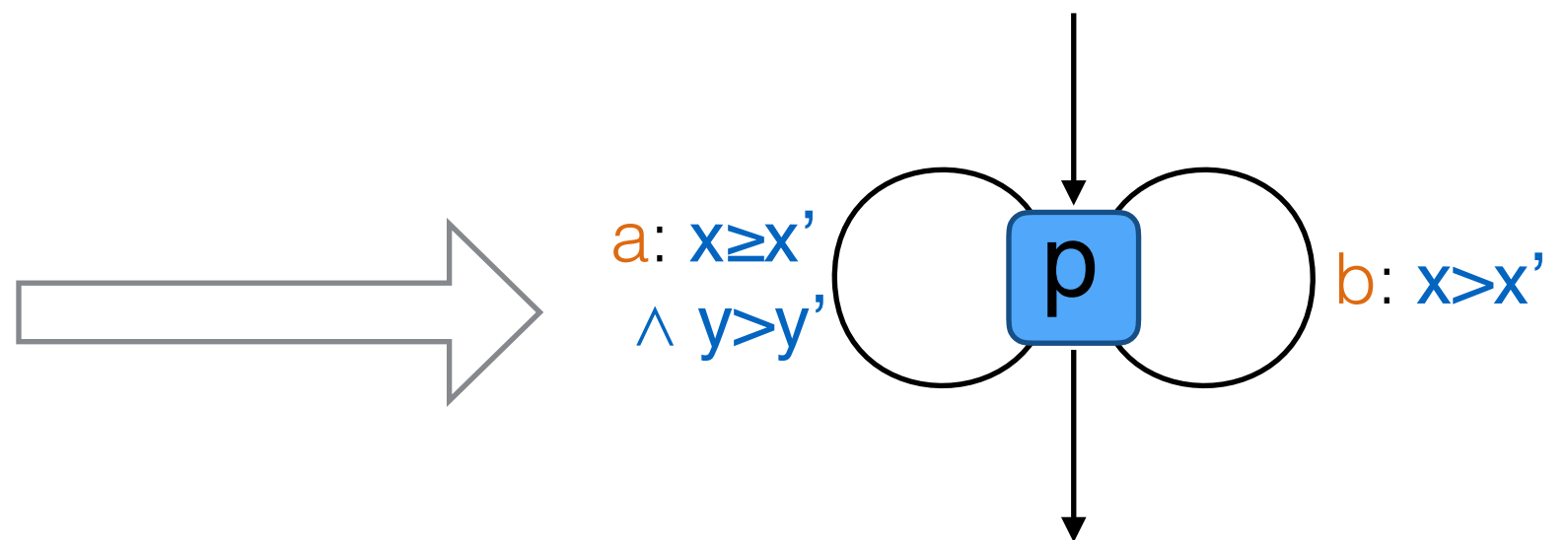
```
void main() {  
  uint x,y;  
  x = read_input();  
  y = read_input();  
  while (x > 0) {  
    if (y > 0)  
      { y--; }  
    else  
      { y = read_input();  
        x--; }  
  }  
}
```



Abstracting

- fix quantities to keep track of, here x, y (can be other quantities)
- construct the control flow graph of the code
- use as guard the best ones you can infer

```
void main() {  
  uint x,y;  
  x = read_input();  
  y = read_input();  
  while (x > 0) {  
    if (y > 0)  
      { y--; }  
    else  
      { y = read_input();  
        x--; }  
  }  
}
```



Remark: every **run** of the original program induces a **run of the SCA** of game size. Hence if the SCA terminates, the original program also does (on all its executions).

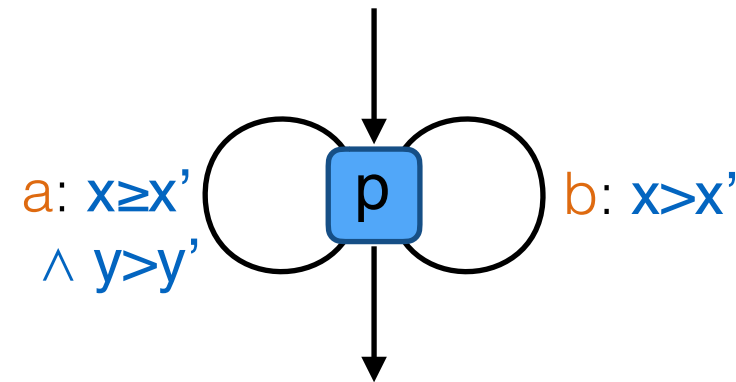
Deciding the termination of size-change abstraction

Deciding the termination of size-change abstraction

[Ben-Amram et al. 01]: The termination of **SCA** is decidable.

Deciding the termination of size-change abstraction

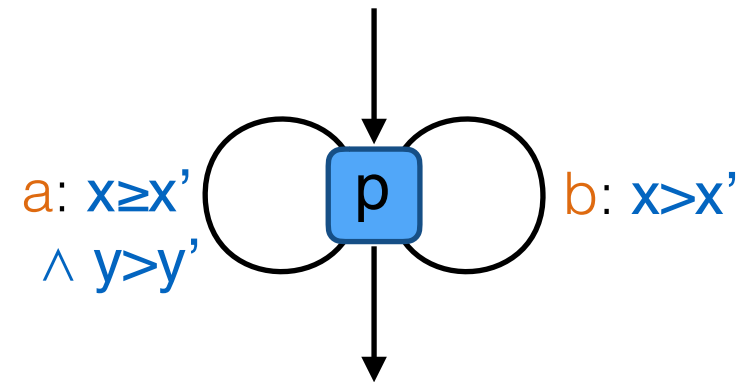
[Ben-Amram et al. 01]: The termination of **SCA** is decidable.



Deciding the termination of size-change abstraction

[Ben-Amram et al. 01]: The termination of **SCA** is decidable.

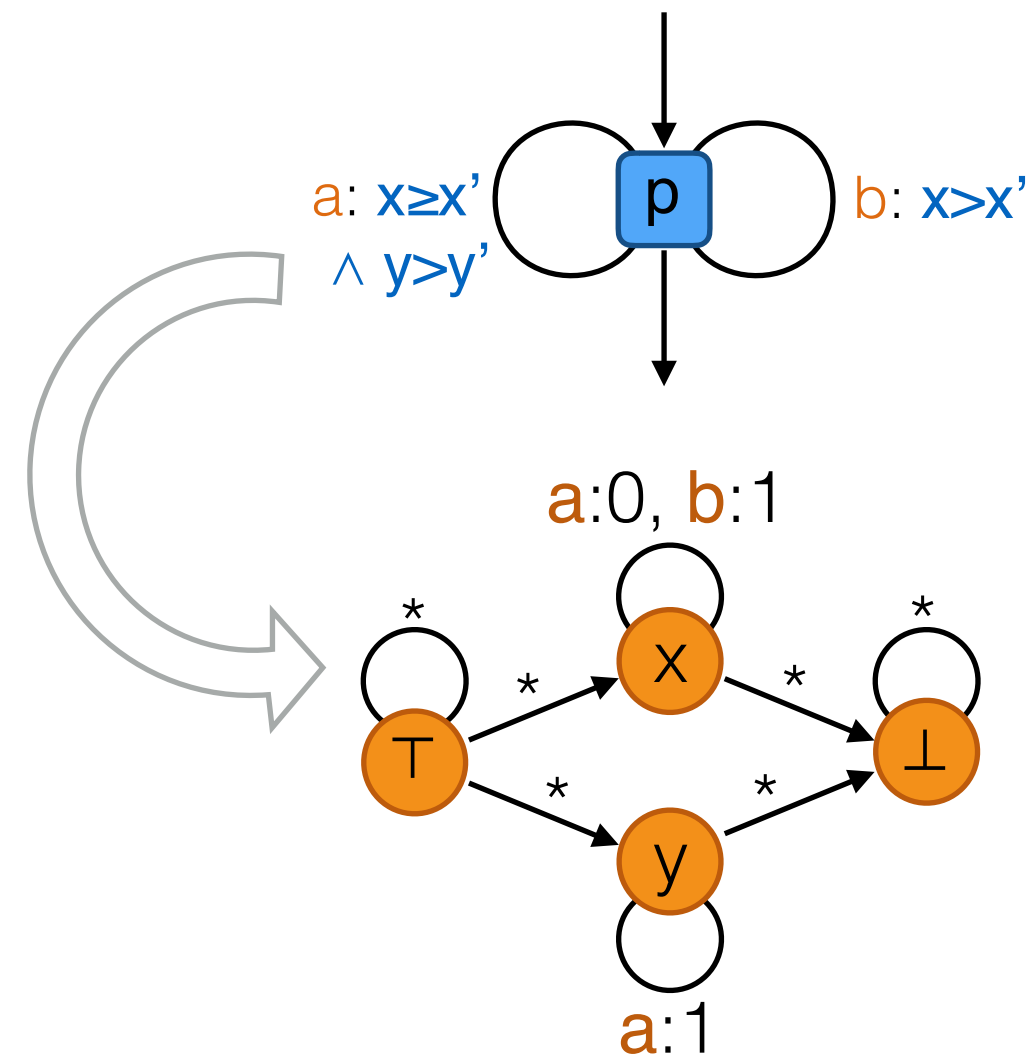
Proof: We construct a **Büchi automaton Aut** as follows.



Deciding the termination of size-change abstraction

[Ben-Amram et al. 01]: The termination of **SCA** is decidable.

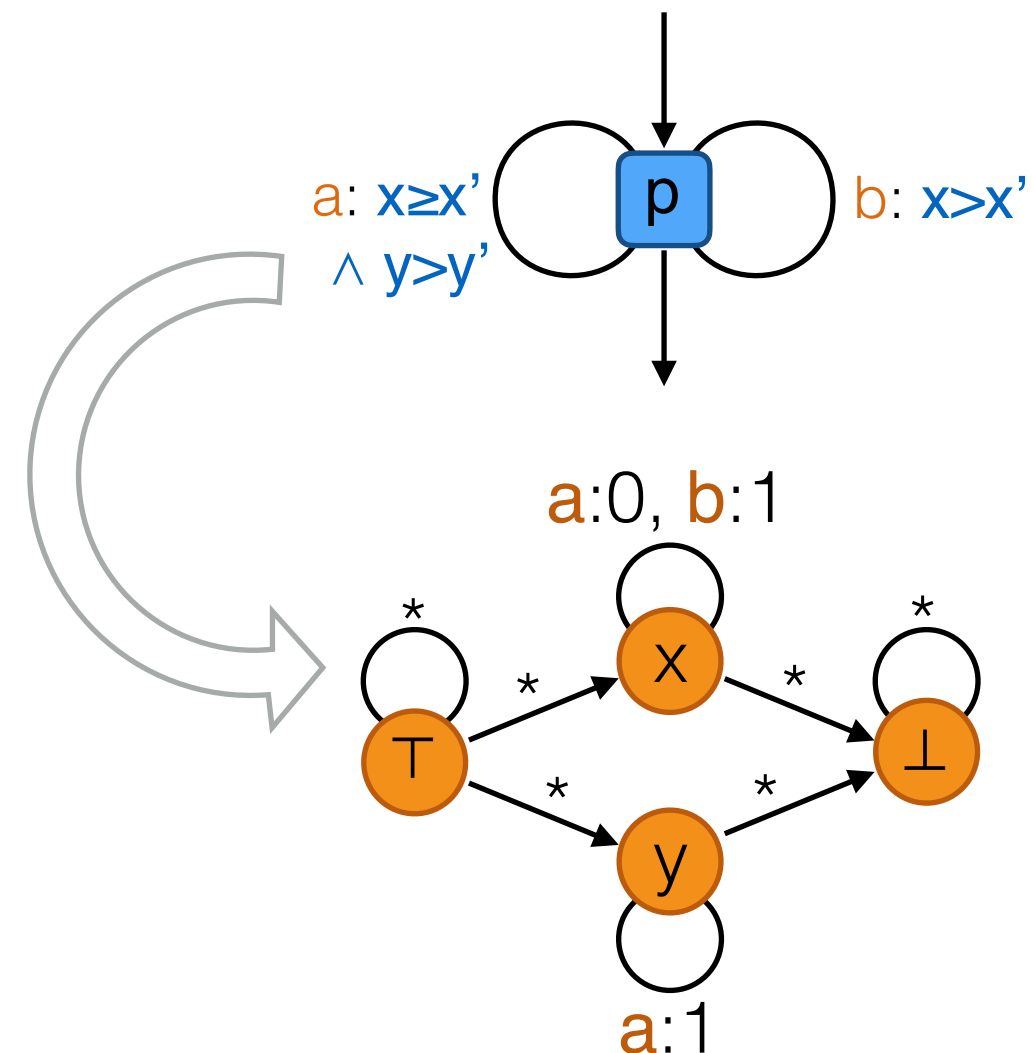
Proof: We construct a **Büchi automaton Aut** as follows.



Deciding the termination of size-change abstraction

[Ben-Amram et al. 01]: The termination of **SCA** is decidable.

Proof: We construct a **Büchi automaton Aut** as follows.
Take as alphabet the transitions of the **SCA**.



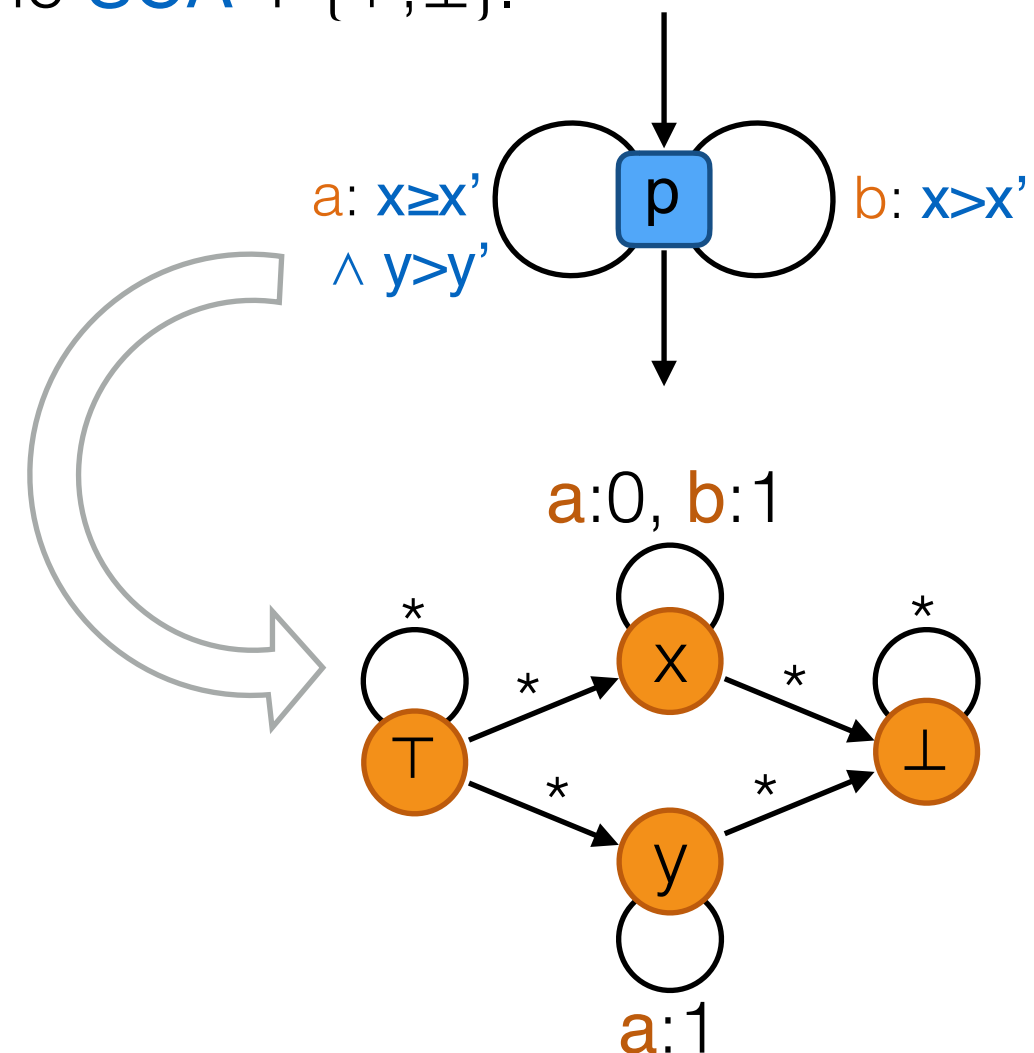
Deciding the termination of size-change abstraction

[Ben-Amram et al. 01]: The termination of **SCA** is decidable.

Proof: We construct a **Büchi automaton Aut** as follows.

Take as alphabet the transitions of the **SCA**.

Take as states of the automaton, the variables of the **SCA** + $\{\top, \perp\}$.



Deciding the termination of size-change abstraction

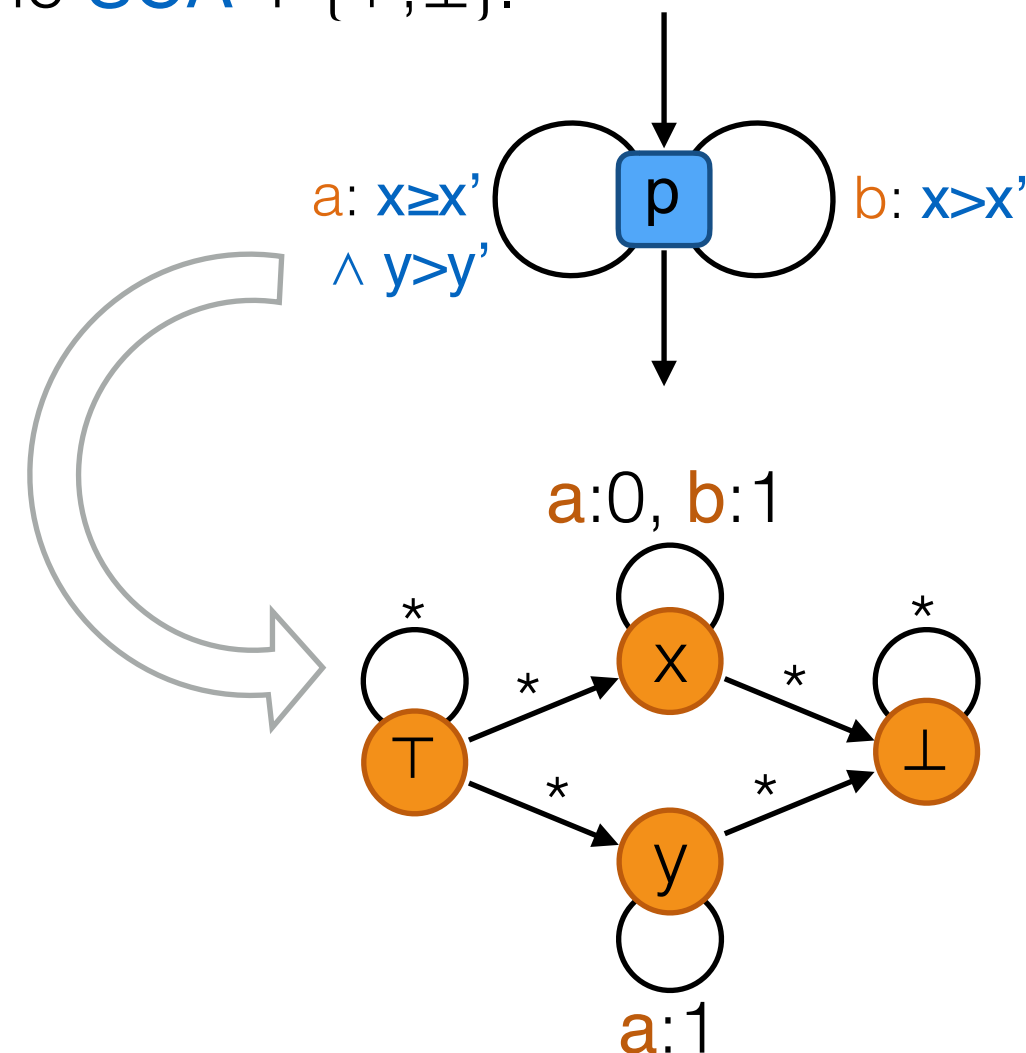
[Ben-Amram et al. 01]: The termination of **SCA** is decidable.

Proof: We construct a **Büchi automaton Aut** as follows.

Take as alphabet the transitions of the **SCA**.

Take as states of the automaton, the variables of the **SCA** + $\{\top, \perp\}$.

All states of the automaton are initial.



Deciding the termination of size-change abstraction

[Ben-Amram et al. 01]: The termination of **SCA** is decidable.

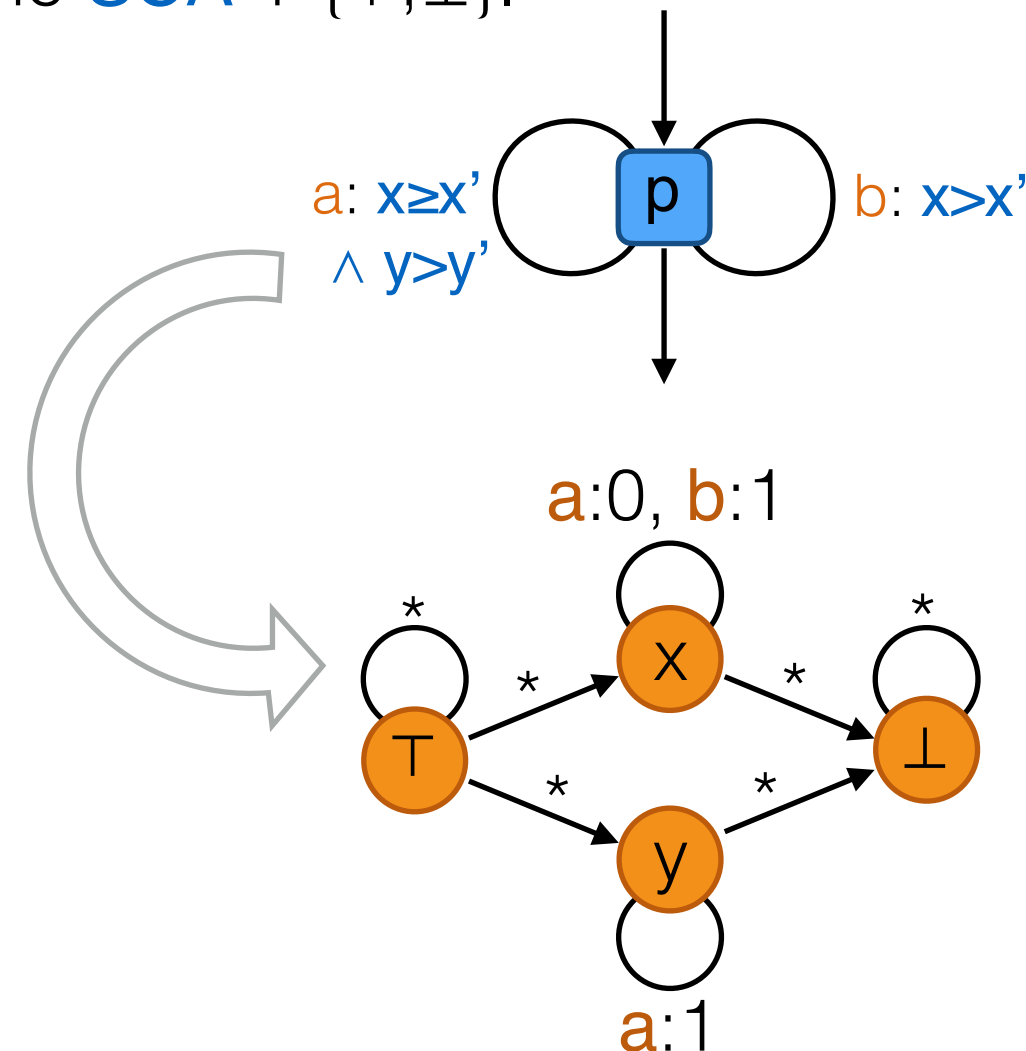
Proof: We construct a **Büchi automaton** **Aut** as follows.

Take as alphabet the transitions of the **SCA**.

Take as states of the automaton, the variables of the **SCA** + $\{\top, \perp\}$.

All states of the automaton are initial.

$$\Delta(x, a, y) = \begin{cases} 0 & \text{if there is a guard } x \geq y' \text{ in } a \\ 1 & \text{if there is a guard } x > y' \text{ in } a \\ -\infty & \text{otherwise (no guard)} \end{cases}$$



Deciding the termination of size-change abstraction

[Ben-Amram et al. 01]: The termination of **SCA** is decidable.

Proof: We construct a **Büchi automaton Aut** as follows.

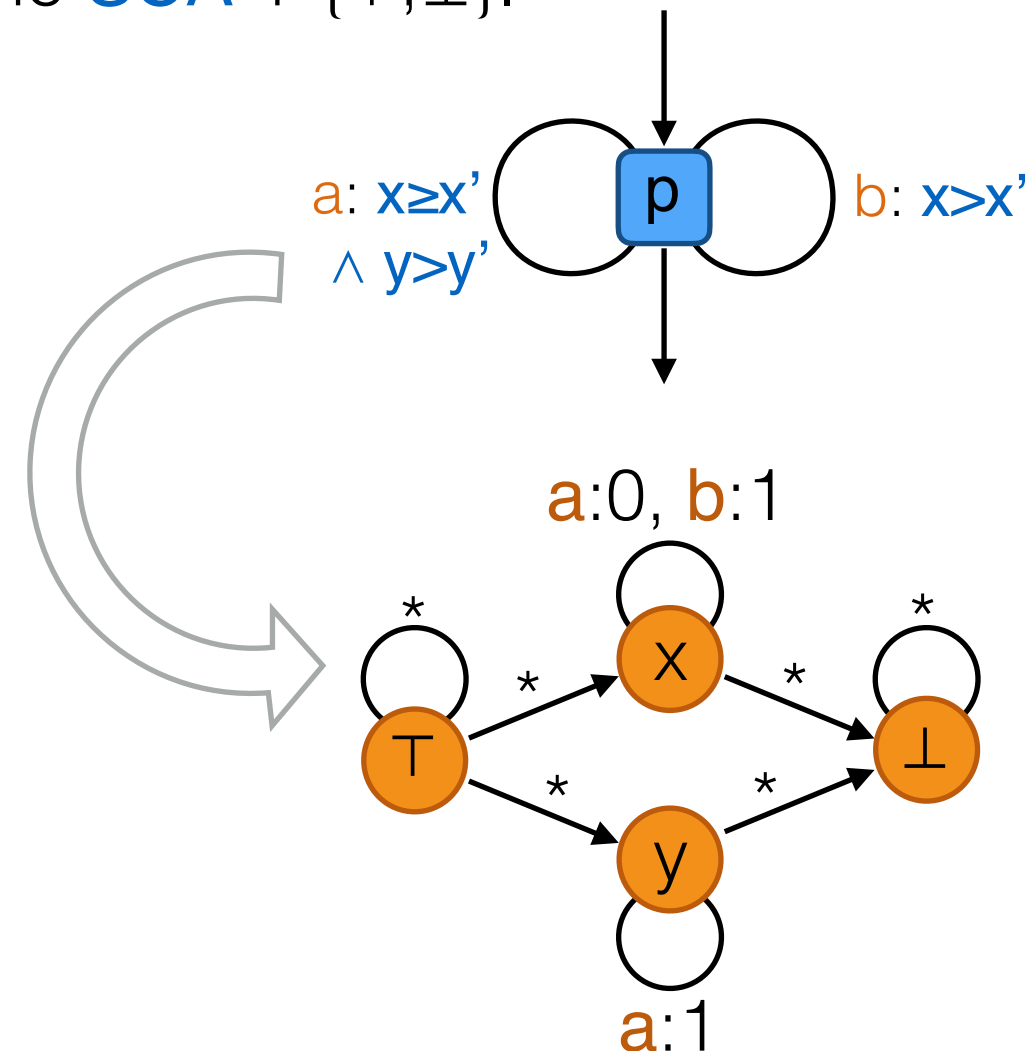
Take as alphabet the transitions of the **SCA**.

Take as states of the automaton, the variables of the **SCA** + $\{\top, \perp\}$.

All states of the automaton are initial.

$$\Delta(x, a, y) = \begin{cases} 0 & \text{if there is a guard } x \geq y' \text{ in } a \\ 1 & \text{if there is a guard } x > y' \text{ in } a \\ -\infty & \text{otherwise (no guard)} \end{cases}$$

$(\Delta(\perp, ?, ?) = 0, \Delta(?, ?, \top) = 0)$



Deciding the termination of size-change abstraction

[Ben-Amram et al. 01]: The termination of **SCA** is decidable.

Proof: We construct a **Büchi automaton Aut** as follows.

Take as alphabet the transitions of the **SCA**.

Take as states of the automaton, the variables of the **SCA** + $\{\top, \perp\}$.

All states of the automaton are initial.

$$\Delta(x, a, y) = \begin{cases} 0 & \text{if there is a guard } x \geq y' \text{ in } a \\ 1 & \text{if there is a guard } x > y' \text{ in } a \\ -\infty & \text{otherwise (no guard)} \end{cases}$$

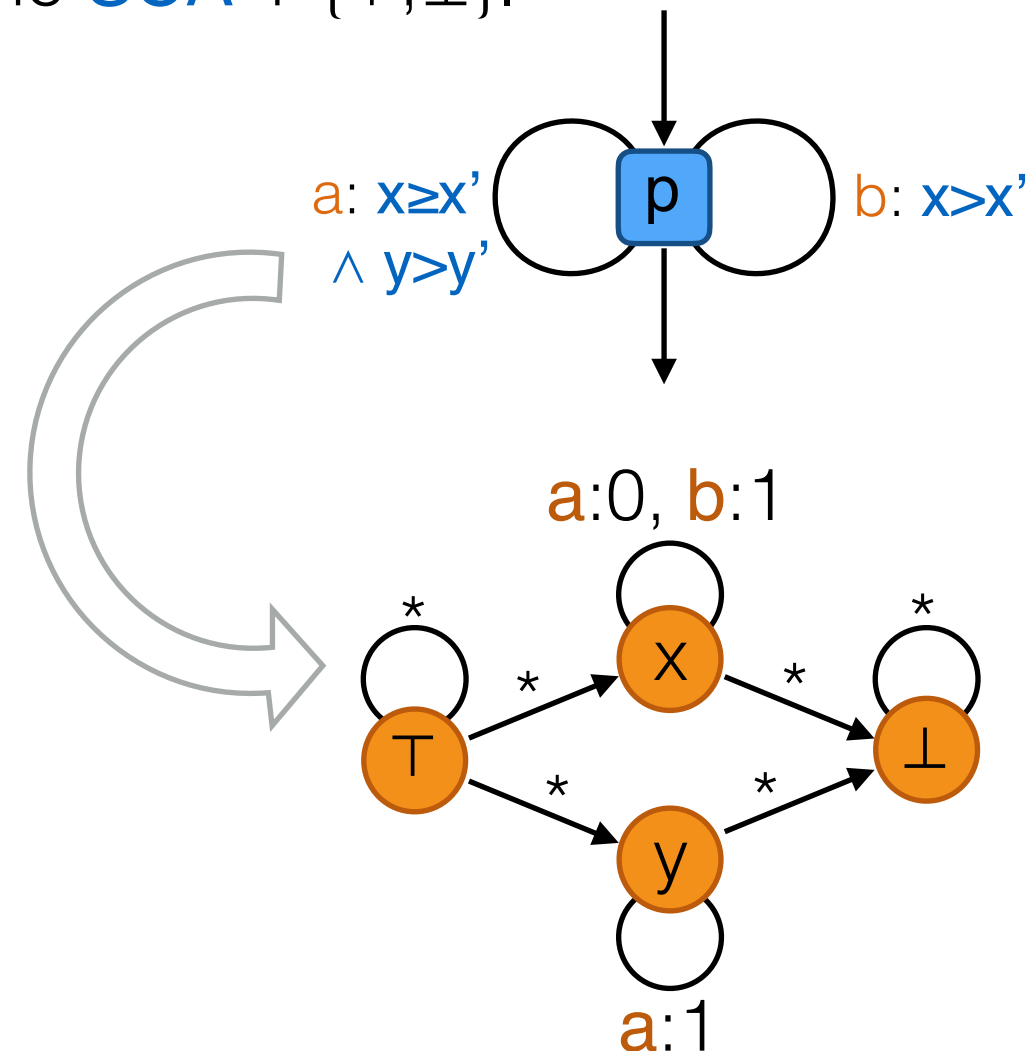
($\Delta(\perp, ?, ?) = 0$, $\Delta(?, ?, \top) = 0$)

Claim: \exists run ρ of **SCA**



\exists input word u for **Aut** of same length such that

- 1) it is a value-free valid run (regular)
- 2) there is no run of **Aut** with infinitely many 1's (Büchi condition)



Deciding the termination of size-change abstraction

[Ben-Amram et al. 01]: The termination of **SCA** is decidable.

Proof: We construct a **Büchi automaton Aut** as follows.

Take as alphabet the transitions of the **SCA**.

Take as states of the automaton, the variables of the **SCA** + $\{\top, \perp\}$.

All states of the automaton are initial.

$$\Delta(x, a, y) = \begin{cases} 0 & \text{if there is a guard } x \geq y' \text{ in } a \\ 1 & \text{if there is a guard } x > y' \text{ in } a \\ -\infty & \text{otherwise (no guard)} \end{cases}$$

($\Delta(\perp, ?, ?) = 0$, $\Delta(?, ?, \top) = 0$)

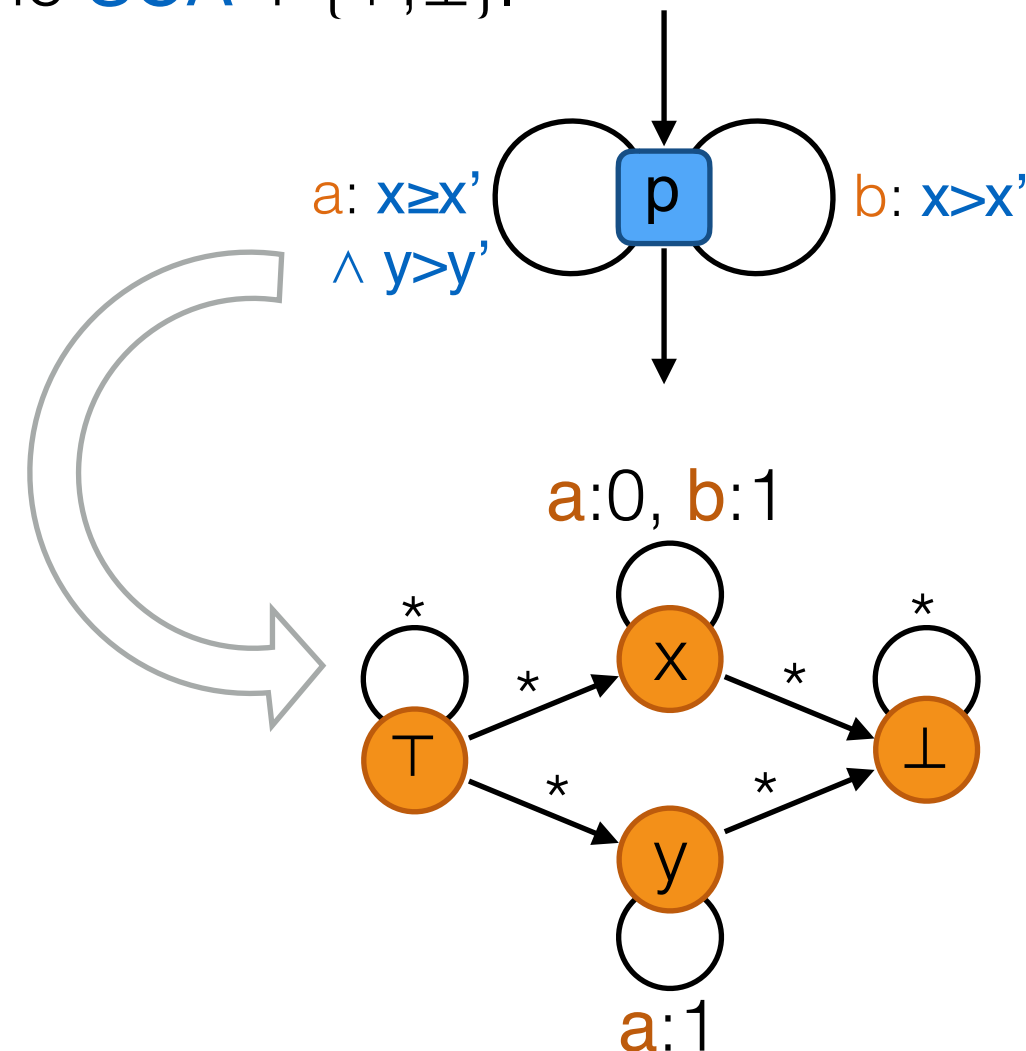
Claim: \exists run ρ of **SCA**



\exists input word u for **Aut** of same length such that

- 1) it is a value-free valid run (regular)
- 2) there is no run of **Aut** with infinitely many 1's (Büchi condition)

$\Rightarrow \text{Runs/Aut} = \emptyset ?$



Deciding the termination of size-change abstraction

[Ben-Amram et al. 01]: The termination of **SCA** is decidable.

Proof: We construct a **Büchi automaton Aut** as follows.

Take as alphabet the transitions of the **SCA**.

Take as states of the automaton, the variables of the **SCA** + $\{\top, \perp\}$.

All states of the automaton are initial.

$$\Delta(x, a, y) = \begin{cases} 0 & \text{if there is a guard } x \geq y' \text{ in } a \\ 1 & \text{if there is a guard } x > y' \text{ in } a \\ -\infty & \text{otherwise (no guard)} \end{cases}$$

($\Delta(\perp, ?, ?) = 0$, $\Delta(?, ?, \top) = 0$)

Claim: \exists run ρ of **SCA**

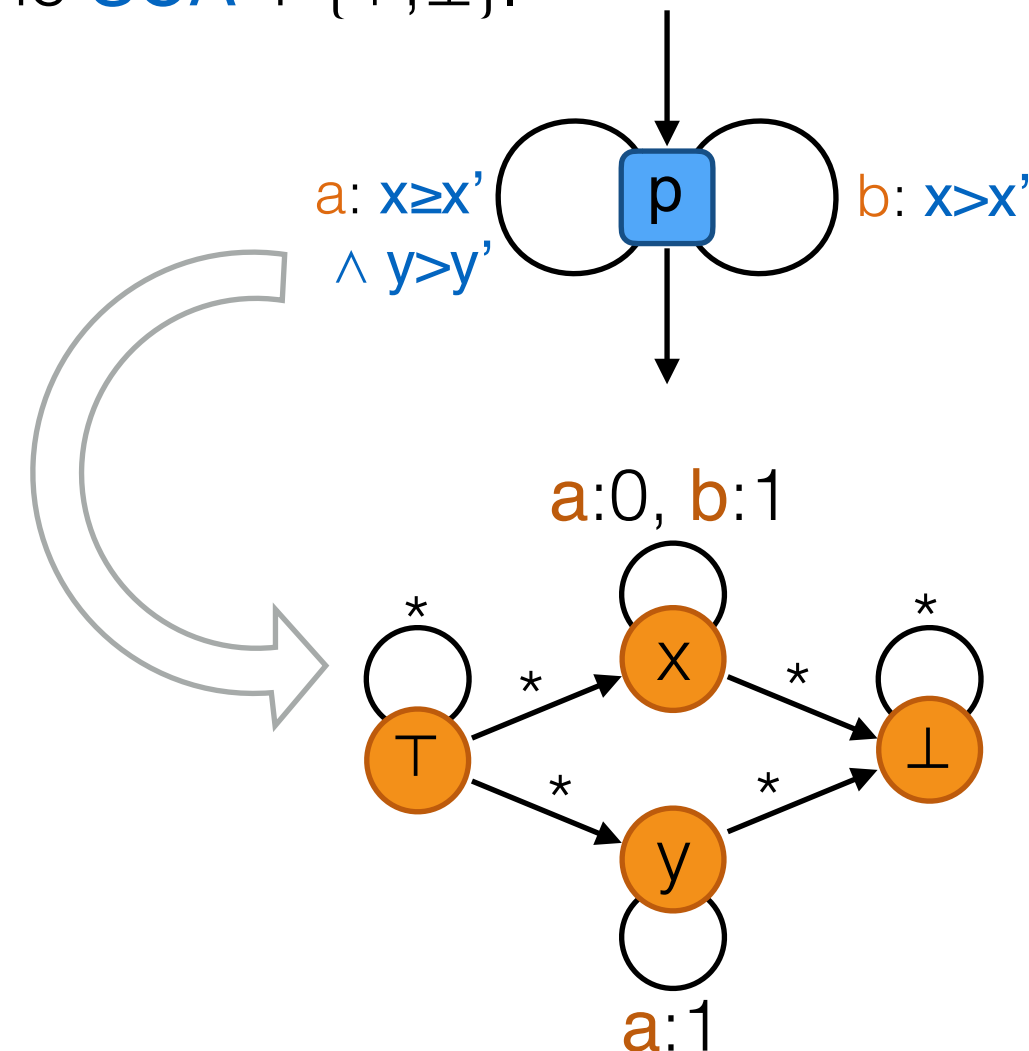


\exists input word u for **Aut** of same length such that

- 1) it is a value-free valid run (regular)
- 2) there is no run of **Aut** with infinitely many 1's (Büchi condition)

$\Rightarrow \text{Runs/Aut} = \emptyset ?$

$\Rightarrow \text{PSPACE}$



Overall picture

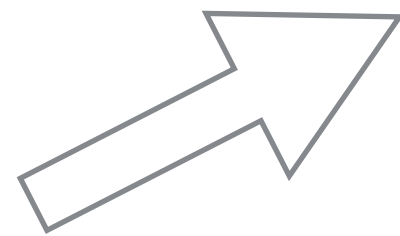
```
void main() {  
  uint x,y;  
  x = read_input();  
  y = read_input();  
  while (x > 0) {  
    if (y > 0)  
      { y--; }  
    else  
      { y = read_input();  
        x--; }  
  }  
}
```

Some
code

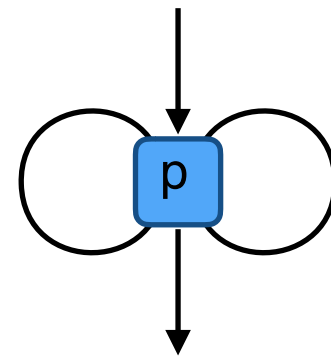
?

Does it terminate?

reflects
termination



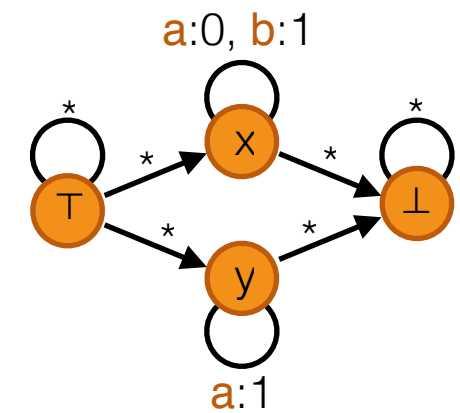
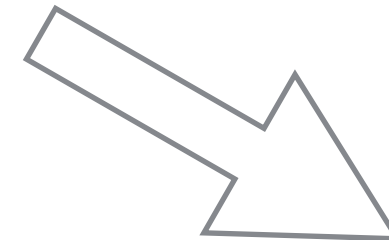
a: $x \geq x'$
 $\wedge y > y'$



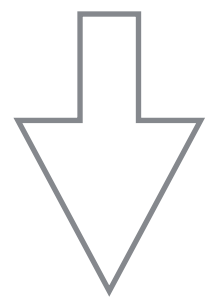
size-change
abstraction

b: $x > x'$

equivalent
for termination



Büchi
automaton



Decide an
inclusion
problem for
Büchi automata

Finer program analysis

Termination

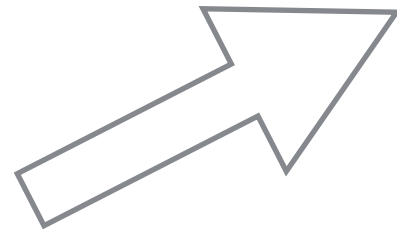
```
void main() {  
  uint x,y;  
  x = read_input();  
  y = read_input();  
  while (x > 0) {  
    if (y > 0)  
      { y--; }  
    else  
      { y = read_input();  
        x--; }  
  }  
}
```

Some
code

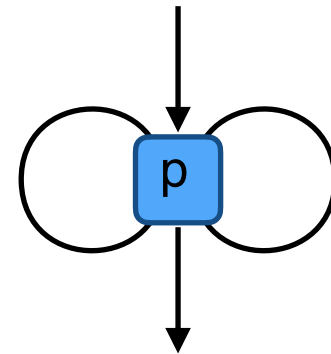
?

does it terminate?

reflects
termination



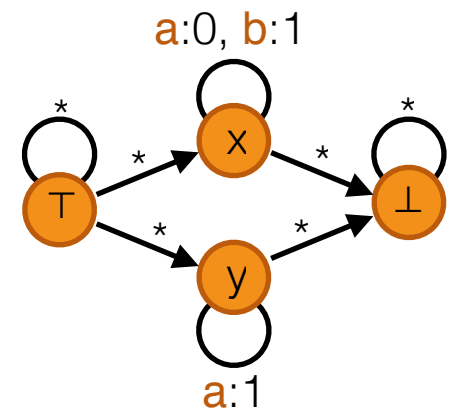
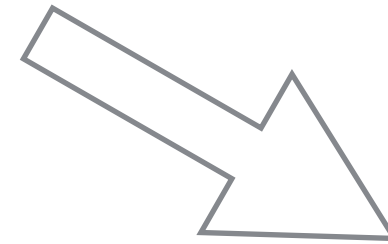
a: $x \geq x'$
 $\wedge y > y'$



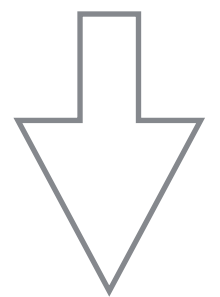
b: $x > x'$

size-change
abstraction

equivalent
for termination



Büchi
automaton



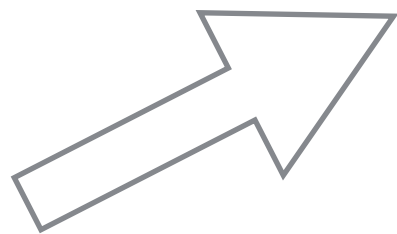
Decide an
inclusion
problem for
Büchi automata

Asymptotic complexity

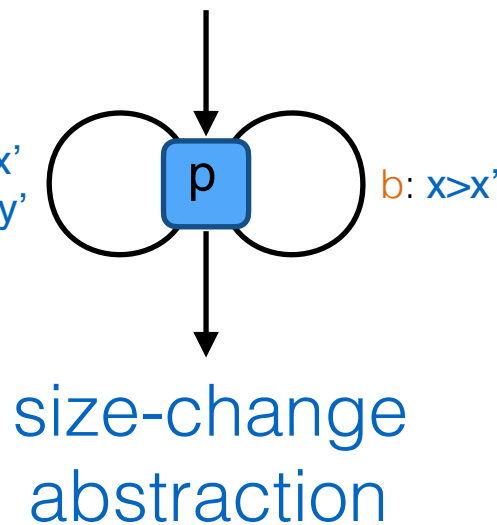
```
void main(uint n) {  
  uint x,y;  
  x = read_input(n);  
  y = read_input(n);  
  while (x > 0) {  
    if (y > 0)  
      { y--; }  
    else  
      { y = read_input(n);  
        x--; }  
  }  
}
```

Some
code

reflects
complexity

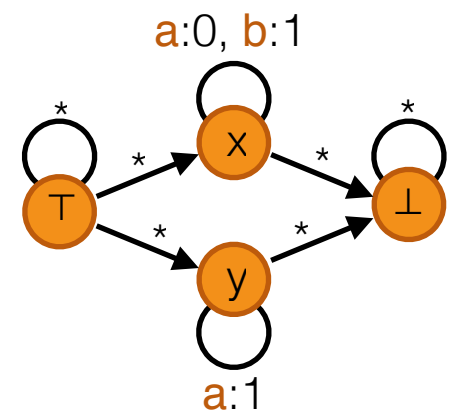
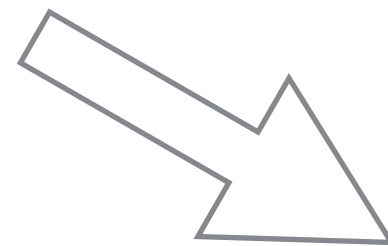


a: $x \geq x'$
 $\wedge y > y'$

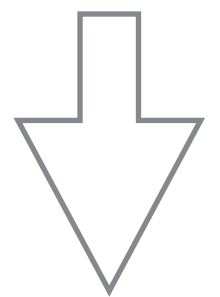


size-change
abstraction

equivalent
for complexity



N-max-plus
automaton



What is its complexity?
(as a function of a parameter **n**)

More precisely, find **a** such that
the program stops in $\Theta(n^a)$.

Compute the
asymptotic
worst-case
behavior

Abstracting

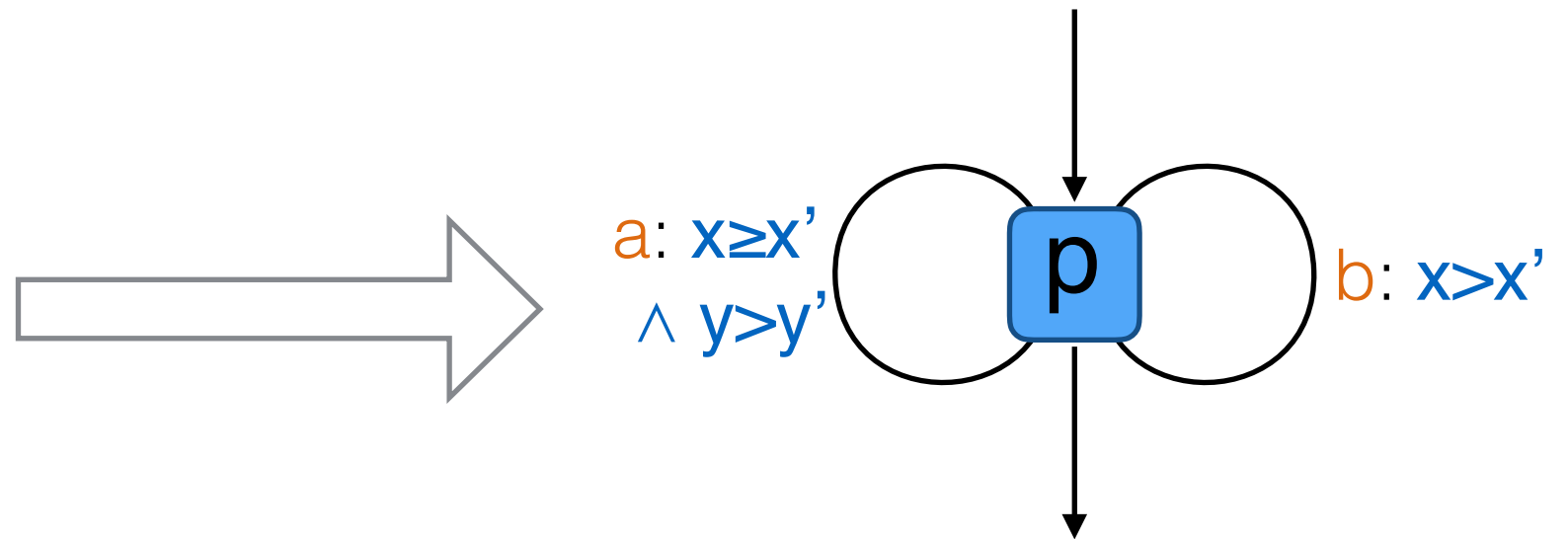
- fix quantities to keep track of, here **x,y** (can be other quantities)
- construct the control flow graph of the code
- use as guard the best ones you can infer

```
void main(uint n) {  
    uint x,y;  
    x = read_input(n);  
    y = read_input(n);  
    while (x > 0) {  
        if (y > 0)  
            { y--; }  
        else  
            { y = read_input(n);  
              x--; }  
    }  
}
```

Abstracting

- fix quantities to keep track of, here x, y (can be other quantities)
- construct the control flow graph of the code
- use as guard the best ones you can infer

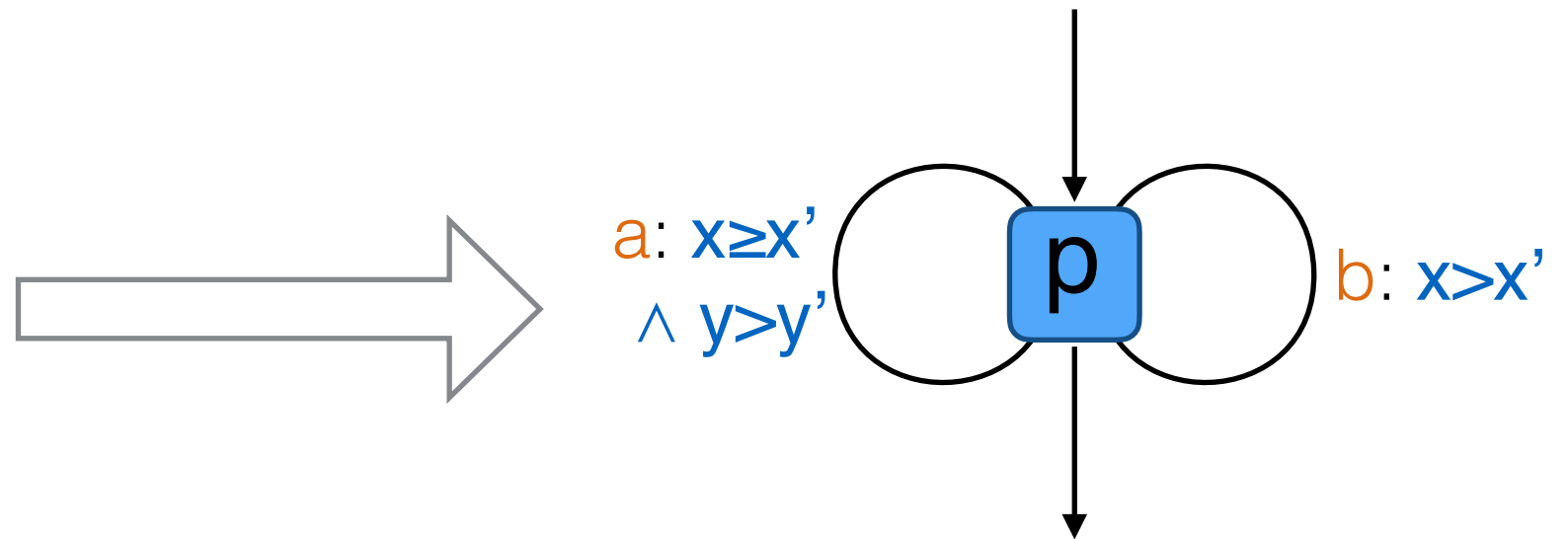
```
void main(uint n) {  
  uint x,y;  
  x = read_input(n);  
  y = read_input(n);  
  while (x > 0) {  
    if (y > 0)  
      { y--; }  
    else  
      { y = read_input(n);  
        x--; }  
  }  
}
```



Abstracting

- fix quantities to keep track of, here x, y (can be other quantities)
- construct the control flow graph of the code
- use as guard the best ones you can infer

```
void main(uint n) {  
  uint x,y;  
  x = read_input(n);  
  y = read_input(n);  
  while (x > 0) {  
    if (y > 0)  
      { y--; }  
    else  
      { y = read_input(n);  
        x--; }  
  }  
}
```

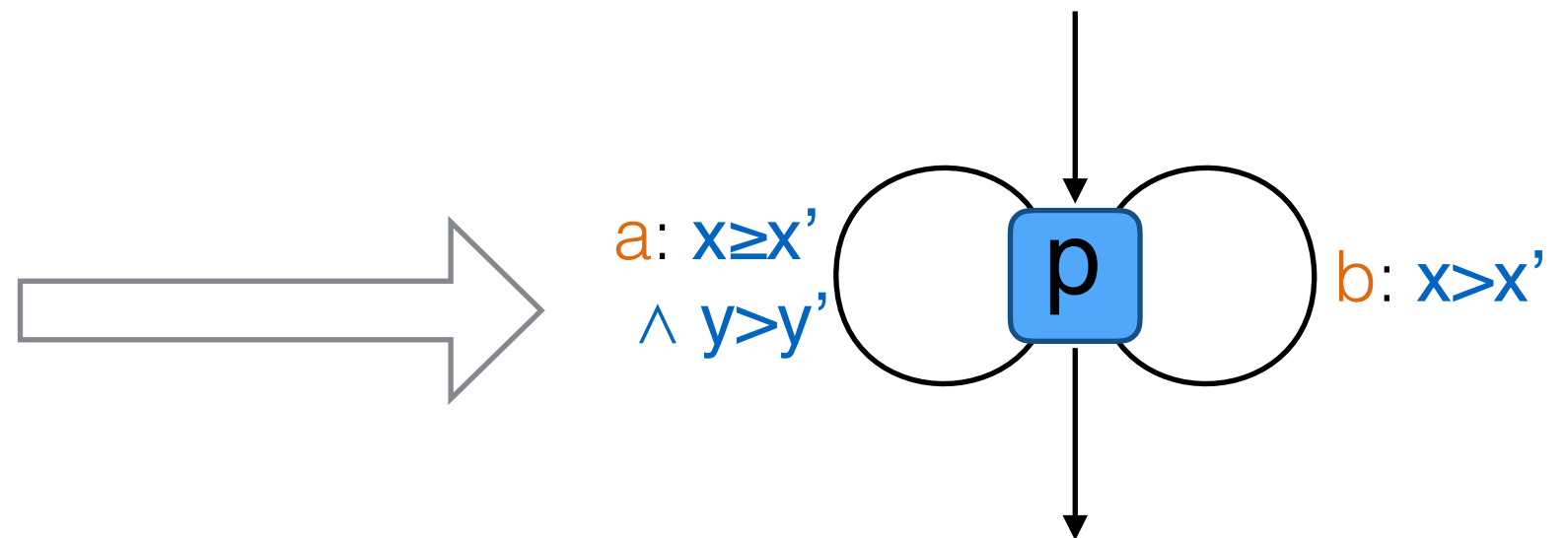


An **n-run** of the SCA is a run in which all the variables take their values in $[1, n]$

Abstracting

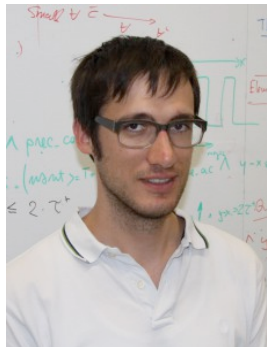
- fix quantities to keep track of, here x, y (can be other quantities)
- construct the control flow graph of the code
- use as guard the best ones you can infer

```
void main(uint n) {  
  uint x,y;  
  x = read_input(n);  
  y = read_input(n);  
  while (x > 0) {  
    if (y > 0)  
      { y--; }  
    else  
      { y = read_input(n);  
        x--; }  
  }  
}
```



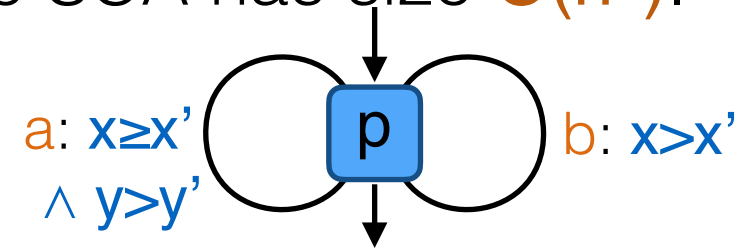
An **n-run** of the SCA is a run in which all the variables take their values in $[1, n]$

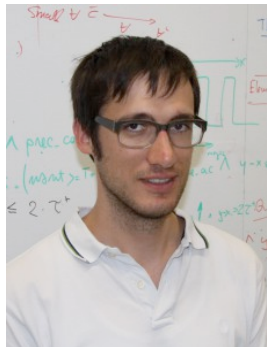
Remark: every **run** of the original program for a given **n** induces an **n-run** of the **SCA** of same length. Hence if the **SCA** terminates in time **t** for a given **n**, the original program also does (on all its executions).



Complexity analysis

[C., Daviaud, Zuleger 14] If the SCA terminates, there exists a computable rational α such that the worst-case length of an n -run of the SCA has size $\Theta(n^\alpha)$.





Complexity analysis

[C., Daviaud, Zuleger 14] If the SCA terminates, there exists a computable rational α such that the worst-case length of an n -run of the SCA has size $\Theta(n^\alpha)$.

Proof: We construct a Büchi automaton Aut as follows:

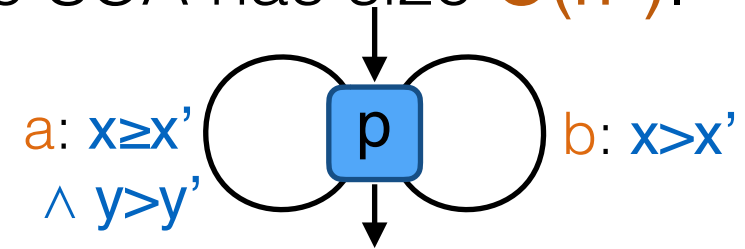
Take as alphabet the transitions of the SCA.

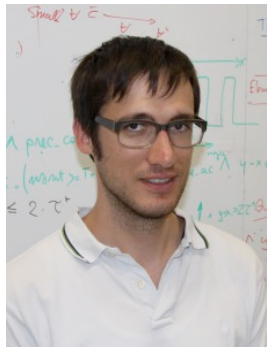
Take as states of the automaton, the variables of the SCA + $\{\top, \perp\}$.

All states of the automaton are initial and final.

$$\Delta(x, a, y) = \begin{cases} 0 & \text{if there is a guard } x \geq y' \text{ in } a \\ 1 & \text{if there is a guard } x > y' \text{ in } a \\ -\infty & \text{otherwise (no guard)} \end{cases}$$

$$(\Delta(\perp, ?, ?) = 0, \Delta(?, ?, \top) = 0)$$





Complexity analysis

[C., Daviaud, Zuleger 14] If the SCA terminates, there exists a computable rational α such that the worst-case length of an n -run of the SCA has size $\Theta(n^\alpha)$.

Proof: We construct a Büchi automaton Aut as follows:

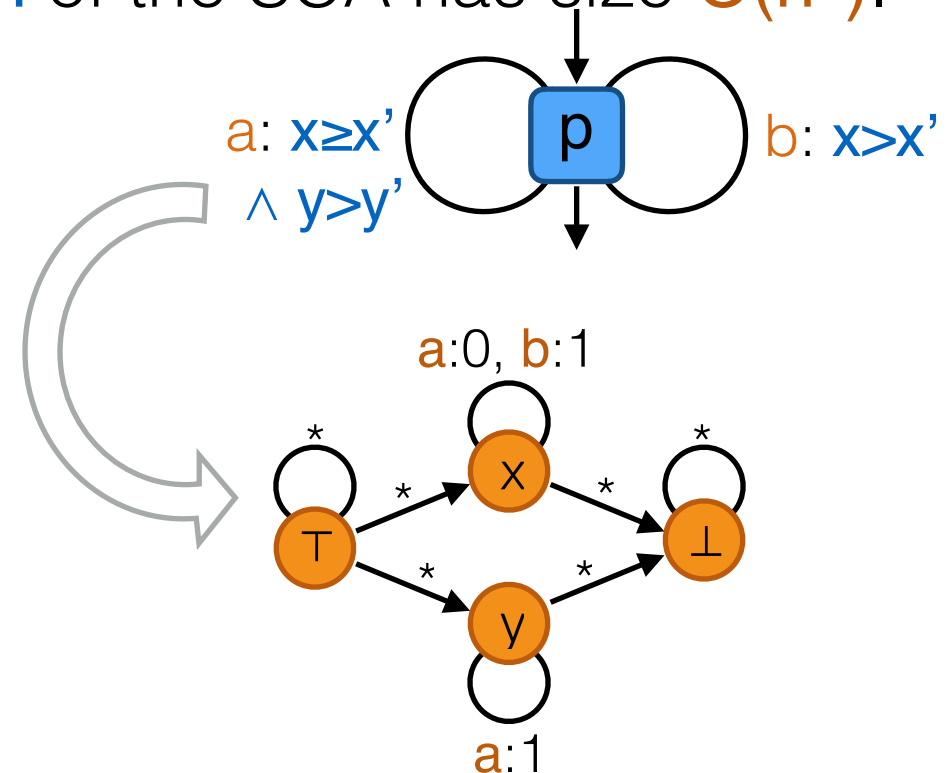
Take as alphabet the transitions of the SCA.

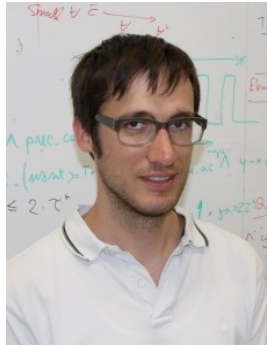
Take as states of the automaton, the variables of the SCA + $\{\top, \perp\}$.

All states of the automaton are initial and final.

$$\Delta(x, a, y) = \begin{cases} 0 & \text{if there is a guard } x \geq y' \text{ in } a \\ 1 & \text{if there is a guard } x > y' \text{ in } a \\ -\infty & \text{otherwise (no guard)} \end{cases}$$

$$(\Delta(\perp, ?, ?) = 0, \Delta(?, ?, \top) = 0)$$





Complexity analysis

[C., Daviaud, Zuleger 14] If the SCA terminates, there exists a computable rational α such that the worst-case length of an n -run of the SCA has size $\Theta(n^\alpha)$.

Proof: We construct a Büchi automaton Aut as follows:

Take as alphabet the transitions of the SCA.

Take as states of the automaton, the variables of the SCA + $\{\top, \perp\}$.

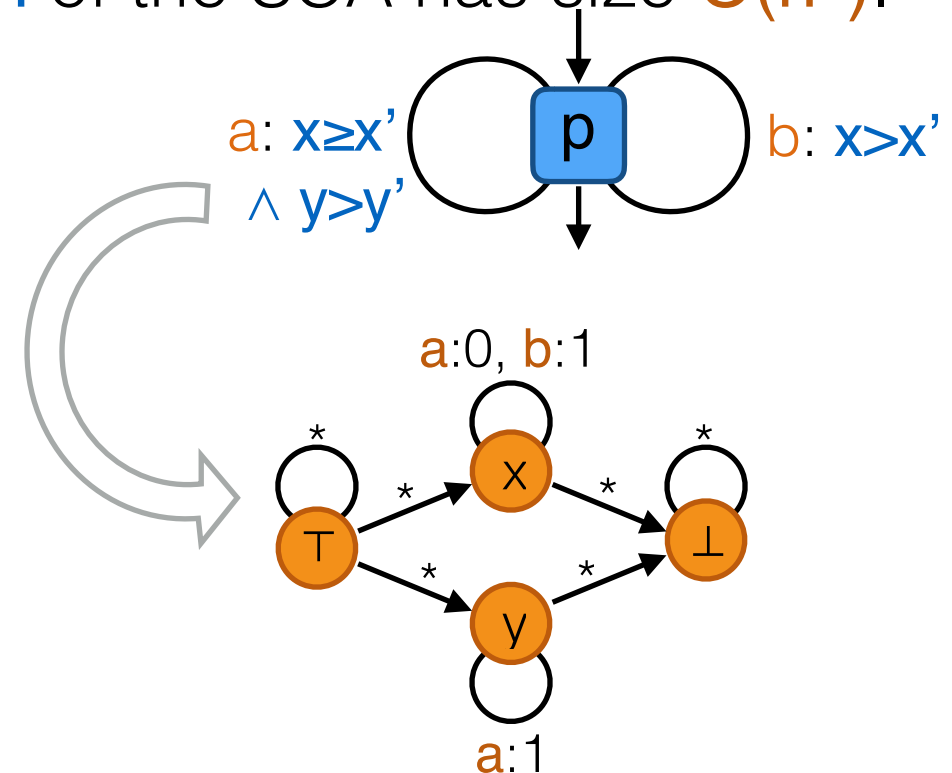
All states of the automaton are initial and final.

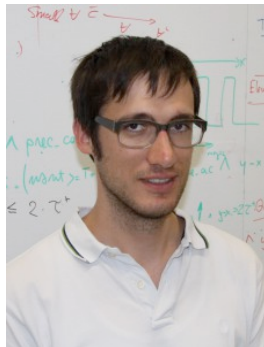
$$\Delta(x, a, y) = \begin{cases} 0 & \text{if there is a guard } x \geq y' \text{ in } a \\ 1 & \text{if there is a guard } x > y' \text{ in } a \\ -\infty & \text{otherwise (no guard)} \end{cases}$$

$$(\Delta(\perp, ?, ?) = 0, \Delta(?, ?, \top) = 0)$$

Claim: $(\exists n\text{-run of SCA of size } s)$

if and only if $\left(\begin{array}{l} \exists \text{ input word } u \text{ of size } s \text{ such that} \\ 1) \text{ it is a value-free valid run (regular)} \\ 2) \text{ there is no run of Aut with weight } > n. \end{array} \right)$





Complexity analysis

[C., Daviaud, Zuleger 14] If the SCA terminates, there exists a computable rational α such that the worst-case length of an n -run of the SCA has size $\Theta(n^\alpha)$.

Proof: We construct a Büchi automaton Aut as follows:

Take as alphabet the transitions of the SCA.

Take as states of the automaton, the variables of the SCA + $\{\top, \perp\}$.

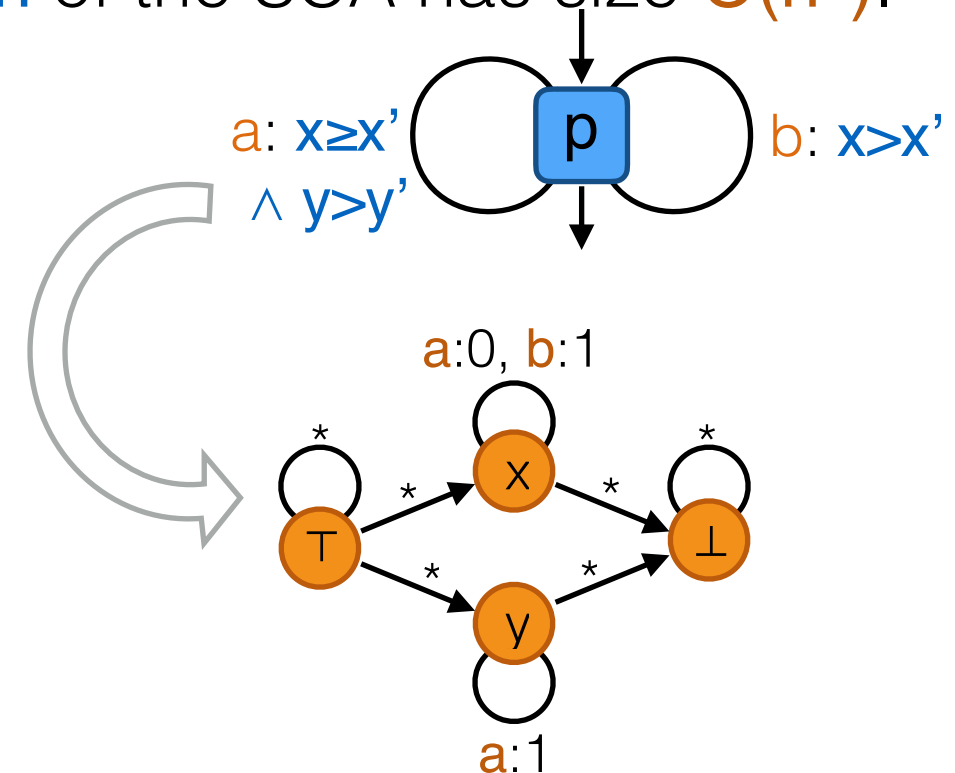
All states of the automaton are initial and final.

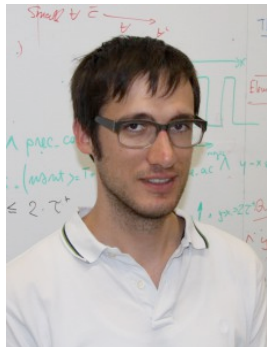
$$\Delta(x, a, y) = \begin{cases} 0 & \text{if there is a guard } x \geq y' \text{ in } a \\ 1 & \text{if there is a guard } x > y' \text{ in } a \\ -\infty & \text{otherwise (no guard)} \end{cases}$$

$$(\Delta(\perp, ?, ?) = 0, \Delta(?, ?, \top) = 0)$$

Claim: $(\exists n\text{-run of SCA of size } s)$

if and only if $\left(\begin{array}{l} \exists \text{ input word } u \text{ of size } s \text{ such that} \\ \text{1) it is a value-free valid run (regular)} \\ \text{2) there is no run of Aut with weight } > n. \end{array} \right)$





Complexity analysis

[C., Daviaud, Zuleger 14] If the SCA terminates, there exists a computable rational α such that the worst-case length of an n -run of the SCA has size $\Theta(n^\alpha)$.

Proof: We construct a Büchi automaton Aut as follows:

Take as alphabet the transitions of the SCA.

Take as states of the automaton, the variables of the SCA + $\{\top, \perp\}$.

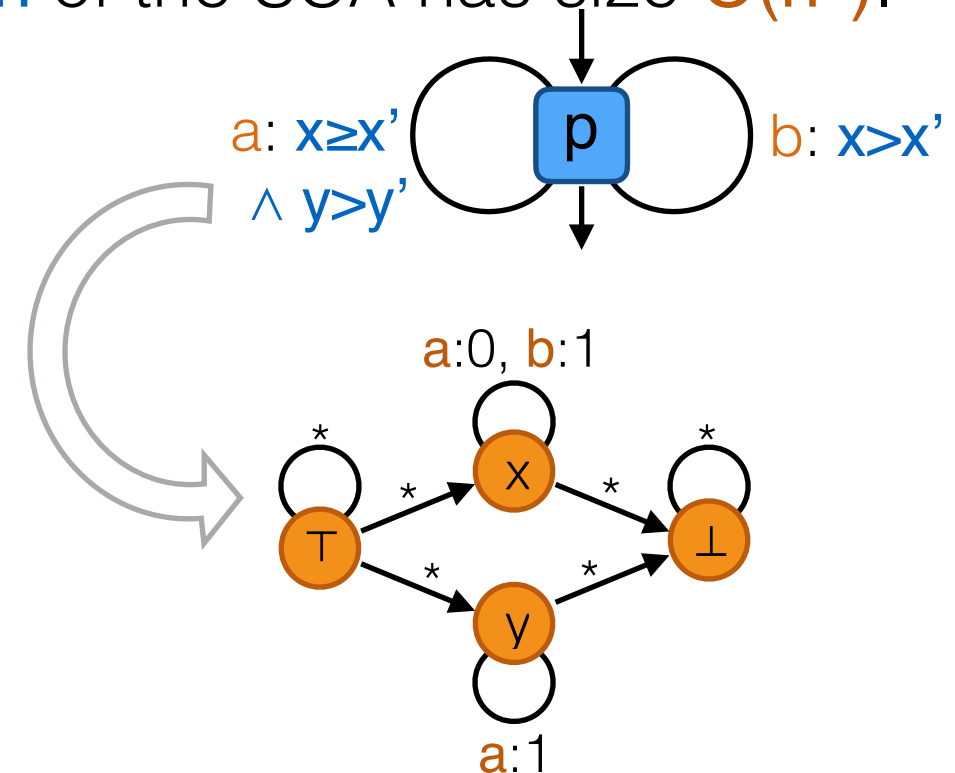
All states of the automaton are initial and final.

$$\Delta(x, a, y) = \begin{cases} 0 & \text{if there is a guard } x \geq y' \text{ in } a \\ 1 & \text{if there is a guard } x > y' \text{ in } a \\ -\infty & \text{otherwise (no guard)} \end{cases}$$

$$(\Delta(\perp, ?, ?) = 0, \Delta(?, ?, \top) = 0)$$

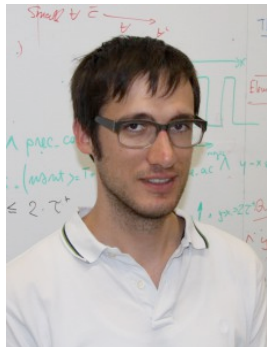
Claim: $(\exists n\text{-run of SCA of size } s)$

if and only if $\left(\begin{array}{l} \exists \text{ input word } u \text{ of size } s \text{ such that} \\ \text{1) it is a value-free valid run (regular)} \\ \text{2) there is no run of Aut with weight } > n. \end{array} \right)$



One needs to find the asymptotic exponent of the size of the longest word that has only run of value at most n :

$$\limsup_{u \in A^*} \frac{\log |u|}{\log \text{Aut}(|u|)} = \alpha$$



Complexity analysis

[C., Daviaud, Zuleger 14] If the SCA terminates, there exists a computable rational α such that the worst-case length of an n -run of the SCA has size $\Theta(n^\alpha)$.

Proof: We construct a Büchi automaton Aut as follows:

Take as alphabet the transitions of the SCA.

Take as states of the automaton, the variables of the SCA + $\{\top, \perp\}$.

All states of the automaton are initial and final.

$$\Delta(x, a, y) = \begin{cases} 0 & \text{if there is a guard } x \geq y' \text{ in } a \\ 1 & \text{if there is a guard } x > y' \text{ in } a \\ -\infty & \text{otherwise (no guard)} \end{cases}$$

$$(\Delta(\perp, ?, ?) = 0, \Delta(?, ?, \top) = 0)$$

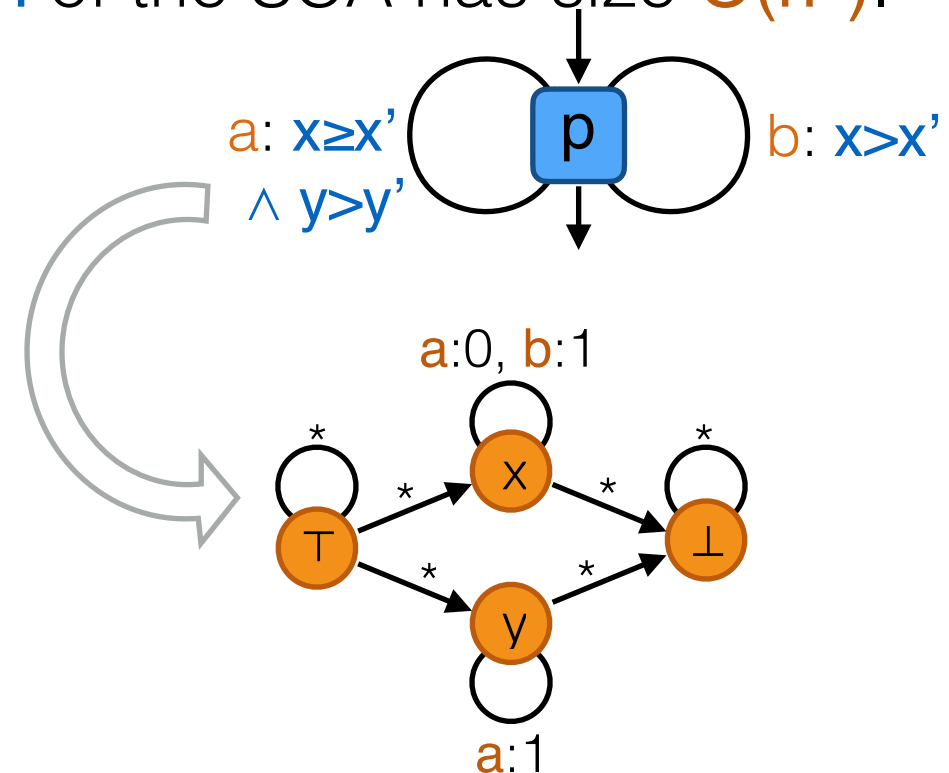
Claim: $(\exists n\text{-run of SCA of size } s)$

if and only if $\left(\begin{array}{l} \exists \text{ input word } u \text{ of size } s \text{ such that} \\ \text{1) it is a value-free valid run (regular)} \\ \text{2) there is no run of Aut with weight } > n. \end{array} \right)$

One needs to find the asymptotic exponent of the size of the longest word that has only run of value at most n :

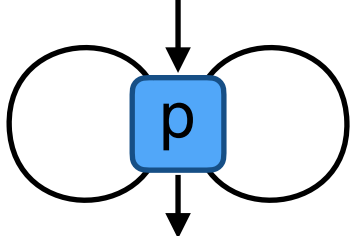
$$\limsup_{u \in A^*} \frac{\log |u|}{\log \text{Aut}(|u|)} = \alpha$$

\Rightarrow Decidable.

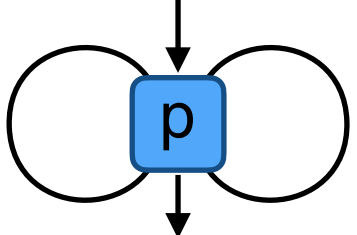


An unexpected
phenomenon

An unexpected phenomenon

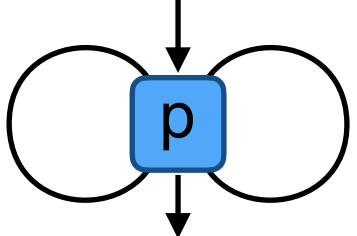
For instance, $a: x \geq x' \wedge y > y'$  $b: x > x'$ has worst-case complexity n^2 .

An unexpected phenomenon

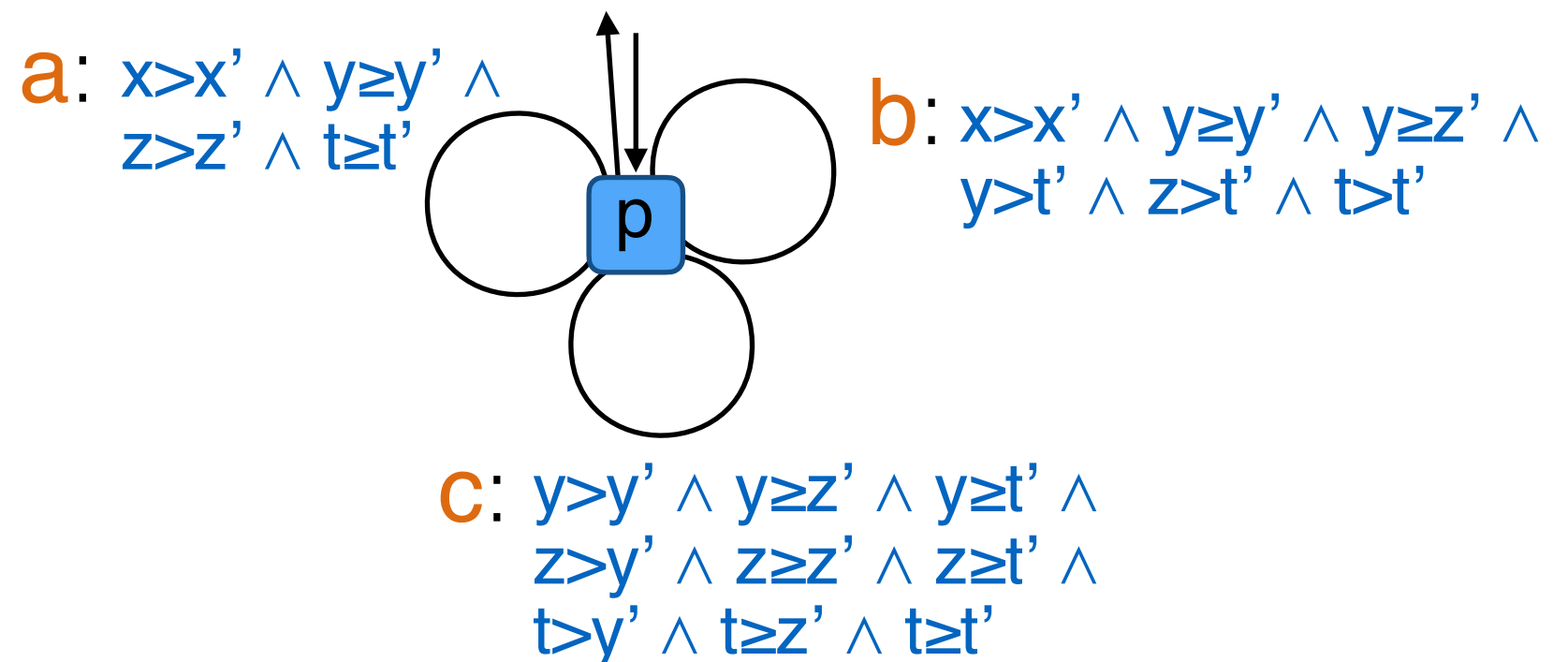
For instance, $a: x \geq x' \wedge y > y'$  $b: x > x'$ has worst-case complexity n^2 .

It was conjectured that the asymptotic worst-case could only have integer exponent.

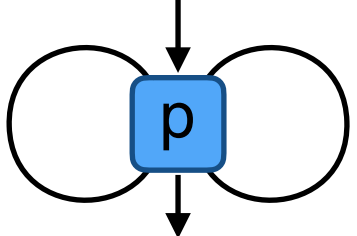
An unexpected phenomenon

For instance, $a: x \geq x' \wedge y > y'$  $b: x > x'$ has worst-case complexity n^2 .

It was conjectured that the asymptotic worst-case could only have integer exponent.



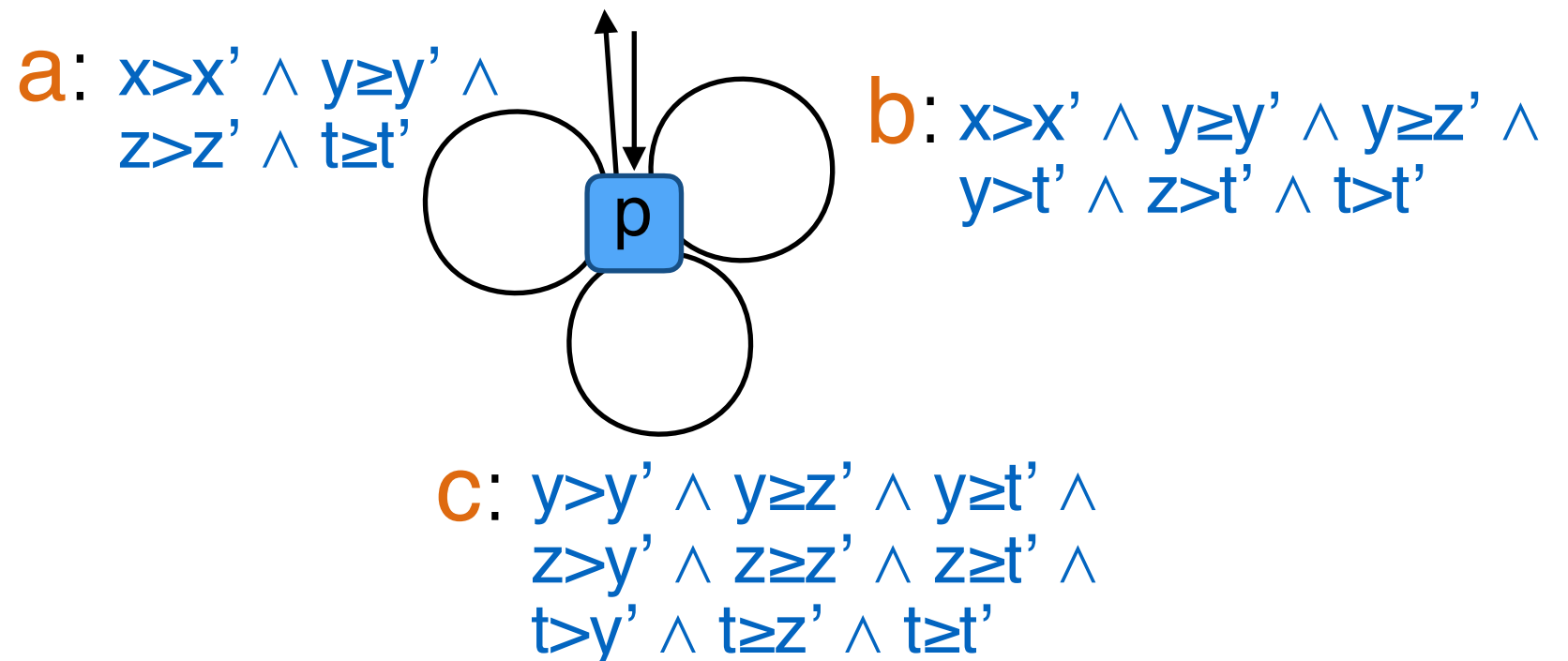
An unexpected phenomenon

For instance, $a: x \geq x' \wedge y > y'$  $b: x > x'$ has worst-case complexity n^2 .

It was conjectured that the asymptotic worst-case could only have integer exponent.

However:

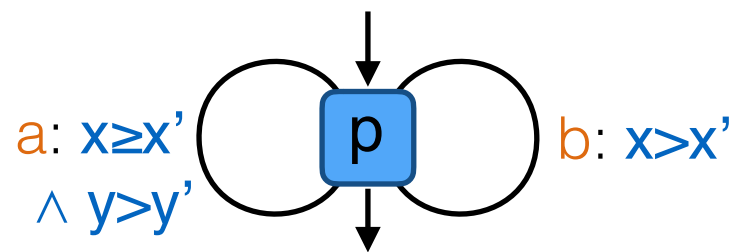
The longest n-run of the following SCA has asymptotical length $\Theta(n^{3/2})$.



Summary

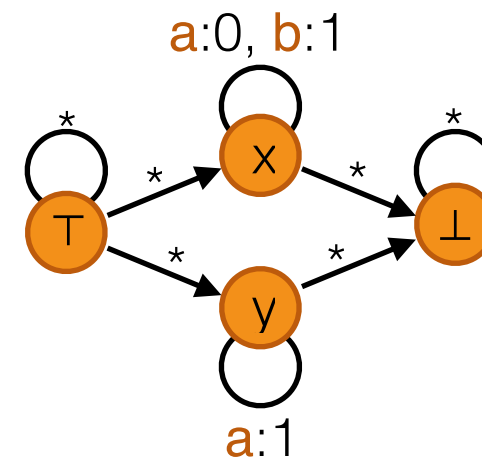
The size-change abstraction is good model for proving the termination of some forms of programs. This offers a natural reduction to question of automata theory.

```
void main(uint n) {  
  uint x,y;  
  x = read_input(n);  
  y = read_input(n);  
  while (x > 0) {  
    if (y > 0) {  
      { y--; }  
    }  
    else {  
      { y = read_input(n);  
        x--; }  
    }  
  }  
}
```



We have shown that this technique can be greatly refined for computing asymptotic worst-case complexity of some programs.

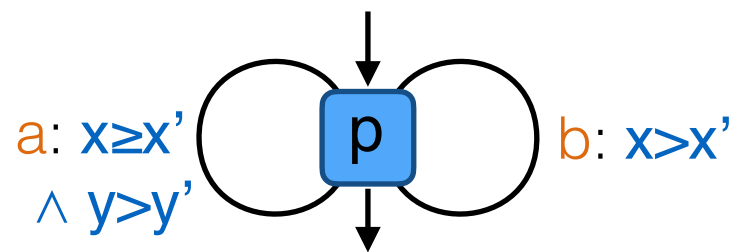
This relies on advanced results on the asymptotic analysis of tropical automata.



Summary

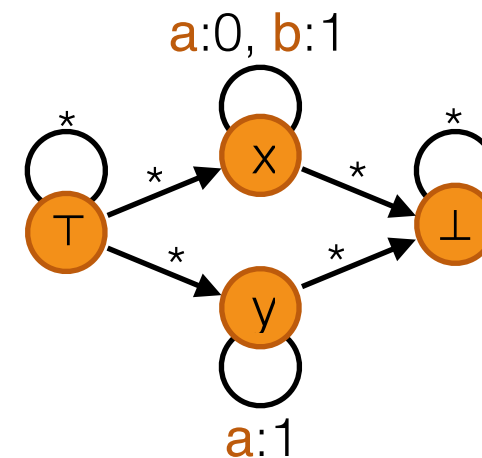
The size-change abstraction is good model for proving the termination of some forms of programs. This offers a natural reduction to question of automata theory.

```
void main(uint n) {  
  uint x,y;  
  x = read_input(n);  
  y = read_input(n);  
  while (x > 0) {  
    if (y > 0) {  
      { y--; }  
    }  
    else {  
      { y = read_input(n);  
        x--; }  
    }  
  }  
}
```



We have shown that this technique can be greatly refined for computing asymptotic worst-case complexity of some programs.

This relies on advanced results on the asymptotic analysis of tropical automata.



Some open questions

What is the exact complexity?

How to construct ranking functions?

Is there a more general model of automata and results?

Thanks !