Computational content of circular proof systems.

Denis Kuperberg   Laureline Pinault   Damien Pous

LIP, ENS Lyon

Warsaw Automata Seminar
Wednesday 13th May 2020

Curry-Howard correspondence

Proof of formula $\varphi \leftrightarrow$ Program of type φ

Example: The identity program is a proof of $p \to p$.

Curry-Howard correspondence

Proof of formula $\varphi \leftrightarrow$ Program of type φ

Example: The identity program is a proof of $p \to p$.

Deduction Rule

$$\text{implies}\downarrow \qquad \frac{A \qquad\qquad B}{C}$$

Curry-Howard correspondence

Proof of formula $\varphi \leftrightarrow$ Program of type φ

Example: The identity program is a proof of $p \to p$.

Deduction Rule

Program instruction

implies $\downarrow$ $\qquad \dfrac{A \qquad\qquad B}{C} \qquad$ $\uparrow$ calls

Curry-Howard correspondence

Proof of formula $\varphi \leftrightarrow$ Program of type φ

Example: The identity program is a proof of $p \to p$.

Deduction Rule

Program instruction

$$\text{implies}\downarrow \qquad \frac{A \qquad\qquad B}{C} \qquad \uparrow\text{calls}$$

Correspondence well-understood for usual proof systems

[Curry,Howard]	Intuitionistic logic	$\leftrightarrow$	Typed λ-calculus
[...]	...	$\leftrightarrow$	...

Curry-Howard correspondence

Proof of formula $\varphi \leftrightarrow$ Program of type φ

Example: The identity program is a proof of $p \to p$.

Deduction Rule

Program instruction

implies $\downarrow$

$$\frac{A \qquad\qquad B}{C}$$

$\uparrow$ calls

Correspondence well-understood for usual proof systems

[Curry,Howard]	Intuitionistic logic	$\leftrightarrow$	Typed λ-calculus
[...]	$\dots$	$\leftrightarrow$	$\dots$

This work: Study the computational content of cyclic proofs.

Cyclic Proofs

Usual proofs:

$$\dfrac{\text{Axiom}_1 \qquad \dfrac{\text{Axiom}_2}{A}}{B} \qquad \dfrac{\text{Axiom}_3}{C}$$

$$\dfrac{B \qquad C}{D}$$

Cyclic Proofs

Usual proofs:

$$\dfrac{\dfrac{\mathrm{Axiom}_1 \qquad \dfrac{\mathrm{Axiom}_2}{A}}{B} \qquad \dfrac{\mathrm{Axiom}_3}{C}}{D}$$

Cyclic Proofs:

$$\dfrac{\dfrac{\mathrm{Axiom} \qquad \dfrac{A}{A}}{B} \qquad \dfrac{D}{C}}{D}$$

Cyclic Proofs

Usual proofs:

$$\dfrac{\dfrac{\mathrm{Axiom}_1 \qquad \dfrac{\mathrm{Axiom}_2}{A}}{B} \qquad \dfrac{\mathrm{Axiom}_3}{C}}{D}$$

Cyclic Proofs:

$$\dfrac{\dfrac{\mathrm{Axiom} \qquad \dfrac{A}{A}}{B} \qquad \dfrac{D}{C}}{D}$$

Validity conditions: Cycles must contain particular rules.

Cyclic Proofs

Usual proofs:

$$\dfrac{\dfrac{\text{Axiom}_1 \qquad \dfrac{\text{Axiom}_2}{A}}{B} \qquad \dfrac{\text{Axiom}_3}{C}}{D}$$

Cyclic Proofs:

$$\dfrac{\dfrac{\text{Axiom} \qquad \dfrac{A}{A}}{B} \qquad \dfrac{D}{C}}{D}$$

Validity conditions: Cycles must contain particular rules.

As programs: recursive calls must be done on smaller arguments.

Cyclic Proofs

Usual proofs:

$$\dfrac{\dfrac{\text{Axiom}_1 \qquad \dfrac{\text{Axiom}_2}{A}}{B} \qquad \dfrac{\text{Axiom}_3}{C}}{D}$$

Cyclic Proofs:

$$\dfrac{\dfrac{\text{Axiom} \qquad \dfrac{A}{A}}{B} \qquad \dfrac{D}{C}}{D}$$

Validity conditions: Cycles must contain particular rules.

As programs: recursive calls must be done on smaller arguments.
$\rightarrow$ guarantees termination.

A proof system for regular expressions

Example: [Das, Pous '17]
Cyclic proof system for inclusion of regular expressions:

$$\dfrac{\dfrac{\dfrac{}{\varepsilon \subseteq \varepsilon}\,(\mathsf{Ax})}{\varepsilon \subseteq a^*}\,(*\text{-right}_1) \qquad \dfrac{\dfrac{}{a \subseteq a}\,(\mathsf{Ax}) \qquad a^* \subseteq a^*}{a, a^* \subseteq a^*}\,(*\text{-right}_2)}{a^* \subseteq a^*}\,(*\text{-left})$$

A proof system for regular expressions

Example: [Das, Pous '17]
Cyclic proof system for inclusion of regular expressions:

$$\dfrac{\dfrac{\dfrac{}{\varepsilon \subseteq \varepsilon}\,(\mathsf{Ax})}{\varepsilon \subseteq a^*}\,(\text{*-right}_1) \qquad \dfrac{\dfrac{}{a \subseteq a}\,(\mathsf{Ax}) \qquad a^* \subseteq a^*}{a, a^* \subseteq a^*}\,(\text{*-right}_2)}{a^* \subseteq a^*}\,(\text{*-left})$$

Soundness and completeness [Das, Pous '17]

$$L(e) \subseteq L(f) \Leftrightarrow \exists \text{ proof of } e \subseteq f.$$

A proof system for regular expressions

Example: [Das, Pous '17]
Cyclic proof system for inclusion of regular expressions:

$$\dfrac{\dfrac{\dfrac{}{\varepsilon \subseteq \varepsilon}\,(\mathsf{Ax})}{\varepsilon \subseteq a^*}\,(\text{*-right}_1) \qquad \dfrac{\dfrac{}{a \subseteq a}\,(\mathsf{Ax}) \qquad a^* \subseteq a^*}{a, a^* \subseteq a^*}\,(\text{*-right}_2)}{a^* \subseteq a^*}\,(\text{*-left})$$

Soundness and completeness [Das, Pous '17]

$$L(e) \subseteq L(f) \Leftrightarrow \exists \text{ proof of } e \subseteq f.$$

Here, we care about computational content.

This example: program whose type is $a^* \to a^*$:

```
let rec f l = match l with
                |[] -> []
                |a::q -> a::(f q)
```

Computing languages

Goal: Avoid transductions, start with languages.

- Regular expressions $e, f := a \in A \mid e.f \mid e + f \mid e^*$
- Boolean type bool (encoded by $\varepsilon + \varepsilon$)

Proof of $A^* \vdash \mathit{bool}$ $\Leftrightarrow$ Program of type $A^* \to \mathit{bool}$
$\Leftrightarrow$ Language $L \subseteq A^*$.

Computing languages

Goal: Avoid transductions, start with languages.

- Regular expressions $e, f := a \in A \mid e.f \mid e + f \mid e^*$
- Boolean type $bool$ (encoded by $\varepsilon + \varepsilon$)

$$\begin{aligned} \text{Proof of } A^* \vdash bool \quad &\Leftrightarrow \text{Program of type } A^* \to bool \\ &\Leftrightarrow \text{Language } L \subseteq A^*. \end{aligned}$$

Structural rules: basic data manipulation (erase, copy).

Computing languages

Goal: Avoid transductions, start with languages.

- Regular expressions $e, f := a \in A \mid e.f \mid e + f \mid e^*$
- Boolean type $bool$ (encoded by $\varepsilon + \varepsilon$)

$$\text{Proof of } A^* \vdash bool \quad \Leftrightarrow \text{Program of type } A^* \to bool$$
$$\Leftrightarrow \text{Language } L \subseteq A^*.$$

Structural rules: basic data manipulation (erase, copy).

On the proof side: reuse or ignore hypotheses (cf *linear logic*)

Simplified proof system

Expressions $e := A \mid A^*$

Lists $E, F = e_1, e_2, \ldots, e_n$ interpreted as tuples

Proof system:

Return true or false

$$\frac{}{\vdash bool}\ (true) \qquad \frac{}{\vdash bool}\ (false)$$

Pattern matchings

$$\frac{(E, F \vdash bool)_{\underline{a \in A}}}{E, \underline{A}, F \vdash bool}\ \text{(A)} \qquad \frac{E, F \vdash bool \qquad E, A, A^*, F \vdash bool}{E, \underline{A^*}, F \vdash bool}\ (*)$$

Erase, copy

$$\frac{E, F \vdash bool}{E, \underline{e}, F \vdash bool}\ \text{(weakening)} \qquad \frac{E, \underline{e, e}, F \vdash bool}{E, \underline{e}, F \vdash bool}\ \text{(contraction)}$$

Proofs as language acceptors

What are the languages computed by cyclic proofs ?

Example on alphabet $\{a, b\}$: The language b^*

$$\dfrac{\dfrac{}{\vdash bool}\,(\mathit{true}) \qquad \dfrac{\dfrac{\dfrac{}{\vdash bool}\,(\mathit{false})}{(\underline{A^*} \vdash bool)_a}\,(\mathsf{wkn}) \qquad (A^* \vdash bool)_b}{\underline{A}, A^* \vdash bool}\,(\mathsf{A})}{\underline{A^*} \vdash bool}\,(*)$$

Proofs as language acceptors

What are the languages computed by cyclic proofs ?

Example on alphabet $\{a, b\}$: The language b^*

$$\dfrac{\dfrac{}{\vdash bool}\;(true) \qquad \dfrac{\dfrac{\dfrac{}{\vdash bool}\;(false)}{(\underline{A^*} \vdash bool)_a}\;(\mathsf{wkn}) \qquad (A^* \vdash bool)_b}{\underline{A}, A^* \vdash bool}\;(\mathsf{A})}{\underline{A^*} \vdash bool}\;(*)$$

No contraction rule: *Affine* system.

Proofs as language acceptors

What are the languages computed by cyclic proofs ?

Example on alphabet $\{a, b\}$: The language b^*

$$\dfrac{\dfrac{}{\vdash bool}\,(\mathit{true}) \qquad \dfrac{\dfrac{\dfrac{}{\vdash bool}\,(\mathit{false})}{(\underline{A^*} \vdash bool)_a}\,(\text{wkn}) \qquad (A^* \vdash bool)_b}{\underline{A}, A^* \vdash bool}\,(\text{A})}{\underline{A^*} \vdash bool}\,(*)$$

No contraction rule: *Affine* system.

Lemma

The affine system captures exactly regular languages.

With contractions: what class of language?

Example on alphabet $\{a, b\}$: Language $\{a^n b^n \mid n \in \mathbb{N}\}$.

Intuition:

- Copy the input u_1 into u_2: $aaabbb \quad aaabbb$
- Erase leading a's in u_2: $aaabbb \quad bbb$
- Match each leading a in u_1 to a leading b in u_2: $bbb \quad \varepsilon$
- When u_2 becomes empty, verify that $u_1 \in b^*$.

With contractions: what class of language?

Example on alphabet $\{a, b\}$: Language $\{a^n b^n \mid n \in \mathbb{N}\}$.

Intuition:

- Copy the input u_1 into u_2: $aaabbb \quad aaabbb$
- Erase leading a's in u_2: $aaabbb \quad bbb$
- Match each leading a in u_1 to a leading b in u_2: $bbb \quad \varepsilon$
- When u_2 becomes empty, verify that $u_1 \in b^*$.

We can also recognize $a^n b^n c^n$ with the same technique.

With contractions: what class of language?

Example on alphabet $\{a, b\}$: Language $\{a^n b^n \mid n \in \mathbb{N}\}$.

Intuition:

- Copy the input u_1 into u_2: $aaabbb \quad aaabbb$
- Erase leading a's in u_2: $aaabbb \quad bbb$
- Match each leading a in u_1 to a leading b in u_2: $bbb \quad \varepsilon$
- When u_2 becomes empty, verify that $u_1 \in b^*$.

We can also recognize $a^n b^n c^n$ with the same technique.

Theorem

The proof system recognizes exactly languages in LOGSPACE.

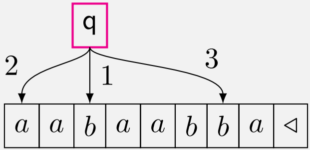
Proof technique: Design an equivalent automaton model.

A new automaton model

Jumping Multihead Automata

A JMA is an automaton with k reading heads.

Transitions: $Q \times (A \cup \{\triangleleft\})^k \to Q \times \{⏭, ◌, J_1, \ldots, J_k\}^k$



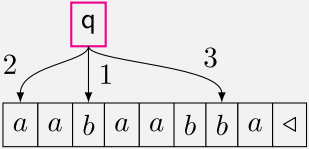
- ⏭ : advance one step
- ◌: stay in place
- J_i: jump to the position of head i

A new automaton model

Jumping Multihead Automata

A JMA is an automaton with k reading heads.

Transitions: $Q \times (A \cup \{\triangleleft\})^k \to Q \times \{\blacktriangleright\!\!|, ◌, J_1, \dots, J_k\}^k$

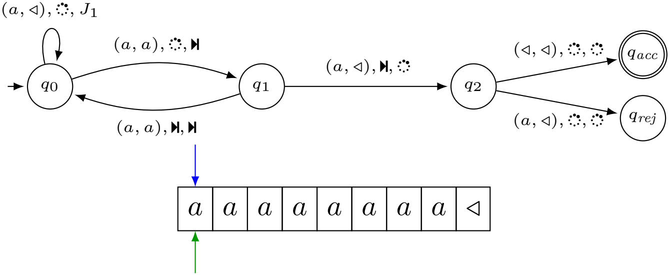


- $\blacktriangleright\!\!|$: advance one step
- ◌: stay in place
- J_i: jump to the position of head i

(optional: Syntactic criterion guaranteeing halting)

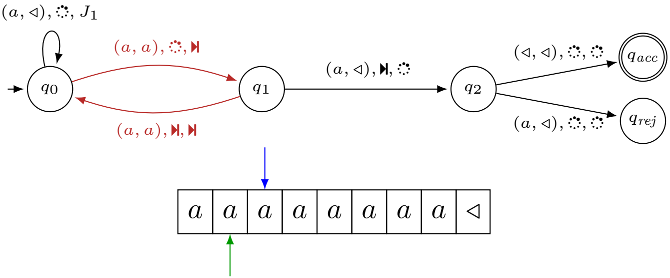
Example of JMA

Example: $\{a^{2^n} \mid n \in \mathbb{N}\}$ is accepted by a 2-head JMA.



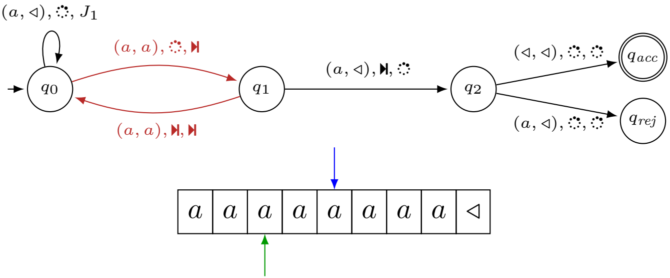
Example of JMA

<u>Example</u>: $\{a^{2^n} \mid n \in \mathbb{N}\}$ is accepted by a 2-head JMA.



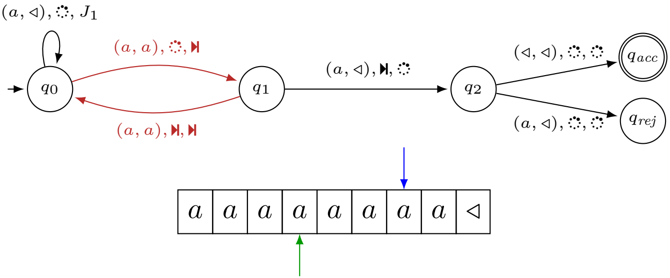
Example of JMA

Example: $\{a^{2^n} \mid n \in \mathbb{N}\}$ is accepted by a 2-head JMA.



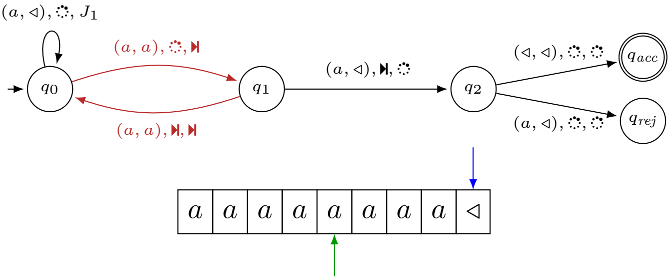
Example of JMA

Example: $\{a^{2^n} \mid n \in \mathbb{N}\}$ is accepted by a 2-head JMA.



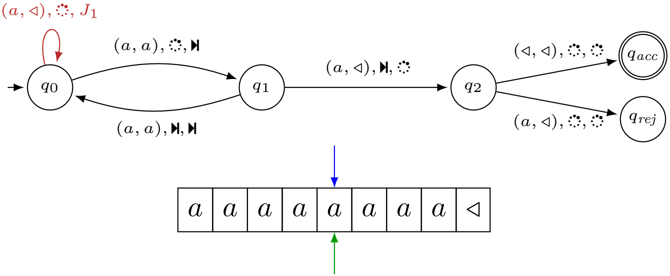
Example of JMA

Example: $\{a^{2^n} \mid n \in \mathbb{N}\}$ is accepted by a 2-head JMA.



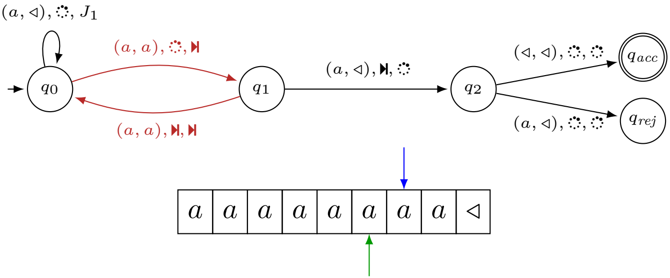
Example of JMA

Example: $\{a^{2^n} \mid n \in \mathbb{N}\}$ is accepted by a 2-head JMA.



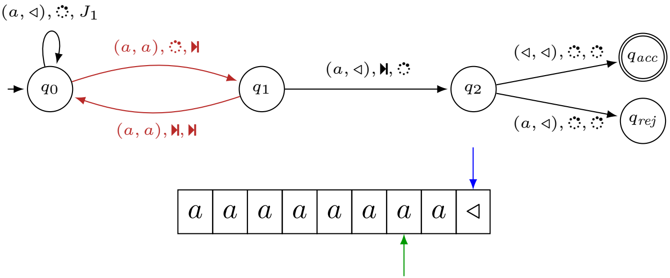
Example of JMA

Example: $\{a^{2^n} \mid n \in \mathbb{N}\}$ is accepted by a 2-head JMA.



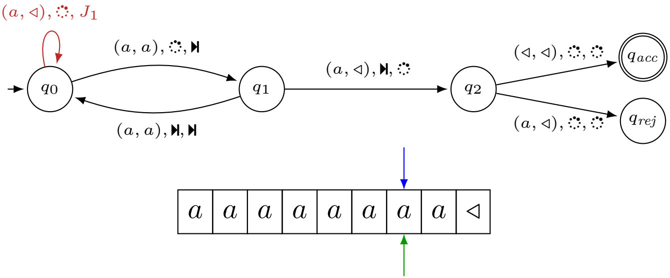
Example of JMA

Example: $\{a^{2^n} \mid n \in \mathbb{N}\}$ is accepted by a 2-head JMA.



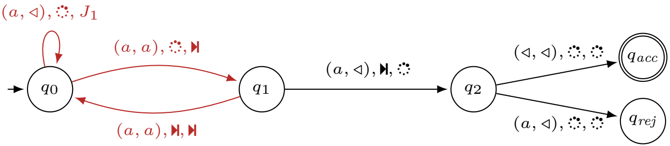
Example of JMA

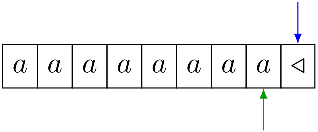
Example: $\{a^{2^n} \mid n \in \mathbb{N}\}$ is accepted by a 2-head JMA.



Example of JMA

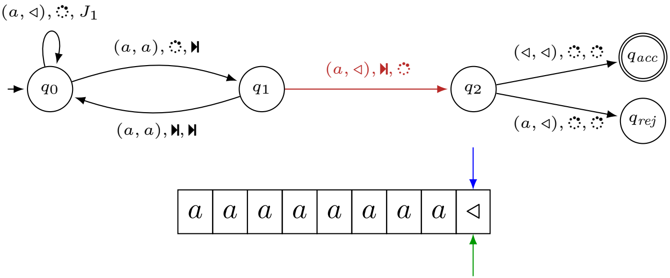
Example: $\{a^{2^n} \mid n \in \mathbb{N}\}$ is accepted by a 2-head JMA.





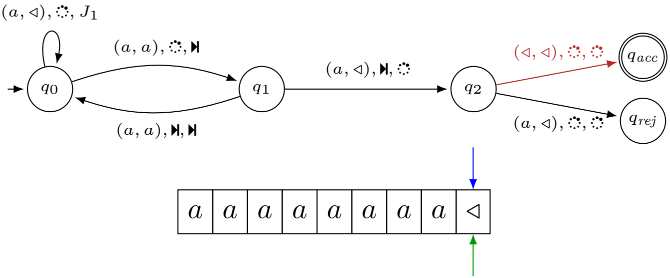
Example of JMA

Example: $\{a^{2^n} \mid n \in \mathbb{N}\}$ is accepted by a 2-head JMA.



Example of JMA

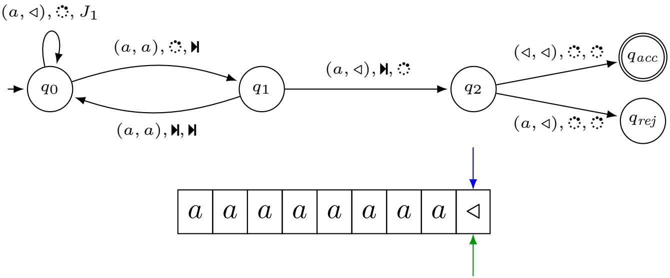
Example: $\{a^{2^n} \mid n \in \mathbb{N}\}$ is accepted by a 2-head JMA.



ACCEPT

Example of JMA

Example: $\{a^{2^n} \mid n \in \mathbb{N}\}$ is accepted by a 2-head JMA.



Theorem

Cyclic proofs and JMA recognize the same class of languages.

Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

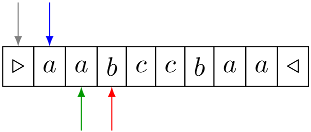
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $\mathit{Palindroms} = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

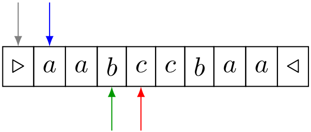
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

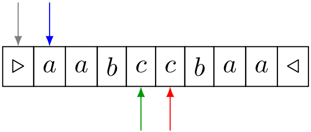
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

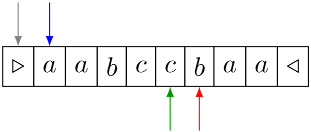
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

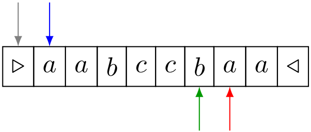
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $\mathit{Palindroms} = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ Logspace easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

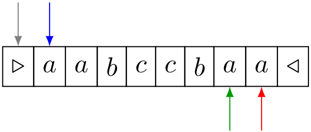
- No jump, but heads can move left or right.
- Characterizes Logspace.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

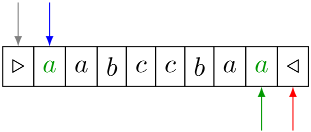
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

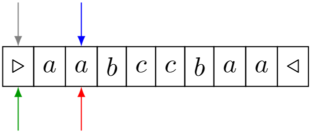
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

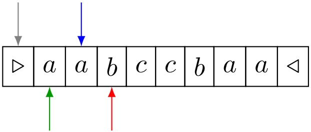
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

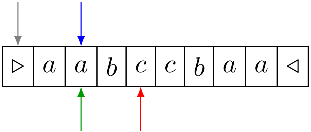
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

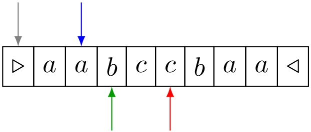
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

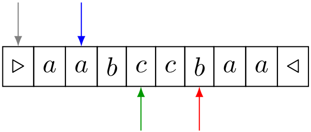
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $\mathit{Palindroms} = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

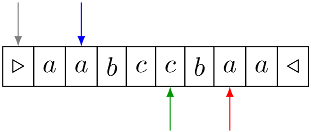
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

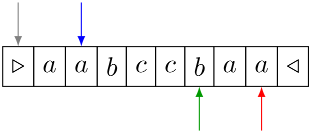
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.

$\triangleright$	a	a	b	c	c	b	a	a	$\triangleleft$

Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

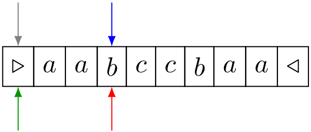
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

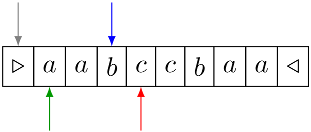
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

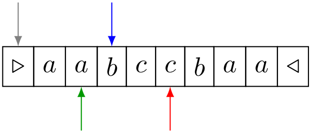
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

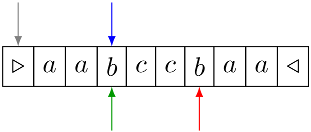
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

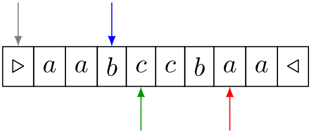
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

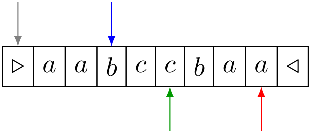
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

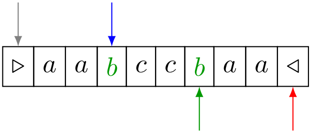
- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.



Expressive power of JMAs

JMAs $\subseteq$ LOGSPACE easy: remember the location of the k heads.

2MAs: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

- No jump, but heads can move left or right.
- Characterizes LOGSPACE.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: $Palindroms = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.

$\triangleright$	a	a	b	c	c	b	a	a	$\triangleleft$

Generalization of this idea $\Rightarrow$ Translation from 2MA to JMA.

Cut rule and integer functions

The cut rule:

$$\frac{E \vdash e \qquad e, F \vdash g}{E, F \vdash g}$$

- Corresponds to composition of programs.
- Fundamental in proof theory.
- Computation $\leftrightarrow$ cut elimination process.

Cut rule and integer functions

The cut rule:

$$\frac{E \vdash e \qquad e, F \vdash g}{E, F \vdash g}$$

- Corresponds to composition of programs.
- Fundamental in proof theory.
- Computation $\leftrightarrow$ cut elimination process.

We can now consider transductions: $A^* \to B^*$.
Focus on functions: $\mathbb{N}^k \to \mathbb{N}$ (unary alphabet).

Cut rule and integer functions

The cut rule:

$$\frac{E \vdash e \qquad e, F \vdash g}{E, F \vdash g}$$

- Corresponds to composition of programs.
- Fundamental in proof theory.
- Computation $\leftrightarrow$ cut elimination process.

We can now consider transductions: $A^* \to B^*$.
Focus on functions: $\mathbb{N}^k \to \mathbb{N}$ (unary alphabet).

Expressions (simplified): $e, f := 1 \mid e \cdot f \mid e + f \mid e^* \mid e \to f$.

Sequents: $(1^*)^k \vdash 1^*$: functions $\mathbb{N}^k \to \mathbb{N}$.

Cut rule and integer functions

The cut rule:

$$\frac{E \vdash e \qquad e, F \vdash g}{E, F \vdash g}$$

- Corresponds to composition of programs.
- Fundamental in proof theory.
- Computation $\leftrightarrow$ cut elimination process.

We can now consider transductions: $A^* \to B^*$.
Focus on functions: $\mathbb{N}^k \to \mathbb{N}$ (unary alphabet).

Expressions (simplified): $e, f := 1 \mid e \cdot f \mid e + f \mid e^* \mid e \to f$.

Sequents: $(1^*)^k \vdash 1^*$: functions $\mathbb{N}^k \to \mathbb{N}$.

Which functions $\mathbb{N}^k \to \mathbb{N}$ can the system with cuts compute ?

System T

As automata before, we want a computational framework to characterize the expressive power of our cyclic proof system.

System T:

- λ-calculus with explicit integer type,
- Explicit recursion operator on integers,
- Type system, typing derivations are finite trees.

System T

As automata before, we want a computational framework to characterize the expressive power of our cyclic proof system.

System T:

- λ-calculus with explicit integer type,
- Explicit recursion operator on integers,
- Type system, typing derivations are finite trees.

System T terms:

$$M, N ::= \quad x \mid 0 \mid s(M) \mid \lambda x.M \mid M\,N \mid \mathbf{Rec}(N, M_0, M_s(x,y))$$

(+constructors/destructors for pairs, lists)

$\mathbf{Rec}(N, M_0, M_s)$ returns $\begin{cases} M_0 & \text{if } N = 0 \\ M_s(N, \mathbf{Rec}(n, M_0, M_s)) & \text{if } N = s(n) \end{cases}$

System T

As automata before, we want a computational framework to characterize the expressive power of our cyclic proof system.

System T:

- λ-calculus with explicit integer type,
- Explicit recursion operator on integers,
- Type system, typing derivations are finite trees.

System T terms:

$$M, N ::= \quad x \mid 0 \mid s(M) \mid \lambda x.M \mid MN \mid \mathbf{Rec}(N, M_0, M_s(x,y))$$

(+constructors/destructors for pairs, lists)

$\mathbf{Rec}(N, M_0, M_s)$ returns $\begin{cases} M_0 & \text{if } N = 0 \\ M_s(N, \mathbf{Rec}(n, M_0, M_s)) & \text{if } N = s(n) \end{cases}$

Example: Addition $a + b$: $\lambda ab.\mathbf{Rec}(b, a, s(y))$

Results

System $\mathsf{T}_{\mathsf{aff}}$: Affine version of System T, data cannot be duplicated.
Example: $\lambda f x. f(f(x))$ is not typable in $\mathsf{T}_{\mathsf{aff}}$.

Results

System $\mathsf{T}_{\mathsf{aff}}$: Affine version of System T, data cannot be duplicated.
Example: $\lambda fx.f(f(x))$ is not typable in $\mathsf{T}_{\mathsf{aff}}$.

Theorem

Affine Cyclic proofs	$\iff$	*System T_{aff}*	$\iff$	*Prim. rec.*
Cyclic proofs	$\iff$	*System T*	$\iff$	*Peano*

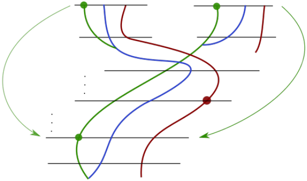
Results

System $\mathsf{T}_{\mathsf{aff}}$: Affine version of System T, data cannot be duplicated.
Example: $\lambda f x. f(f(x))$ is not typable in $\mathsf{T}_{\mathsf{aff}}$.

Theorem

Affine Cyclic proofs $\iff$ *System T_{aff}* $\iff$ *Prim. rec.*

Cyclic proofs $\iff$ *System T* $\iff$ *Peano*



$$\lambda ab.\mathbf{Rec}(b, a, s(y))$$

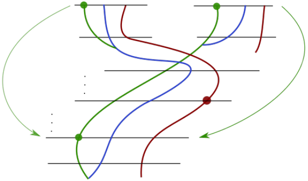
Results

System $\mathsf{T}_{\mathsf{aff}}$: Affine version of System T, data cannot be duplicated.
Example: $\lambda f x. f(f(x))$ is not typable in $\mathsf{T}_{\mathsf{aff}}$.

Theorem

Affine Cyclic proofs $\iff$ *System T_{aff}* $\iff$ *Prim. rec.*

Cyclic proofs $\iff$ *System T* $\iff$ *Peano*



Easy

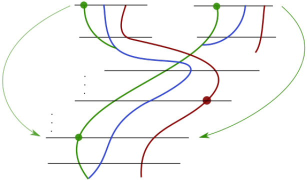
$$\lambda ab.\mathbf{Rec}(b, a, s(y))$$

Results

System $\mathbf{T}_{\mathbf{aff}}$: Affine version of System T, data cannot be duplicated.
Example: $\lambda f x. f(f(x))$ is not typable in $\mathsf{T}_{\mathsf{aff}}$.

Theorem

Affine Cyclic proofs	$\Longleftrightarrow$	*System T_{aff}*	$\Longleftrightarrow$	*Prim. rec.*
Cyclic proofs	$\Longleftrightarrow$	*System T*	$\Longleftrightarrow$	*Peano*



Easy

$$\lambda ab.\mathbf{Rec}(b, a, s(y))$$

Hard

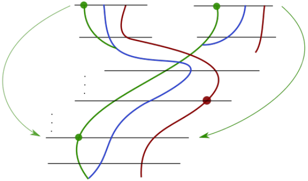
Results

System $\mathbf{T_{aff}}$: Affine version of System T, data cannot be duplicated.
Example: $\lambda f x. f(f(x))$ is not typable in $\mathsf{T}_{\mathsf{aff}}$.

Theorem

Affine Cyclic proofs $\iff$ *System T_{aff}* $\iff$ *Prim. rec.*

Cyclic proofs $\iff$ *System T* $\iff$ *Peano*



Easy

$$\lambda ab.\mathbf{Rec}(b, a, s(y))$$

Hard

Open problems in proof theory: infinite descent versus induction.

Proof schemes

Affine proofs $\to$ T_{aff}:

- normal form for proofs, with explicit hierarchy of cycles,
- inductively build T_{aff} terms.

Proof schemes

Affine proofs $\to$ $\mathsf{T}_{\mathsf{aff}}$:

- normal form for proofs, with explicit hierarchy of cycles,
- inductively build $\mathsf{T}_{\mathsf{aff}}$ terms.

Affine proofs $\to$ Prim. rec.:

- Stronger normal form through $\mathsf{T}_{\mathsf{aff}}$,
- RCA_0: constructive fragment of 2^{nd}-order arithmetic,
- $\forall$ affine cyclic proof, prove in RCA_0 that its computation terminates,
- From reverse maths: $\mathrm{RCA}_0 \leftrightarrow$ Prim. rec.

Proof schemes

Affine proofs $\to$ $\mathsf{T}_{\mathsf{aff}}$:

- normal form for proofs, with explicit hierarchy of cycles,
- inductively build $\mathsf{T}_{\mathsf{aff}}$ terms.

Affine proofs $\to$ Prim. rec.:

- Stronger normal form through $\mathsf{T}_{\mathsf{aff}}$,
- RCA_0: constructive fragment of 2^{nd}-order arithmetic,
- $\forall$ affine cyclic proof, prove in RCA_0 that its computation terminates,
- From reverse maths: $\mathsf{RCA}_0 \leftrightarrow$ Prim. rec.

Proofs $\to$ System T:

- ACA_0: RCA_0 + König's lemma,
- $\forall$ cyclic proof , prove in ACA_0 that its computation terminates,
- Conservativity result: $\mathsf{ACA}_0 \leftrightarrow$ Peano for integer functions,
- Classic result: Peano $\leftrightarrow$ System T.

Conclusion

Open problems:

- Avoid the "blackbox" of reverse maths.
- Use of reverse maths $\rightarrow$ some results do not lift to transductions.
- Generalize the normal form with hierarchy of cycles to other cyclic proof systems.

Conclusion

Open problems:

- Avoid the "blackbox" of reverse maths.
- Use of reverse maths $\rightarrow$ some results do not lift to transductions.
- Generalize the normal form with hierarchy of cycles to other cyclic proof systems.

Thank you for your attention !