Computational content of circular proof systems.

Denis Kuperberg     Laureline Pinault     Damien Pous

LIP, ENS Lyon

Warsaw Automata Seminar
Wednesday 13th May 2020
1  Context

2  Computing languages

3  Computing functions
Curry-Howard correspondence

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

Example: The identity program is a proof of $p \rightarrow p$. 
Curry-Howard correspondence

Proof of formula $\varphi \iff$ Program of type $\varphi$

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

Deduction Rule

$\quad A \quad B$

$\downarrow$

implies

$C$
Curry-Howard correspondence

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

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

Deduction Rule

$$\text{implies} \quad \frac{A}{C} \quad \frac{B}{C}$$

Program instruction

$$\text{calls}$$
Curry-Howard correspondence

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

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

Deduction Rule

$\begin{array}{c}
A \\
\Rightarrow \\
C \\
\end{array} \quad B$

Program instruction

$\downarrow$ calls

Correspondence well-understood for usual proof systems

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

[...] ... $\leftrightarrow$ ...
Curry-Howard correspondence

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

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

Deduction Rule

implies

Program instruction

Correspondence well-understood for usual proof systems

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

This work: Study the computational content of cyclic proofs.
Cyclic Proofs

Usual proofs:

\[
\frac{A}{A} \quad \frac{B}{A} \quad \frac{C}{C}
\]

Validity conditions: Cycles must contain particular rules. As programs: recursive calls must be done on smaller arguments. → guarantees termination.
Cyclic Proofs

Usual proofs:

\[
\text{Axiom}_1 \quad \frac{A}{B} \quad \text{Axiom}_2 \quad \frac{A}{C} \quad \text{Axiom}_3
\]

Cyclic Proofs:

\[
\begin{array}{c}
\text{Axiom} \\
\frac{A}{B} \\
\frac{D}{C}
\end{array}
\]

Validity conditions: Cycles must contain particular rules.

As programs: recursive calls must be done on smaller arguments. → guarantees termination.
Cyclic Proofs

Usual proofs:

\[
\frac{A}{B} \quad \frac{A}{C} \quad \frac{B}{D} \quad \frac{C}{D}
\]

Cyclic Proofs:

\[
\frac{A}{A} \quad \frac{B}{D} \quad \frac{C}{D}
\]

Validity conditions: Cycles must contain particular rules.
Cyclic Proofs

Usual proofs:

\[
\frac{A}{A} ~ \frac{B}{B} \quad \frac{A}{A} ~ \frac{C}{C} \quad \frac{D}{D}
\]

Cyclic Proofs:

\[
\frac{A}{A} \quad \frac{D}{D}
\]

Validity conditions: Cycles must contain particular rules.

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

Usual proofs:

\[
\begin{align*}
\text{Axiom}_1 & \quad A \\
\text{Axiom}_2 & \quad A \\
\text{Axiom}_3 & \quad C
\end{align*}
\]

Cyclic Proofs:

\[
\begin{align*}
\text{Axiom} & \quad A \\
D & \quad D
\end{align*}
\]

Validity conditions: Cycles must contain particular rules.

As programs: recursive calls must be done on smaller arguments.
→ guarantees termination.
A proof system for regular expressions

**Example:** [Das, Pous ’17]

Cyclic proof system for inclusion of regular expressions:

\[
\begin{align*}
\varepsilon \subseteq \varepsilon & \quad \text{(Ax)} \\
\varepsilon \subseteq a^* & \quad \text{(*-right)} \quad \text{(
ax)} \\
a \subseteq a & \quad \text{(Ax)} \\
 a, a^* \subseteq a^* & \quad \text{(*-right)} \\
 a^* \subseteq a^* & \quad \text{(*-left)}
\end{align*}
\]

Soundness and completeness [Das, Pous ’17]

Let \( L(e) \subseteq L(f) \iff \exists \) proof of \( e \subseteq f \).

Here, we care about computational content.

This example: program whose type is \( a^* \rightarrow a^* \):

\[
\text{let rec } f \text{ l } = \begin{cases} \\
|\emptyset \rightarrow \emptyset \\
|a::q \rightarrow a::(f q)
\end{cases}
\]

A proof system for regular expressions

**Example**: [Das, Pous ’17]

Cyclic proof system for inclusion of regular expressions:

\[
\begin{align*}
\varepsilon \subseteq \varepsilon & \quad \text{(Ax)} \\
\varepsilon \subseteq a^* & \quad \text{(*-right}_1) \\
a \subseteq a & \quad \text{(Ax)} \\
a^* \subseteq a^* & \quad \text{(*-right}_2) \\
a, a^* \subseteq a^* & \quad \text{(*-left)} \\
a^* \subseteq a^* &
\end{align*}
\]

Soundness and completeness [Das, Pous ’17]

\[L(e) \subseteq L(f) \iff \exists \text{ proof of } e \subseteq f.\]
A proof system for regular expressions

**Example:** [Das, Pous ’17]

Cyclic proof system for inclusion of regular expressions:

\[
\begin{align*}
\varepsilon \subseteq \varepsilon & \quad \text{(Ax)} \\
\varepsilon \subseteq a^* & \quad \text{(*-right)} \\
a \subseteq a & \quad \text{(Ax)} \\
a, a^* \subseteq a^* & \quad \text{(*-right)} \\
a^* \subseteq a^* & \quad \text{(*-left)}
\end{align*}
\]

Soundness and completeness [Das, Pous ’17]

\[L(e) \subseteq L(f) \iff \exists \text{ proof of } e \subseteq f.\]

Here, we care about computational content.

**This example:** program whose type is \(a^* \rightarrow a^*\):

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

5/19
1. Context

2. Computing languages

3. Computing functions
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 $A^* \vdash bool$ $\iff$ Program of type $A^* \rightarrow bool$

$\iff$ 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 \( \text{bool} \) (encoded by \( \varepsilon + \varepsilon \))

\[
\text{Proof of } A^* \vdash \text{bool} \iff \text{Program of type } A^* \rightarrow \text{bool} \\
\iff \text{Language } L \subseteq A^*.
\]

*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 \( \varepsilon + \varepsilon \))

**Proof** of \( A^* \vdash bool \iff \text{Program of type } A^* \rightarrow bool \iff \text{Language } L \subseteq A^*. \)

*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:**

\[
\begin{align*}
\frac{}{\vdash \text{bool} \quad \text{(true)}} \\
\frac{}{\vdash \text{bool} \quad \text{(false)}}
\end{align*}
\]

**Pattern matchings**

\[
\begin{align*}
\frac{(E, F \vdash \text{bool})_{a \in A}}{E, A, F \vdash \text{bool} \quad \text{(A)}} & \quad & \frac{E, F \vdash \text{bool} \quad E, A, A^*, F \vdash \text{bool}}{E, A^*, F \vdash \text{bool} \quad \text{(*)}}
\end{align*}
\]

**Erase, copy**

\[
\begin{align*}
\frac{E, F \vdash \text{bool}}{E, e, F \vdash \text{bool} \quad \text{(weakening)}} & \quad & \frac{E, e, e, F \vdash \text{bool}}{E, e, F \vdash \text{bool} \quad \text{(contraction)}}
\end{align*}
\]
Proofs as language acceptors

What are the languages computed by cyclic proofs?

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

\[
\begin{align*}
\vdash \text{bool} & \quad \text{(false)} \\
(A^* \vdash \text{bool})_a & \quad \text{wkn} \\
(A^* \vdash \text{bool})_b & \quad \text{(A)} \\
\vdash \text{true} & \quad \text{(true)} \\
A^*, A^* \vdash \text{bool} & \quad \text{(*)} \\
A^* \vdash \text{bool} & \quad \text{(*)}
\end{align*}
\]
Proofs as language acceptors

What are the languages computed by cyclic proofs?

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

\[
\begin{align*}
\vdash \text{false} \quad & \quad \text{(false)} \\
(A^* \vdash \text{bool})_a \quad & \quad \text{(wkn)} \\
A, A^* \vdash \text{bool} \quad & \quad \text{(A)} \\
A^* \vdash \text{bool} \quad & \quad \text{(*)}
\end{align*}
\]

No contraction rule: Affine system.
Proofs as language acceptors

What are the languages computed by cyclic proofs?

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

\[
\begin{align*}
\vdash \text{true} & \quad (\text{true}) \\
(A^* \vdash \text{bool})_a & \quad (\text{wkn}) \\
(A^* \vdash \text{bool})_b & \quad (A) \\
& \quad A, A^* \vdash \text{bool} \quad (\ast) \\
& \quad A^* \vdash \text{bool} \\
\end{align*}
\]

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 \( u_2 \): \( aaabbb \quad aaabbbb \)
- 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 \(u_2\): \(aaabbb\) \(aabb\bb\)
- Erase leading \(a\)'s in \(u_2\): \(aaabbb\) \(b\bb\)
- Match each leading \(a\) in \(u_1\) to a leading \(b\) in \(u_2\): \(b\bb\) \(\varepsilon\)
- 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 \( u_2 \): \( aaabbb \) \( aaabbb \)
- Erase leading \( a \)'s in \( u_2 \): \( aaabbb \) \( bbb \)
- Match each leading \( a \) in \( u_1 \) to a leading \( b \) in \( u_2 \): \( bbb \) \( \varepsilon \)
- 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: \( Q \times (A \cup \{\leftarrow\})^k \rightarrow Q \times \{\leftarrow, \rightarrow, J_1, \ldots, J_k\}^k \)

- ▶️ : 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:**

$$Q \times (A \cup \{\text{◁}\})^k \rightarrow Q \times \{\text{▷}, \text{♠}, J_1, \ldots, J_k\}^k$$

- $\text{▷}$: advance one step
- $\text{♠}$: 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.

\[\begin{align*}
(a, ◁), \emptyset, J_1 \\
q_0 \\
→ (a, a), \emptyset, \wedge \\
q_1 \\
→ (a, ◁), \wedge, \emptyset \\
q_2 \\
→ (a, ◁), \emptyset, \emptyset \\
q_{acc} \\
→ (a, ◁), \emptyset, \emptyset \\
q_{rej}
\end{align*}\]
Example of JMA

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

Example: \( \{a^{2n} \mid n \in \mathbb{N}\} \) 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.

\[
\begin{aligned}
&\text{Example: } \{a^{2^n} \mid n \in \mathbb{N}\} \text{ is accepted by a 2-head JMA.}
\end{aligned}
\]
Example of JMA

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

\[
\begin{align*}
(a, a), &\quad \triangleright, J_1 \\
(a, a), &\quad \triangledown, J_1 \\
(a, \triangle), &\quad \triangledown, J_1 \\
(a, \triangle), &\quad \triangledown, J_1 \\
\end{align*}
\]
**Example of JMA**

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

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

\[
\begin{align*}
(a, a), \Diamond, J_1 \\
(q_0, a, a, H, H, H) &\rightarrow (q_1, a, a, H, H, H) \\
(q_1, a, a, H, H, H) &\rightarrow (q_2, a, a, H, H, H) \\
(q_2, a, a, H, H, H) &\rightarrow (q_{acc}, a, a, H, H, H) \\
(q_0, a, a, H, H, H) &\rightarrow (q_{rej}, a, a, H, H, H)
\end{align*}
\]
**Example of JMA**

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

\[
\begin{align*}
q_0 &\xrightarrow{(a, a), \text{L}, J_1} q_1 \\
q_1 &\xrightarrow{(a, a), \text{L}, \text{L}} q_1 \\
q_1 &\xrightarrow{(a, a), \text{R}, \text{R}} q_1 \\
q_1 &\xrightarrow{(a, a), \text{L}, \text{L}} q_2 \\
q_2 &\xrightarrow{(a, \lambda), \text{R}, \text{R}} q_{\text{acc}} \\
q_2 &\xrightarrow{(a, \lambda), \text{L}, \text{L}} q_{\text{acc}} \\
q_1 &\xrightarrow{(a, \lambda), \text{R}, \text{R}} q_{\text{rej}} \\
q_1 &\xrightarrow{(a, \lambda), \text{L}, \text{L}} q_{\text{rej}} \\
\end{align*}
\]
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.

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

```plaintext
(a, ◁), ◁, J_1
q_0

(a, a), ◁, □
q_1

(a, a), □, □

(a, a), □, □

(a, ◁), □, ◁
q_2

(q_0, q_1, q_2, q_{\text{acc}}, q_{\text{rej}})

\[
\begin{array}{cccccc}
a & a & a & a & a & a \\
\end{array}
\]
```

Theorem: Cyclic proofs and JMA recognize the same class of languages.
Example of JMA

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

\[
(a, a), \triangleright, J_1
\]

\[
(a, a), \triangleright, \triangleright
\]

\[
(a, a), \triangleright, \triangleright
\]

\[
(a, a), \triangleright, \triangleright
\]

\[
(a, a), \triangleright, \triangleright
\]

\[
(a, a), \triangleright, \triangleright
\]

\[
(q_{acc})
\]

\[
(q_{rej})
\]

\[
\begin{array}{ccccccccc}
  \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \text{a} & \triangleright \\
\end{array}
\]

ACCEPT
Example of JMA

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

Theorem

*Cyclic proofs and JMA recognize the same class of languages.*
Expressive power of JMAs

\[ \text{JMAs} \subseteq \text{Logspace easy}: \text{remember the location of the } k \text{ heads.} \]
Expressive power of JMAs

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

$\text{2MAs}$: 2-way Multihead Automata [Holzer, Kutrib, Malcher ’08]:
- No jump, but heads can move left or right.
- Characterizes $\text{Logspace}$. 

Example: Palindroms $\{u \in \Sigma^* | u = u^R\}$ is accepted by a JMA.
Expressive power of JMAs

**JMAs \( \subseteq \text{Logspace} \) 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 \( \text{Logspace} \).

**Theorem**

*Any 2MA can be simulated by a JMA*
Expressive power of JMAs

**JMAs** $\subseteq \text{Logspace}$ 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 $\{u \in \Sigma^* | u = u^R\}$ is accepted by a JMA.
Expressive power of JMAs

**JMAs ⊆ 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:** $\text{Palindroms} = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.
Expressive power of JMA

\( \text{JMA} \subseteq \text{Logspace} \) 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 \( \text{Logspace} \).

**Theorem**

*Any 2MA can be simulated by a JMA*

Difficulty: simulate a left move of some head.

**Example:** \( \text{Palindroms} = \{u \in \Sigma^* \mid u = u^R\} \) is accepted by a JMA.
Expressive power of JMAs

**JMAs \( \subseteq \text{Logspace} \) 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 \( \text{Logspace} \).

**Theorem**

*Any 2MA can be simulated by a JMA*

Difficulty: simulate a left move of some head.

**Example:** \( \text{Palindroms} = \{ u \in \Sigma^* \mid u = u^R \} \) is accepted by a JMA.
Expressive power of JMA$	ext{s}$

$\text{JMA\textsubscript{s}} \subseteq \text{LOGSPACE easy}$: remember the location of the $k$ heads.

$\text{2MA}\text{s}$: 2-way Multihead Automata [Holzer, Kutrib, Malcher ’08]:

- No jump, but heads can move left or right.
- Characterizes $\text{LOGSPACE}$.

**Theorem**

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

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

JMAs $\subseteq \text{Logspace}$ 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 \text{Logspace}.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

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

\[ \begin{array}{cccccccc}
\uparrow & a & a & b & c & c & b & a & a \\
\end{array} \]
Expressive power of JMAs

**JMAs ⊆ 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:** $\text{Palindroms} = \{u \in \Sigma^* \mid u = u^R\}$ is accepted by a JMA.
Expressive power of JMA

\( \text{JMA} \subseteq \text{Logspace} \) easy: remember the location of the \( k \) heads.

\( 2\text{MAs} \): 2-way Multihead Automata [Holzer, Kutrib, Malcher ’08]:
- No jump, but heads can move left or right.
- Characterizes \( \text{Logspace} \).

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

Example: \( \text{Palindroms} = \{ u \in \Sigma^* \mid u = u^R \} \) is accepted by a JMA.
Expressive power of JMAs

\[ \text{JMAs} \subseteq \text{Logspace} \text{ easy: remember the location of the } k \text{ heads.} \]

\textbf{2MAs:} 2-way Multihead Automata [Holzer, Kutrib, Malcher ’08]:

- No jump, but heads can move left or right.
- Characterizes \text{Logspace}.

\textbf{Theorem}

\underline{Any 2MA can be simulated by a JMA}

Difficulty: simulate a left move of some head.

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

[Diagram of a JMA accepting a palindrom with arrows indicating head movements]
Expressive power of JMAs

\textbf{JMAs} \subseteq \textsc{Logspace} easy: remember the location of the $k$ heads.

\textbf{2MAs}: 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:

- No jump, but heads can move left or right.
- Characterizes \textsc{Logspace}.

\textbf{Theorem}

\textit{Any 2MA can be simulated by a JMA}

Difficulty: simulate a left move of some head.

\textbf{Example}: \textit{Palindromes} = \{u \in \Sigma^* \mid u = u^R\} is accepted by a JMA.

\begin{center}
\begin{tikzpicture}

\node (a) at (0,0) {\textbf{a}};
\node (b) at (1,0) {\textbf{a}};
\node (c) at (2,0) {\textbf{b}};
\node (d) at (3,0) {\textbf{c}};
\node (e) at (4,0) {\textbf{c}};
\node (f) at (5,0) {\textbf{b}};
\node (g) at (6,0) {\textbf{a}};
\node (h) at (7,0) {\textbf{a}};
\node (i) at (8,0) {\textbf{\triangleright}};

\draw[->,red] (a) -- (b);
\draw[->,green] (b) -- (c);
\end{tikzpicture}
\end{center}
Expressive power of JMAs

**JMAs \( \subseteq \text{Logspace} \) 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 \( \text{Logspace} \).

**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 $\subseteq \text{Logspace}$ 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 $\text{Logspace}$.

**Theorem**

*Any 2MA can be simulated by a JMA*

Difficulty: simulate a left move of some head.

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

\[
\begin{array}{ccccccc}
\uparrow & a & a & b & c & c & b & a & a & \downarrow \\
\end{array}
\]
Expressive power of JMA\textsc{s}

\textbf{JMAs} \subseteq \textsc{Logspace} easy: remember the location of the $k$ heads.

\textbf{2MAs}: 2-way Multihead Automata [Holzer, Kutrib, Malcher ’08]:

\begin{itemize}
  \item No jump, but heads can move left or right.
  \item Characterizes \textsc{Logspace}.
\end{itemize}

\textbf{Theorem}

\textit{Any 2MA can be simulated by a JMA}

Difficulty: simulate a left move of some head.

\textbf{Example:}\ \textit{Palindroms} = \{u \in \Sigma^* \mid u = u^R\} is accepted by a JMA.
Expressive power of JMAs

\[ \text{JMAs} \subseteq \text{Logspace easy}: \text{remember the location of the } k \text{ heads.} \]

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: \( \text{Palindroms} = \{ u \in \Sigma^* \mid u = u^R \} \) is accepted by a JMA.
Expressive power of JMAs

\[ \text{JMAs} \subseteq \text{LOGSPACE} \text{ easy: remember the location of the } k \text{ heads.} \]

**2MAs:** 2-way Multihead Automata [Holzer, Kutrib, Malcher '08]:
- No jump, but heads can move left or right.
- Characterizes \text{LOGSPACE}.

**Theorem**

*Any 2MA can be simulated by a JMA*

Difficulty: simulate a left move of some head.

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

![Diagram](attachment:image.png)
Expressive power of JMAs

\[\text{JMAs} \subseteq \text{Logspace easy}: \text{remember the location of the } k \text{ heads.}\]

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

- No jump, but heads can move left or right.
- Characterizes \text{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.

\[
\begin{array}{cccccccc}
\triangleright & a & a & b & c & c & b & a & a \\
\end{array}
\]

\[
\begin{array}{cccccccc}
\swarrow & a & a & b & c & c & b & a & a \\
\end{array}
\]

\[
\begin{array}{cccccccc}
\nwarrow & a & a & b & c & c & b & a & a \\
\end{array}
\]
Expressive power of JMAs

**JMAs \( \subseteq \text{Logspace} \) 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 \( \text{Logspace} \).

**Theorem**

*Any 2MA can be simulated by a JMA*

**Difficulty:** simulate a left move of some head.

**Example:** \( \text{Palindroms} = \{ u \in \Sigma^* \mid u = u^R \} \) 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 $\subseteq \text{LOGSPACE easy}$: 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 $\text{LOGSPACE}$.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

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

\[ \begin{array}{cccccccc}
\triangleright & a & a & b & c & c & b & a & a & \triangleleft \\
\end{array} \]
Expressive power of JMAs

**JMAs ⊆ \text{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 \text{Logspace}.

**Theorem**

*Any 2MA can be simulated by a JMA*

Difficulty: simulate a left move of some head.

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

JMAs $\subseteq \text{Logspace}$ 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 $\text{Logspace}$.

Theorem

Any 2MA can be simulated by a JMA

Difficulty: simulate a left move of some head.

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

**JMAs \( \subseteq \text{Logspace} \) 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 \text{Logspace}.

**Theorem**

*Any 2MA can be simulated by a JMA*

Difficulty: simulate a left move of some head.

**Example:** \( \text{Palindroms} = \{ u \in \Sigma^* \mid u = u^R \} \) is accepted by a JMA.
Expressive power of JMA\(s\)

\(\text{JMA}\(s\) \subseteq \text{Logspace} \text{ easy}:\) remember the location of the \(k\) heads.

**2MA\(s\):** 2-way Multihead Automata [Holzer, Kutrib, Malcher ’08]:

▶ No jump, but heads can move left or right.

▶ Characterizes \text{Logspace}.

**Theorem**

*Any 2MA can be simulated by a JMA*

Difficulty: simulate a left move of some head.

**Example:** \(\text{Palindroms} = \{ u \in \Sigma^* \mid u = u^R \}\) is accepted by a JMA.
Expressive power of JMAs

\( \text{JMAs} \subseteq \text{LOGSPACE} \) 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 \( \text{LOGSPACE} \).

**Theorem**

*Any 2MA can be simulated by a JMA*

Difficulty: simulate a left move of some head.

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

\[
\begin{array}{cccccccc}
\uparrow & a & a & b & c & c & b & a & a \\
\end{array}
\]

Generalization of this idea \( \Rightarrow \) Translation from 2MA to JMA.
1 Context

2 Computing languages

3 Computing functions
Cut rule and integer functions

The cut rule:

\[
\frac{E \vdash e \quad e, F \vdash g}{E, F \vdash g}
\]

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

We can now consider transductions: $A \ast \rightarrow B \ast$. Focus on functions: $N^k \rightarrow N$ (unary alphabet). Expressions (simplified): $e, f := 1 \mid e \cdot f \mid e + f \mid e^* \mid e \rightarrow f$. Sequents: $(1^* \ast)^k \vdash 1^* \ast$: functions $N^k \rightarrow N$. Which functions $N^k \rightarrow N$ can the system with cuts compute?
Cut rule and integer functions

The cut rule:

\[
\frac{E \vdash e \quad e, F \vdash g}{E, F \vdash g}
\]

- Corresponds to composition of programs.
- Fundamental in proof theory.
- Computation $\iff$ 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:

\[
\frac{E \vdash e \quad e, F \vdash g}{E, F \vdash g}
\]

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

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

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

**Sequents**: $(1^*)^k \vdash 1^*$: functions $\mathbb{N}^k \rightarrow \mathbb{N}$. 
Cut rule and integer functions

The cut rule:

\[
E \vdash e \quad e, F \vdash g
\]

\[
E, F \vdash g
\]

► Corresponds to composition of programs.
► Fundamental in proof theory.
► Computation ↔ 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.

Example: Addition $a + b$: 

$$\lambda a b. \text{Rec}(b, a, s(y))$$
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:

\[ M, N ::= x \mid 0 \mid s(M) \mid \lambda x.M \mid MN \mid \text{Rec}(N, M_0, M_s(x, y)) \]

(+constructors/destructors for pairs, lists)

\[ \text{Rec}(N, M_0, M_s) \text{ returns } \begin{cases} M_0 & \text{if } N = 0 \\ M_s(N, \text{Rec}(n, M_0, M_s)) & \text{if } N = s(n) \end{cases} \]
System T

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

System T:

▶ \( \lambda \)-calculus with explicit integer type,
▶ Explicit recursion operator on integers,
▶ Type system, typing derivations are finite trees.

System T terms:

\[
M, N ::= x \mid 0 \mid s(M) \mid \lambda x.M \mid MN \mid \text{Rec}(N, M_0, M_s(x, y)) \\
\text{(constructors/destructors for pairs, lists)}
\]

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

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

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

Theorem

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

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

λab. Rec(b,a,s(y))

Easy

Hard

Open problems in proof theory: infinite descent versus induction.
Results

**System $T_{\text{aff}}$:** Affine version of System T, data cannot be duplicated.

**Example:** $\lambda f \cdot x. f(f(x))$ is not typable in $T_{\text{aff}}$.

---

<table>
<thead>
<tr>
<th>Theorem</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Affine Cyclic proofs</strong></td>
<td>$\iff$</td>
<td><strong>System $T_{\text{aff}}$</strong></td>
<td>$\iff$</td>
</tr>
<tr>
<td><strong>Cyclic proofs</strong></td>
<td>$\iff$</td>
<td><strong>System $T$</strong></td>
<td>$\iff$</td>
</tr>
</tbody>
</table>
**Results**

**System** $\mathbf{T}_{aff}$: Affine version of System $\mathbf{T}$, data cannot be duplicated.

**Example**: $\lambda f. x. f(f(x))$ is not typable in $\mathbf{T}_{aff}$.

<table>
<thead>
<tr>
<th>Affine Cyclic proofs</th>
<th>$\iff$</th>
<th>System $\mathbf{T}_{aff}$</th>
<th>$\iff$</th>
<th>Prim. rec.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cyclic proofs</strong></td>
<td>$\iff$</td>
<td>System $\mathbf{T}$</td>
<td>$\iff$</td>
<td>Peano</td>
</tr>
</tbody>
</table>

\[
\lambda ab. \text{Rec}(b, a, s(y))
\]
Results

**System $T_{\text{aff}}$:** Affine version of System T, data cannot be duplicated.

**Example:** $\lambda f . x . f(f(x))$ is not typable in $T_{\text{aff}}$.

<table>
<thead>
<tr>
<th>Theorem</th>
<th>Affine Cyclic proofs</th>
<th>$\iff$</th>
<th>System $T_{\text{aff}}$</th>
<th>$\iff$</th>
<th>Prim. rec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclic proofs</td>
<td>$\iff$ System $T_{\text{aff}}$</td>
<td>$\iff$</td>
<td>System $T$</td>
<td>$\iff$</td>
<td>Peano</td>
</tr>
</tbody>
</table>

Easy

$\lambda a b . \text{Rec}(b, a, s(y))$
Results

**System T\textsubscript{aff}:** Affine version of System T, data cannot be duplicated.

**Example:** \( \lambda f x. f(f(x)) \) is not typable in T\textsubscript{aff}.

| Theorem | |
|-----------------|-----------------|-----------------|
| **Affine Cyclic proofs** | \( \iff \) | **System T\textsubscript{aff}** | \( \iff \) | **Prim. rec.** |
| **Cyclic proofs** | \( \iff \) | **System T** | \( \iff \) | **Peano** |

\[ \lambda a b. \text{Rec}(b, a, s(y)) \]

Easy

Hard
Results

**System $T_{aff}$**: Affine version of System T, data cannot be duplicated.

Example: $\lambda f . x . f ( f ( x ) )$ is not typable in $T_{aff}$.

<table>
<thead>
<tr>
<th>Theorem</th>
<th>$\iff$</th>
<th>$\iff$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Affine Cyclic proofs</strong></td>
<td>$\iff$</td>
<td>$\iff$</td>
</tr>
<tr>
<td><strong>System $T_{aff}$</strong></td>
<td>$\iff$</td>
<td>$\iff$</td>
</tr>
<tr>
<td><strong>Cyclic proofs</strong></td>
<td>$\iff$</td>
<td>$\iff$</td>
</tr>
<tr>
<td><strong>System $T$</strong></td>
<td>$\iff$</td>
<td>$\iff$</td>
</tr>
<tr>
<td><strong>Prim. rec.</strong></td>
<td>$\iff$</td>
<td>$\iff$</td>
</tr>
<tr>
<td><strong>Peano</strong></td>
<td>$\iff$</td>
<td>$\iff$</td>
</tr>
</tbody>
</table>

Open problems in proof theory: infinite descent versus induction.
Proof schemes

Affine proofs $\rightarrow T_{\text{aff}}$:  
- normal form for proofs, with explicit hierarchy of cycles,
- inductively build $T_{\text{aff}}$ terms.
Proof schemes

**Affine proofs → $T_{aff}$:**

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

**Affine proofs → Prim. rec.:**

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

**Affine proofs → $T_{aff}$:**
- normal form for proofs, with explicit hierarchy of cycles,
- inductively build $T_{aff}$ terms.

**Affine proofs → Prim. rec.:**
- Stronger normal form through $T_{aff}$,
- $RCA_0$: constructive fragment of 2nd-order arithmetic,
- $\forall$ affine cyclic proof, prove in $RCA_0$ that its computation terminates,
- From reverse maths: $RCA_0 \leftrightarrow$ Prim. rec.

**Proofs → System $T$:**
- $ACA_0$: $RCA_0 +$ König’s lemma,
- $\forall$ cyclic proof, prove in $ACA_0$ that its computation terminates,
- Conservativity result: $ACA_0 \leftrightarrow$ Peano for integer functions,
- Classic result: Peano $\leftrightarrow$ System $T$. 


Conclusion

Open problems:

▶ Avoid the “blackbox” of reverse maths.
▶ Use of reverse maths → 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 → 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!