Theory of Computing

Nao Hirokawa

School of Information Science, JAIST
Background
Research Area

**AREA**

**term rewriting** (science of computation)
Research Area

**term rewriting** (science of computation)

- Turing-complete computational model
Research Area

**term rewriting** (science of computation)

- Turing-complete computational model
- abstract model of programs
Research Area

**AREA**

term rewriting (science of computation)

- Turing-complete computational model
- abstract model of programs
- used in many areas:
  - programming language semantics
  - theorem proving
  - type theory
  - Mathematica
  - CafeOBJ
  - ...

Research Area

term rewriting (science of computation)

- Turing-complete computational model
- abstract model of programs
- used in many areas:
  - programming language semantics
  - theorem proving
  - type theory
  - Mathematica
  - CafeOBJ
  - ...

AIM

- establish techniques for analysing properties of rewriting
- investigate further applications of rewriting
Example 1

Haskell program

\[
\begin{align*}
\text{append}([], z) &= z \\
\text{append}(x : y, z) &= x : \text{append}(y, z)
\end{align*}
\]
Example 1

Haskell program

\[
\begin{align*}
\text{append}([], z) &= z \\
\text{append}(x : y, z) &= x : \text{append}(y, z)
\end{align*}
\]

corresponding term rewrite system (TRS)

\[
\begin{align*}
\text{append}([], z) &\rightarrow z \\
\text{append}(x : y, z) &\rightarrow x : \text{append}(y, z)
\end{align*}
\]
Example 1

Haskell program

\[
\text{append}([], z) = z \\
\text{append}(x : y, z) = x : \text{append}(y, z)
\]

corresponding term rewrite system (TRS)

\[
\text{append}([], z) \to z \\
\text{append}(x : y, z) \to x : \text{append}(y, z)
\]

execution of program is rewriting:

\[
\text{append}(1 : 2 : [], 3 : 4 : []) \to 1 : \text{append}(2 : [], 3 : 4 : []) \\
\to 1 : 2 : \text{append}([], 3 : 4 : []) \\
\to 1 : 2 : 3 : 4 : []
\]
Example II

C program

```c
int sum(int n) {
    int x = 0;
    while (n > 0)
        x += n--;
    return x;
}
```

---

corresponding TRS

\[ \text{sum}(n) \rightarrow f(0, n) \]
\[ 0 + y \rightarrow y \]
\[ f(x, s(n)) \rightarrow f(x + s(n), n) \]
\[ s(x) + y \rightarrow s(x + y) \]
\[ f(x, 0) \rightarrow x \]
Example II

C program

```c
int sum(int n) {
    int x = 0;
    while (n > 0)
        x += n --;
    return x;
}
```

corresponding TRS

\[
\begin{align*}
\text{sum}(n) & \rightarrow f(0, n) \\
f(x, s(n)) & \rightarrow f(x + s(n), n) \\
f(x, 0) & \rightarrow x
\end{align*}
\]

\[
\begin{align*}
0 + y & \rightarrow y \\
s(x) + y & \rightarrow s(x + y)
\end{align*}
\]
Example III

TRS in Mathematica

\[
\frac{d}{dx}(x) \rightarrow 1 \quad \frac{d}{dx}(f + g) \rightarrow \frac{d}{dx}(f) + \frac{d}{dx}(g)
\]

\[
\frac{d}{dx}(n) \rightarrow 0 \quad \frac{d}{dx}(f \times g) \rightarrow \frac{d}{dx}(f) \times g + f \times \frac{d}{dx}(g)
\]
Research Topics
Research Topic I: Complexity Analysis

program should terminate within feasible time
Research Topic I: Complexity Analysis

program should terminate within feasible time

MISSION

implement complexity tool

program  →  COMPLEXITY TOOL  →  complexity

e.g. quick sort

$O(n^2)$
Research Topic I: Complexity Analysis

program should terminate **within feasible time**

**MISSION**

implement complexity tool

program → **COMPLEXITY TOOL** → complexity

e.g. quick sort

how to analyse?

\( O(n^2) \)
Quiz

TRS

\[
\begin{align*}
\text{append}([], z) & \rightarrow z \\
\text{append}(x : y, z) & \rightarrow x : \text{append}(y, z)
\end{align*}
\]

The execution of the program is rewriting:

\[
\begin{align*}
\text{append}(1 : 2 : [], 3 : 4 : []) & \rightarrow 1 : \text{append}(2 : [], 3 : 4 : []) \\
& \rightarrow 1 : 2 : \text{append}([], 3 : 4 : []) \\
& \rightarrow 1 : 2 : 3 : 4 : []
\end{align*}
\]
Quiz

TRS

\[
\text{append}([], z) \rightarrow z \\
\text{append}(x : y, z) \rightarrow x : \text{append}(y, z)
\]

execution of program is rewriting:

\[
\text{append}(1 : 2 : [], 3 : 4 : []) \rightarrow 1 : \text{append}(2 : [], 3 : 4 : [])) \\
\quad \rightarrow 1 : 2 : \text{append}([], 3 : 4 : []) \\
\quad \rightarrow 1 : 2 : 3 : 4 : []
\]

QUESTIONS

- does \text{append} terminate in $O(n^3)$ (cubic) ?
Quiz

TRS

\[
\text{append}([], z) \rightarrow z \\
\text{append}(x : y, z) \rightarrow x : \text{append}(y, z)
\]

execution of program is rewriting:

\[
\text{append}(1 : 2 : [], 3 : 4 : []) \rightarrow 1 : \text{append}(2 : [], 3 : 4 : []) \\
\rightarrow 1 : 2 : \text{append}([], 3 : 4 : []) \\
\rightarrow 1 : 2 : 3 : 4 : []
\]

QUESTIONS

- does \text{append} terminate in \(O(n^3)\) (cubic) ? — yes
- does \text{append} terminate in \(O(n^2)\) (quadratic) ?
Quiz

TRS

\[
\text{append}([], z) \rightarrow z
\]
\[
\text{append}(x : y, z) \rightarrow x : \text{append}(y, z)
\]

execution of program is rewriting:

\[
\text{append}(1 : 2 : [], 3 : 4 : []) \rightarrow 1 : \text{append}(2 : [], 3 : 4 : [] )
\]
\[
\rightarrow 1 : 2 : \text{append}([], 3 : 4 : [] )
\]
\[
\rightarrow 1 : 2 : 3 : 4 : []
\]

QUESTIONS

• does \text{append} terminate in \(O(n^3)\) (cubic) ? — yes
• does \text{append} terminate in \(O(n^2)\) (quadratic) ? — yes
• does \text{append} terminate in \(O(n)\) (linear) ?
Quiz

TRS

append([], z) → z
append(x : y, z) → x : append(y, z)

execution of program is rewriting:

→ 1 : 2 : append([], 3 : 4 : [])
→ 1 : 2 : 3 : 4 : []

QUESTIONS

- does append terminate in $O(n^3)$ (cubic) ? — yes
- does append terminate in $O(n^2)$ (quadratic) ? — yes
- does append terminate in $O(n)$ (linear) ? — yes
- does append terminate in $O(1)$ (constant) ?
Quiz

TRS

\[
\text{append}([], z) \rightarrow z \\
\text{append}(x : y, z) \rightarrow x : \text{append}(y, z)
\]

execution of program is rewriting:

\[
\text{append}(1 : 2 : [], 3 : 4 : []) \rightarrow 1 : \text{append}(2 : [], 3 : 4 : []) \\
\rightarrow 1 : 2 : \text{append}([], 3 : 4 : []) \\
\rightarrow 1 : 2 : 3 : 4 : []
\]

**QUESTIONS**

- does \text{append} terminate in $O(n^3)$ (cubic) ? — yes
- does \text{append} terminate in $O(n^2)$ (quadratic) ? — yes
- does \text{append} terminate in $O(n)$ (linear) ? — yes
- does \text{append} terminate in $O(1)$ (constant) ? — no
Automated Complexity Analysis

Hirokawa & Moser, IJCAR 2008

TRS

append([], z) → z
append(x : y, z) → x : append(y, z)
Automated Complexity Analysis

Hirokawa & Moser, IJCAR 2008

TRS

\[
\text{append}([], z) \rightarrow z \\
\text{append}(x : y, z) \rightarrow x : \text{append}(y, z)
\]

\(\mu\)-compatible constructor-restricted interpretation \(\mathcal{A}\)

\[
\text{append}_\mathcal{A}(x, y) = x \\
[.]_\mathcal{A} = 1 \\
x :_\mathcal{A} y = y + 1 \\
c_\mathcal{A} = 0
\]
Automated Complexity Analysis

Hirokawa & Moser, IJCAR 2008

TRS

\[
\text{append}([], z) \rightarrow z \\
\text{append}(x : y, z) \rightarrow x : \text{append}(y, z)
\]

\(\mu\)-compatible constructor-restricted interpretation \(A\)

\[
\text{append}_A^\mu(x, y) = x \quad []_A = 1 \quad x :_A y = y + 1 \quad c_A = 0
\]

then

\[
\text{append}_A^\mu([], z) = 1 > 0 = c_A \\
\text{append}_A^\mu(x : y, z) = y + 1 > y = \text{append}_A^\mu(y, z)
\]
Automated Complexity Analysis

Hirokawa & Moser, IJCAR 2008

TRS

\[
\begin{align*}
\text{append}([], z) & \rightarrow z \\
\text{append}(x : y, z) & \rightarrow x : \text{append}(y, z)
\end{align*}
\]

\(\mu\)-compatible constructor-restricted interpretation \(\mathcal{A}\)

\[
\begin{align*}
\text{append}^\mathcal{A}(x, y) & = x & \mathcal{A}[] & = 1 & x : \mathcal{A} y & = y + 1 & c_{\mathcal{A}} & = 0
\end{align*}
\]

then

\[
\begin{align*}
\text{append}^\mathcal{A}([], z) & = 1 > 0 = c_{\mathcal{A}} \\
\text{append}^\mathcal{A}(x : y, z) & = y + 1 > y = \text{append}^\mathcal{A}(y, z)
\end{align*}
\]

hence, append terminates in \(O(n)\)
Research

1. develop your own method
Research

1. develop your own method

2. compete against complexity analysers:
   
   \text{TcT} \quad \text{CaT} \quad \text{AProve} \quad \text{Matchbox} \quad \ldots
Research

1. develop your own method

2. compete against complexity analysers:
   TcT    CaT    AProVE    Matchbox    ...

3. attend international competition:
   termCOMP
Research Topic II: Confluence

TRS is confluent if $^* \leftarrow \cdot \rightarrow^* \subseteq \rightarrow^* \cdot \leftarrow^*$

program should have same output for same input
Research Topic II: Confluence

TRS is confluent if \( * \leftarrow \cdot \rightarrow * \subseteq \rightarrow * \cdot * \leftarrow \)

Program should have **same output for same input**

**MISSION**

Implement **confluence tool**

TRS \( \rightarrow \) **CONFLUENCE TOOL** \( \rightarrow \) YES/NO/MAYBE
Past Research Themes

main projects:

- Termination Analysis for LISP/Scheme
- Solving Inequalities
- Theorem Proving

minor projects:

- Confluence Tool
- Database based on Ambiguous Search
- Mini-Mathematica
Past Research Themes

main projects:

- Termination Analysis for LISP/Scheme
Past Research Themes

main projects:

- Termination Analysis for LISP/Scheme
- Solving Inequalities

minor projects:

- Confluence Tool
- Database based on Ambiguous Search
- Mini-Mathematica
Past Research Themes

main projects:

- Termination Analysis for LISP/Scheme
- Solving Inequalities
- Theorem Proving
Past Research Themes

main projects:

- Termination Analysis for LISP/Scheme
- Solving Inequalities
- Theorem Proving

minor projects:
Past Research Themes

main projects:

- Termination Analysis for LISP/Scheme
- Solving Inequalities
- Theorem Proving

minor projects:

- Confluence Tool
Past Research Themes

main projects:

- Termination Analysis for LISP/Scheme
- Solving Inequalities
- Theorem Proving

minor projects:

- Confluence Tool
- Database based on Ambiguous Search
Past Research Themes

main projects:

- Termination Analysis for LISP/Scheme
- Solving Inequalities
- Theorem Proving

minor projects:

- Confluence Tool
- Database based on Ambiguous Search
- Mini-Mathematica
Research Life
What You Learn

- functional programming
  OCaml, F#, Haskell, Scala

qsort \([\ ]\) = [\]
qsort (x : z) = qsort [y | x ← z, y ≤ x] ++ [x] ++
qsort [y | x ← z, y > x]
What You Learn

- **Functional programming**  OCaml, F#, Haskell, Scala

  \[
  \text{qsort } [[]] = []
  \]
  \[
  \text{qsort } (x : z) = \text{qsort } [y | x ← z, y ≤ x] ++ [x] ++ \text{qsort } [y | x ← z, y > x]
  \]

- **Constraint solving**  SAT, SMT
What You Learn

- **functional programming**  
  OCaml, F#, Haskell, Scala

  \[
  \text{qsort} \ [\ ] = \ [\ ] \\
  \text{qsort} \ (x : z) = \text{qsort} \ [y \mid x \leftarrow z, y \leq x] + + [x] ++ \\
  \text{qsort} \ [y \mid x \leftarrow z, y > x]
  \]

- **constraint solving**  
  SAT, SMT

- **basic set theory**
Sudoku

1  8  7
3  2
7

7  1
6  4
3

4  5  3
2  8
6
how to solve?

- develop algorithm?
- use SAT solver!
Events

Informal Meetings (Sep & Feb)
TRS Meeting (Aug & Feb), Japan-Austria Joint Project
Events

Informal Meetings (Sep & Feb)
TRS Meeting (Aug & Feb), Japan-Austria Joint Project

International Conferences (submission: Jan & Feb)
RTA, CADE, IJCAR, ...
Events

Informal Meetings (Sep & Feb)
TRS Meeting (Aug & Feb), Japan-Austria Joint Project

International Conferences (submission: Jan & Feb)
RTA, CADE, IJCAR, ...

Tool Competitions (June)
TERMCOMP, CoCo, ...
Contact Information

Hirokawa Lab

• room: I-54a

student who likes functional programming

CafeOBJ, OCaml, Ruby, Scala, ...

is welcome!
Contact Information

Hirokawa Lab

- room: I-54a
- email: hirokawa@jaist.ac.jp
Contact Information

Hirokawa Lab

- **room:** I-54a
- **email:** hirokawa@jaist.ac.jp

❤️ student who likes functional programming is welcome!

CafeOBJ, OCaml, Ruby, Scala, ...