Theory and Practice of Second-Order Rewriting: Foundation, Evolution, and SOL

Makoto Hamana

Department of Computer Science,
Gunma University, Japan

FLOPS 2020

Virtual Symposium, 14th September, 2020

This Talk

An overview of the theory and practice of second-order rewriting

Introduction

- ▷ Rewrite rules are ubiquitous in Programming Language research

(beta) $(\lambda x.\, M)\, N \quad \Rightarrow \quad M[x := N]$

(assoc) let y=(let x=L in M) in N $\Rightarrow$ let x=L in (let y=M in N)

(unit) $0 + X \quad \Rightarrow \quad X$

(unitR) return(x) $\gg\!=$ λy.K y $\Rightarrow$ K x

Main problems

- ▷ Terminating?
- ▷ Unique normal forms? $\Leftarrow$ confluence
- ▷ Decidability of equational theory
- ▶ General framework: Second-Order Computation Systems
- ▶ System: SOL, Second-Order Laboratory

This Talk

1. Foundation
   - ▷ Formal definition
   - ▷ Properties: Confluence and termination
2. Evolution
   - ▷ Second-order abstract syntax
   - ▷ Second-order algebraic theories
   - ▷ Second-order computation systems
   - ▷ Examples: $\boldsymbol{\lambda}$-calculus, global state
3. SOL
   - ▷ Interface: GHCi, Web interface
   - ▷ Live Demo

A Dream of Pragmatics

How Second-Order Rewriting Evolved

Plotkin RTA 1998 Binding Algebras

(Tsukuba)

How Second-Order Rewriting Evolved

Plotkin RTA 1998 Binding Algebras

(Tsukuba)

- Algebras $\rightsquigarrow$ Types
- Algebraization of program concepts

How Second-Order Rewriting Evolved

Plotkin	RTA (Tsukuba)	1998	Binding Algebras: A Step from Universal Algebra to Type Theory
Fiore, Plotkin, Turi	LICS	1999	Abstract Syntax and Variable Binding

How Second-Order Rewriting Evolved

Plotkin	RTA	1998	Binding Algebras:
	(Tsukuba)		A Step from Universal Algebra to Type Theory
Fiore, Plotkin, Turi	LICS	1999	Abstract Syntax and Variable Binding
Fiore	PPDP	2002	Typed Abstract Syntax with Variable Binding
Hamana	APLAS	2004	Free Σ-monoids: HOAS with Metavariables
Hamana	RTA	2005	Complete algebraic semantics of second-order rewriting
Ghani,Uustalu,Hamana	HOSC	2006	HOAS with Explicit substitutions
Hamana	PPDP	2007	Higher-order Semantic labelling for Typed rewriting
Fiore,Hur,Mahmoud	MFCS,CSL	2010	Second-Order Algebraic Theories
Fiore, Hamana	LICS	2013	Polymorphic Algebraic Theories
Hamana	ICFP	2017	Second-Order Computation Systems and SOL
Hamana	FLOPS	2018	PolySOL
Hamana,Abe,Kikuchi	SCP	2020	Polymorphic Confluence with Call-by-Value

This Talk

1. Foundation
   - ▷ Formal definition
   - ▷ Properties: Confluence and termination
2. Evolution
   - ▶ Second-order abstract syntax
   - ▷ Second-order algebraic theories
   - ▷ Second-order computation systems
   - ▷ Examples: $\boldsymbol{\lambda}$-calculus, global state
3. SOL
   - ▷ Interface: Web interface
   - ▷ Live Demo

What is Second-Order Abstract Syntax

- Ordinary notation

$$\boldsymbol{\lambda x^a.fx} \qquad\qquad \forall \boldsymbol{x}.\mathsf{bird}(\boldsymbol{x}) \longrightarrow \mathsf{fly}(\boldsymbol{x})$$

- Second-order abstract syntax

$$\mathsf{lam}(\boldsymbol{x}^a.\mathsf{app}(\boldsymbol{f},\boldsymbol{x})), \qquad\qquad \mathsf{forall}(\boldsymbol{x}.\mathsf{imp}(\mathsf{bird}(\boldsymbol{x}),\mathsf{fly}(\boldsymbol{x})))$$

- using a signature

$$\mathsf{lam} : (\boldsymbol{a} \longrightarrow \boldsymbol{b}) \longrightarrow \mathsf{Arr}(\boldsymbol{a},\boldsymbol{b}), \quad \mathsf{app} : \mathsf{Arr}(\boldsymbol{a},\boldsymbol{b}), \boldsymbol{a} \longrightarrow \boldsymbol{b}, \quad \mathsf{forall} : (\boldsymbol{i} \longrightarrow \boldsymbol{i}) \longrightarrow \boldsymbol{i},$$

- with metavariables

$$\mathsf{lam}(\boldsymbol{x}.\mathrm{M}[\boldsymbol{x}]) \qquad\qquad \mathsf{forall}(\boldsymbol{x}.\mathsf{imp}(\mathrm{P}[\boldsymbol{x}],\mathrm{Q}[\boldsymbol{x}]))$$

- Substitution for metavariables

$$\mathsf{lam}(\boldsymbol{x}.\mathrm{M}[\boldsymbol{x}])\{\mathrm{M} \mapsto \boldsymbol{y}.\mathsf{app}(\boldsymbol{f},\boldsymbol{y})\} = \mathsf{lam}(\boldsymbol{x}.\mathsf{app}(\boldsymbol{f},\boldsymbol{x}))$$

$$\mathsf{forall}(\boldsymbol{x}.\mathrm{L}[\boldsymbol{x}])\{\mathrm{L} \mapsto \boldsymbol{y}.\mathsf{forall}(\boldsymbol{y}.\mathsf{imp}(\mathsf{bird}(\boldsymbol{y}),\mathsf{fly}(\boldsymbol{y})))\} = \cdots$$

Second-Order Abstract Syntax: Related Work

- Meta-terms
  - Peter Aczel. *A general Church-Rosser theorem*, unpublished, University of Manchester, 1978.
  - Peter Aczel. *Frege structures and the notions of proposition, truth and set*, The Kleene Symposium: Proceedings of the Symposium Held June 18-24, 1978 at Madison, Wisconsin, U.S.A.
  - J. W. Klop. *Combinatory Reduction Systems*, CWI, volume 127 of Mathematical Centre Tracts, 1980.

Second-Order Abstract Syntax: Types and Signature

Typed syntax with variable binding and metavariables

- $\mathcal{A}$ is a set of atomic types (e.g. Bool, Nat, etc.)
- the set of molecular types (mol types) $\mathcal{B}$ is

$$\mathcal{B} = \mathcal{A} \cup \{T(a_1, \ldots, a_n) \mid a_1, \ldots, a_n \in \mathcal{B},\ T \text{ is an } n\text{-ary type constructor}\}$$

- A signature Σ is a set of function symbols of the form

$$f : (\overline{a_1} \to b_1), \ldots, (\overline{a_m} \to b_m) \to c$$

  where all a_i, b_i, c are mol types

- Any function symbol is of up to second-order type

Second-Order Abstract Syntax: Terms

- A metavariable is a variable of (at most) first-order function type, declared as $M : \overline{a} \longrightarrow b$
- A variable is always of a molecular type b (not of a function type)
- The raw syntax

  - Terms have the form $t ::= x \mid x.t \mid f(t_1, \ldots, t_n)$
  - Meta-terms extend terms to $t ::= x \mid x.t \mid f(t_1, \ldots, t_n) \mid M[t_1, \ldots, t_n]$

  The last form $M[t_1, \ldots, t_n]$ is called meta-application

Second-Order Abstract Syntax: Well-typed Terms

- Judgment

$$\Theta \ \triangleright \ \Gamma \ \vdash \ \ t \ \ : b$$

- Typing rules

$$\text{(Var)}\ \frac{y : b \in \Gamma}{\Theta \ \triangleright \ \Gamma \ \vdash \ \ y \ \ : b} \qquad \text{(MetaApp)}\ \frac{(M : a_1, \ldots, a_m \to b) \in \Theta \quad \Theta \ \triangleright \ \Gamma \ \vdash \ \ t_i \ \ : a_i \ \ (1 \leq i \leq m)}{\Theta \ \triangleright \ \Gamma \ \vdash \ \ M[t_1, \ldots, t_m] \ \ : b}$$

$$\text{(Fun)}\ \frac{f : (\overline{a_1} \to b_1), \cdots, (\overline{a_m} \to b_m) \to c \in \Sigma \quad \Theta \ \triangleright \ \Gamma, \overline{x_i : a_i} \ \vdash \ \ t_i \ \ : b_i \ \ (1 \leq i \leq m)}{\Theta \ \triangleright \ \Gamma \ \vdash \ \ f(\ \overline{x_1^{a_1}}.t_1, \ldots \overline{x_m^{a_m}}.t_m\ ) \ \ : c}$$

Second-Order Abstract Syntax, Example: the simply-typed $\boldsymbol{\lambda}$-calculus

- The signature $\boldsymbol{\Sigma_{\mathbf{lam}}}$ for the $\boldsymbol{\lambda}$-terms

$$\begin{aligned} \mathsf{lam}_{a,b} &: (a \longrightarrow b) \longrightarrow \mathsf{Arr}(a,b) \\ \mathsf{app}_{a,b} &: \mathsf{Arr}(a,b), a \longrightarrow b \end{aligned}$$

  - parameterised by mol types $a, b \in \mathcal{B}$
  - Arr is a binary type constructor
  - app is denoted by @.

- Encoding of $\boldsymbol{\lambda}$-terms

$$\triangleright \vdash \mathsf{lam}_{a,b}(x^a.x) : \mathsf{Arr}(a,b) \qquad\qquad \triangleright f : \mathsf{Arr}(a,b) \vdash \mathsf{lam}_{a,b}(x^a.\mathsf{app}(f,x)) : b$$

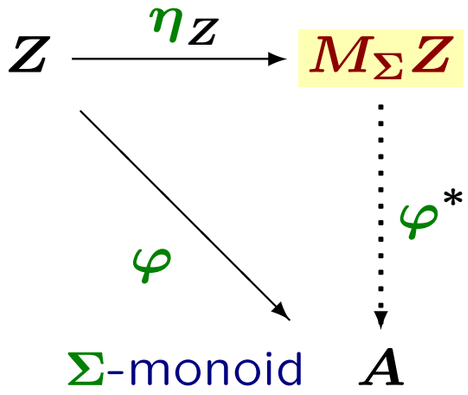
- Encoding of meta-level $\boldsymbol{\lambda}$-terms

$$\begin{aligned} M : a \longrightarrow b,\ N : a \triangleright \vdash \mathsf{lam}_{a,b}(x.M[x])\ @_{a,b}\ N : b \\ M : a \longrightarrow b,\ N : a \triangleright \vdash \qquad M[N]\ : b \end{aligned}$$

Second-Order Abstract Syntax: Semantics

Algebraic models of abstract syntax with variable binding and substitutions
= Σ-monoids [Fiore,Plotkin,Turi LICS'99]
= Σ-algebras + monoids in a presheaf category $\mathbf{Set}^{\mathbb{F}}$

A free Σ-monoid $M_\Sigma Z$ generated by Z is a second-order abstract syntax [Hamana APLAS'04] . with metavariables



unique Σ-monoid morphism = interpretation

- A is arbitrary algebraic model
- $M_\Sigma Z_b(\Gamma) \ni t ::= x \mid x.t \mid f(t_1, \ldots, t_n) \mid M[t_1, \ldots, t_n]$
- Models are useful to evaluate terms (e.g. sizes)

This Talk

1. Foundation
   - ▷ Formal definition
   - ▷ Properties: Confluence and termination
2. Evolution
   - ▷ Second-order abstract syntax
   - ▶ Second-order algebraic theories
   - ▷ Second-order computation systems
   - ▷ Examples: $\boldsymbol{\lambda}$-calculus, global state
3. SOL
   - ▷ Interface: Web interface
   - ▷ Live Demo

Second-Order Equational Logic

- An equation is of the form

$$\Theta \ \triangleright \ \Gamma \ \vdash \quad s \ = \ t \ : b$$

  for meta-terms $\Theta \ \triangleright \ \Gamma \ \vdash \quad s \ : b$ and $\Theta \ \triangleright \ \Gamma \ \vdash \quad t \ : b$

- Second-order equational logic is an equational logic on second-order terms
- A given set $\mathcal{E}$ of equations (axioms) generates a second-order equational theory.
- The inference rules:

$$\text{(Ax)} \ \frac{\begin{array}{c}\Theta \ \triangleright \ \Gamma', \overline{x_i : a_i} \ \vdash \quad s_i \ : b_i \quad (1 \leq i \leq k) \\ (M_1 : (\overline{a_1} \to b_1), \ldots, M_k : (\overline{a_k} \to b_k) \ \triangleright \ \Gamma \ \vdash \quad t_1 = t_2 \ : c) \in \mathcal{E}\end{array}}{\Theta \ \triangleright \ \Gamma, \Gamma' \ \vdash \quad t_1 \, [\, \overline{M \mapsto \overline{x}.s} \,] = t_2 \, [\, \overline{M \mapsto \overline{x}.s} \,] \ : c}$$

  and reflexivity, symmetry, transitivity, and congruence

- Models: second-order algebras satisfying $\mathcal{E}$

Second-Order Equational Logic, Example: Simply-typed $\boldsymbol{\lambda}$-calculus

- The signature $\boldsymbol{\Sigma_{\mathbf{lam}}}$ for the $\boldsymbol{\lambda}$-terms

$$\begin{aligned} \mathsf{lam}_{a,b} &: (a \longrightarrow b) \longrightarrow \mathsf{Arr}(a,b) \\ \mathsf{app}_{a,b} &: \mathsf{Arr}(a,b), a \longrightarrow b \end{aligned}$$

  - parameterised by mol types $a, b \in \mathcal{B}$
  - Arr is a binary type constructor
  - app is denoted by @.

- The β and η laws

$$\begin{aligned} M : a \longrightarrow b,\ N : a \triangleright \vdash \mathsf{lam}_{a,b}(x^a.M[x])\ @_{a,b}\ N &= M[N] : b \\ M : b \triangleright \vdash \mathsf{lam}_{a,b}(x^a.M\ @_{a,b}\ x) &= M : b \end{aligned}$$

- Ordinary $\boldsymbol{\lambda}$-theory is obtained as a second-order equational theory
- The CCC model of $\boldsymbol{\lambda}$-calculus can be an second-order algebraic model [Fiore, PPDP'02]

This Talk

1. Foundation
   - ▷ Formal definition
   - ▷ Properties: Confluence and termination
2. Evolution
   - ▷ Second-order abstract syntax
   - ▷ Second-order algebraic theories
   - ▶ Second-order computation systems
   - ▷ Examples: $\boldsymbol{\lambda}$-calculus, global state
3. SOL
   - ▷ Interface: Web interface
   - ▷ Live Demo

Second-Order Rewriting

▷ A computation rule is of the form

$$\Theta \ \triangleright \ \Gamma \ \vdash \quad \ell \Rightarrow r \quad : b$$

1. ℓ is a deterministic second-order pattern
2. all metavariables in r appear in ℓ.

▷ Second-order pattern [Miller'01]:
a meta-term ℓ in which every occurrence of meta-application is of the form

$$M[x_1, \ldots, x_n],$$

where $x_1, \ldots, x_n$ are distinct bound variables.

▷ Deterministic second-order pattern [Yokoyama, Hu, Takeichi, IPL'04]:
every occurrence of meta-application $M[t_1, \ldots, t_n]$ satisfies

1. every t_i is a term without binders, metavariables or free variables, but it can contain function symbols with arity $n > 0$ and bound variables
2. every t_i contains at least one bound variable,
3. $t_i \not\trianglelefteq t_j$ for every $1 \leq i, j \leq n$.

Second-Order Rewriting

- A computation rule is of the form

$$\Theta \rhd \Gamma \vdash \quad \ell \Rightarrow r \quad : b$$

  1. ℓ is a deterministic second-order pattern
  2. all metavariables in r appear in ℓ.

- Second-order pattern [Miller'01]:
  a meta-term ℓ in which every occurrence of meta-application is of the form

$$M[x_1, \ldots, x_n],$$

  where $x_1, \ldots, x_n$ are distinct bound variables.

- Deterministic second-order pattern [Yokoyama, Hu, Takeichi, IPL'04]:
  Good: A solvable matching problem $p = t$ has a unique matcher
  for a deterministic second-order pattern p and a term.

- E.g. `M[cons(x,y)]`, `lam(x.M[var(x)])`, `f(Y,c(x.y.K[y,d(x)]))`

Second-Order Rewriting

- A signature Σ and a set $\mathcal{C}$ of computation rules form a computation system $(\Sigma, \mathcal{C})$.
- Given $\mathcal{C}$, one-step rewriting $\Rightarrow_{\mathcal{C}}$ is defined by

$$\text{(Rule)}\ \frac{\begin{array}{c}\Theta \ \triangleright\ \Gamma', \overline{x_i : a_i} \vdash\ s_i\ : b_i \quad (1 \le i \le k) \\ (M_1 : (\overline{a_1} \to b_1), \ldots, M_k : (\overline{a_k} \to b_k) \triangleright\ \vdash\ \ell \Rightarrow r\ : c) \in \mathcal{C}\end{array}}{\Theta \ \triangleright\ \Gamma' \vdash\ \ell\,[\,\overline{M \mapsto \overline{x}.s}\,] \Rightarrow_{\mathcal{C}} r\,[\,\overline{M \mapsto \overline{x}.s}\,]\ : c}$$

$$\text{(Fun)}\ \frac{\begin{array}{c}f : (\overline{a_1} \to b_1), \cdots, (\overline{a_k} \to b_k) \to c \in \Sigma \\ \Theta \ \triangleright\ \Gamma, \overline{x_i : a_i} \vdash\ t_i \Rightarrow_{\mathcal{C}} t_i'\ : b_i \quad (\text{some } i \text{ s.t. } 1 \le i \le k)\end{array}}{\Theta \ \triangleright\ \Gamma \vdash\ f(\overline{x_1^{a_1}}.t_1, \ldots, \overline{x_i^{a_i}}.t_i, \ldots, \overline{x_1^{a_1}}.t_k) \Rightarrow_{\mathcal{C}} f(\overline{x_1^{a_1}}.t_1, \ldots, \overline{x_i^{a_i}}.t_i', \ldots, \overline{x_1^{a_1}}.t_k)\ : c}$$

Confluence

- To check a computation system $(\boldsymbol{\Sigma}, \boldsymbol{\mathcal{C}})$ confluent
- Based on Newman's lemma: local confluence & SN implies confluence
- Local confluence is checked by Knuth-Bendix critical pair checking
- Critical pairs are computed by second-order unification, modified Function-as-Constructor Unification (FCU) [Libal,Miller FSCD'16]
- Unification between deterministic second-order patterns
- For more details, [H. PACMPL'17,JFP'19][Hamana,Abe,Kikuchi SCP'20]

Termination Checking of Second-Order Computation Systems

▷ To check a computation system $(\Sigma, \mathcal{C})$ Strongly Normalising (SN)

1. Interpretation by an algebraic model with well-founded order

$$\begin{array}{ccc} s & \Rightarrow_{\mathcal{C}} & t \\ \downarrow & & \downarrow \\ [\![s]\!] & >_A & [\![t]\!] \end{array} \qquad \begin{array}{l} \text{a rewrite step} \\ [\![-]\!] \\ \text{in a model } (A, >_A) \end{array}$$

Thm. [H. RTA'04] A computation system $(\Sigma, \mathcal{C})$ is SN iff there exists a well-founded model $(A, >_A)$ of $(\Sigma, \mathcal{C})$.

2. Syntactic check: The General Schema [Blanqui'00, RTA'00,TCS'16]
3. Modular termination

   $\mathbf{A}$: SN and $\mathbf{B}$: SN $\quad\Rightarrow\quad$ $\mathbf{A} \uplus \mathbf{B}$ SN

Modular Termination Checking

- $\mathbf{A}$: SN and $\mathbf{B}$: SN $\Rightarrow$ $\mathbf{A} \uplus \mathbf{B}$ SN?
- **Toyama's counterexample [IPL 1987]**

$\mathbf{A}$:
```
f(0,1,X) => f(X,X,X)
```

$\mathbf{B}$:
```
or(X,Y)  => X
or(X,Y)  => Y
```

- Each of $\mathbf{A}$ and $\mathbf{B}$ is terminating.
- But

```
  f(or(0,1),or(0,1),or(0,1))
=> f(  0    ,or(0,1),or(0,1))
=> f(  0    ,   1   ,or(0,1))
=> f(or(0,1),or(0,1),or(0,1))
```

Modular Termination Checking

Theorem [Hamana arXiv:1912.03434]

Given computation systems, where $\mathbf{\Theta}$ is the set of constructors

- ▷ $(\mathbf{\Sigma_A} \uplus \mathbf{\Theta}, \mathbf{A})$
- ▷ $(\mathbf{\Sigma_A} \uplus \mathbf{\Sigma_B} \uplus \mathbf{\Theta}, \mathbf{B})$, if
  - (i) $\mathbf{A}$ is accessible
  - (ii) each $\mathbf{A}$-function calls only $\mathbf{A}$-functions with second-order patterns
  - (iii) $\mathbf{A}$ with projection rules ($\langle \boldsymbol{M_1}, \boldsymbol{M_2} \rangle \Rightarrow \boldsymbol{M_i}$) is SN (not necessary by GS)
  - (iv) $\mathbf{B}$ is SN by the General Schema.

then $(\mathbf{\Sigma_A} \uplus \mathbf{\Sigma_B} \uplus \mathbf{\Theta},\ \mathbf{A} \uplus \mathbf{B})$ is SN.

Proof By higher-order semantic labelling [H. PPDP'07]
based on the algebraic models of second-order rewriting [H. RTA'04] and GS.

Modular Termination Checking

Theorem [Hamana arXiv:1912.03434]

Given computation systems, where $\boldsymbol{\Theta}$ is the set of constructors

- ▷ $(\boldsymbol{\Sigma_A} \uplus \qquad \boldsymbol{\Theta}, \mathbf{A})$
- ▷ $(\boldsymbol{\Sigma_A} \uplus \boldsymbol{\Sigma_B} \uplus \boldsymbol{\Theta}, \mathbf{B})$, if
  - (i) $\mathbf{A}$ is accessible
  - (ii) each $\mathbf{A}$-function calls only $\mathbf{A}$-functions with second-order patterns
  - (iii) $\mathbf{A}$ with projection rules ($\langle \boldsymbol{M_1}, \boldsymbol{M_2} \rangle \Rightarrow \boldsymbol{M_i}$) is SN (not necessary by GS)
  - (iv) $\mathbf{B}$ is SN by the General Schema.

then $(\boldsymbol{\Sigma_A} \uplus \boldsymbol{\Sigma_B} \uplus \boldsymbol{\Theta},\ \mathbf{A} \uplus \mathbf{B})$ is SN.

Applications

1. Dividing $\boldsymbol{\mathcal{C}}$ into first-order part $\mathbf{A}$ + higher-order part $\mathbf{B}$, and invoking external FO SN checkers (implemented in SOL)

Modular Termination Checking

Theorem [Hamana arXiv:1912.03434]

Given computation systems, where $\mathbf{\Theta}$ is the set of constructors

- ▷ $(\mathbf{\Sigma_A} \uplus \mathbf{\Theta}, \mathbf{A})$
- ▷ $(\mathbf{\Sigma_A} \uplus \mathbf{\Sigma_B} \uplus \mathbf{\Theta}, \mathbf{B})$, if
  - (i) $\mathbf{A}$ is accessible
  - (ii) each $\mathbf{A}$-function calls only $\mathbf{A}$-functions with second-order patterns
  - (iii) $\mathbf{A}$ with projection rules ($\langle \boldsymbol{M_1}, \boldsymbol{M_2} \rangle \Rightarrow \boldsymbol{M_i}$) is SN (not necessary by GS)
  - (iv) $\mathbf{B}$ is SN by the General Schema.

then $(\mathbf{\Sigma_A} \uplus \mathbf{\Sigma_B} \uplus \mathbf{\Theta},\ \mathbf{A} \uplus \mathbf{B})$ is SN.

Applications

2. SN of CBPV + algebraic effect handler + effect theory

. [Levy'06] [Pretnar,Plotkin'13] [Plotkin,Power'02]

Example: Theory of Global State

- The theory of global state for the single location [Plotkin,Power'02] as computation rules

```
sigst = [signature|  lk : (Val -> A) -> A;  ud : Val,A -> A   |]

gstate = [rule|
 (lu)  lk(v.ud(v,X))       => X
 (ll)  lk(w.lk(v.X[v,w])) => lk(v.X[v,v])
 (uu)  ud(V,ud(W,X))       => ud(W,X)
 (ul)  ud(V,lk(w.X[w]))    => ud(V,X[V]) |]
```

- `(lu)` says that looking-up the state, binding the value to `v`, then updating the state to `v`, is equivalent to doing nothing.
- Correspond to state monad
- An algebraization of program concept
- Decidable?
- SN: the number of `lk,ud` are decreasing.

Example: Local Confluence of Theory of Global State

We check local confluence in SOL

Step 1: Enumerate critical pairs

```
*SOL> cri gstate sigst
1: Overlap (lu)-(uu_v)--- V'|-> z1.z1, X|-> ud(H1,H2), W'|-> z1.H1, X'|-> z1.H2 --
   (lu) lk(v.ud(v,X)) => X
   (uu_v) ud(V'[v],ud(W'[v],X'[v])) => ud(W'[v],X'[v])
                          lk(v.ud(v,ud(H1,H2)))
                 ud(H1,H2) <-(lu)-∧-(uu_v)-> lk(v.ud(H1,H2))
                  ---> ud(H1,H2) =#= lk(v.ud(H1,H2)) <---
...
8: Overlap (ul)-(ll)--- X|-> z1.lk(v'.X'[v',z1]) ----------------------------
   (ul) ud(V,lk(w.X[w])) => ud(V,X[V])
   (ll) lk(w'.lk(v'.X'[v',w'])) => lk(v'.X'[v',v'])
                       ud(V,lk(w.lk(v'.X'[v',w])))
     ud(V,lk(v'.X'[v',V])) <-(ul)-∧-(ll)-> ud(V,lk(vd32.X'[vd32,vd32]))
               ---> ud(V,X'[V,V]) =OK= ud(V,X'[V,V]) <---
#NON 3 joinable... (Total 8 CPs)
```

Example: Theory of Global State

Step 2: Manual Knuth-Bendix competition

Add rules from non-joinable critical pairs

```
gstateEx1 = [rule| (lu1)  lk(v1.ud(W',X'))      => ud(W',X')
                   (lu2)  lk(v2.ud(v2,X'[v2])) => lk(w'.X'[w']) |]
```

Step 3: Recheck

```
*SOL> cri (gstate ++ gstateEx1) sigst
..
#NON 1 joinable... (Total 17 CPs)
```

Step 4: Repeat this process until all CPs are joinable

```
gstateEx2 = [rule| (l) lk(w.X) => X |]
```

- ▷ The theory of global state is decidable.
- ▷ Computation system (`gstate ++ gstateEx1 ++ gstateEx2`) provides a decision procedure.

Further Topics and Future Work

1. Foundation
   - ▷ Properties: Confluence and termination
   - ▶ Churh-Rosser modulo equivalence
2. Evolution
   - ▷ Second-order abstract syntax, algebraic theories, computation systems
   - ▷ Polymorphic computatin systems with call-by-value
   - ▷ Examples: λ-calculus, global state
   - ▶ Examples: π-calculus [Stark TCS'08][H. JFP'19], coherence of monoidal category [H. JFP'19], Skew monoidal category [H. SCP'20]
3. SOL
   - ▷ Interface: GHCi, Web interface
   - ▶ More powerful and generic rewriting tool
   - ▶ More checking properties, strategic/manual rewriting engine and UI
   - ▶ Open source at GitHub, Hackage – in preparation

Summary

An overview of the theory and practice of second-order rewriting

- Importance of algebraization of program concepts
- Second-order algebraic theories can axiomatise program concepts
- Second-order rewriting gives a computational method for second-order algebraic theories

- Looking for
  - Collaborators
  - Postdocs/internships welcome!
- Contact <hamana@gunma-u.ac.jp>

- Try SOL on web
  `http://www.cs.gunma-u.ac.jp/hamana/sol/`

How Second-Order Rewriting Evolved

Plotkin	RTA (Tsukuba)	1998	Binding Algebras: A Step from Universal Algebra to Type Theory
Fiore, Plotkin, Turi	LICS	1999	Abstract Syntax and Variable Binding
Fiore	PPDP	2002	Typed Abstract Syntax with Variable Binding
Hamana	APLAS	2004	Free Σ-monoids: HOAS with Metavariables
Hamana	RTA	2005	Complete algebraic semantics of second-order rewriting
Ghani,Uustalu,Hamana	HOSC	2006	HOAS with Explicit substitutions
Hamana	PPDP	2007	Higher-order Semantic labelling for Typed rewriting
Fiore,Hur,Mahmoud	MFCS,CSL	2010	Second-Order Algebraic Theories
Fiore, Hamana	LICS	2013	Polymorphic Algebraic Theories
Hamana	ICFP	2017	Second-Order Computation Systems and SOL
Hamana	FLOPS	2018	PolySOL
Hamana,Abe,Kikuchi	SCP	2020	Polymorphic Confluence with Call-by-Value