How to Prove Your Calculus Is Decidable

Practical Applications of Second-Order Algebraic Theories and Computation

Makoto Hamana

Department of Computer Science,
Gunma University, Japan

September 5, ICFP 2017, Oxford

This Work

- Propose the system SOL: Second-Order Laboratory
- Haskell-based tool for analysing confluence and termination of second-order computatoin rules
  - HO version of Knuth and Bendix's critical pair checking using an extended HO pattern unification (FCU) [Libal,Miller'16]
  - Powerful enough HO termination checker
- Confluence + termination $\Rightarrow$ Decidability of calculus
- Demonstrate usefulness through 8 sample HO PL calculi (global state, computational meta-lang., linear $\boldsymbol{\lambda}$, $\boldsymbol{\pi}$-calculus,$\cdots$)

Example: Monad

A monad T is a structure with two operations

$$\text{return} : a \to Ta \qquad\qquad \gg\!= \; : Ta \to (a \to Tb) \to Tb$$

satisfying

$$\begin{array}{lrcl}
\text{(unitL)} & \text{return}(x) \gg\!= \lambda y.k\,y & = & k\,x \\
\text{(unitR)} & e \gg\!= \lambda y.\text{return}(y) & = & e \\
\text{(assoc)} & (e \gg\!= \lambda x.k\,x) \gg\!= \lambda y.\ell\,y & = & e \gg\!= \lambda x.(k\,x \gg\!= \lambda y.\ell\,y)
\end{array}$$

The theory of monad is **decidable**.

Q. Given two well-typed terms s, t consisting of return, $\gg\!=$ and variables, is an equation $s = t$ derivable from the three laws?

Monad Laws as Rewrite Rules

$$\begin{array}{lrcl}
\text{(unitL)} & \text{return}(\boldsymbol{X}) \gg\!= \boldsymbol{y}.\boldsymbol{K}[\boldsymbol{y}] & \Rightarrow & \boldsymbol{K}[\boldsymbol{X}] \\
\text{(unitR)} & \boldsymbol{E} \gg\!= \boldsymbol{y}.\text{return}(\boldsymbol{y}) & \Rightarrow & \boldsymbol{E} \\
\text{(assoc)} & (\boldsymbol{E} \gg\!= \boldsymbol{x}.\boldsymbol{K}[\boldsymbol{x}]) \gg\!= \boldsymbol{y}.\boldsymbol{L}[\boldsymbol{y}] & \Rightarrow & \boldsymbol{E} \gg\!= \boldsymbol{x}.(\boldsymbol{K}[\boldsymbol{x}] \gg\!= \boldsymbol{y}.\boldsymbol{L}[\boldsymbol{y}])
\end{array}$$

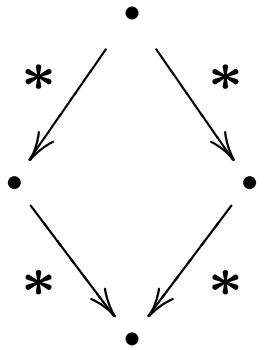
- ▷ Write the free variables in capitals

Two Important Rewriting Properties

CR

Confluence

Church-Rosser



SN

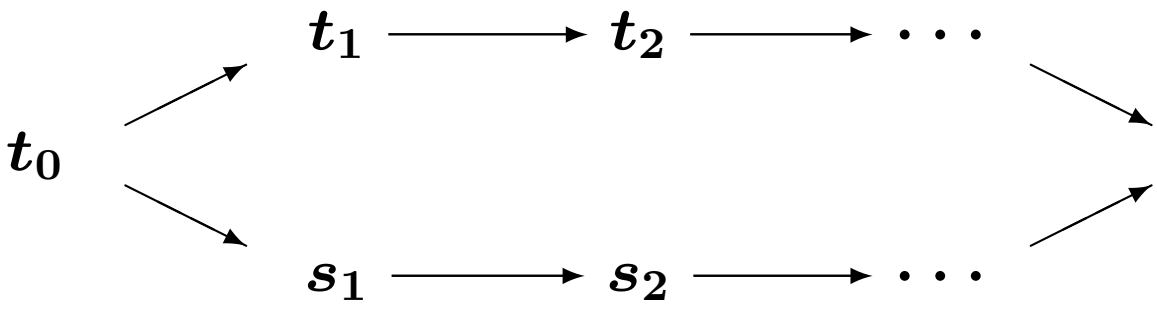
Strong Normalisation

finite

Basic Idea

If a rewrite system C has the properties of CR & SN

1. There always exist unique normal forms



(Normal form)

2. Gives a decidable proof method of arbitrary equation

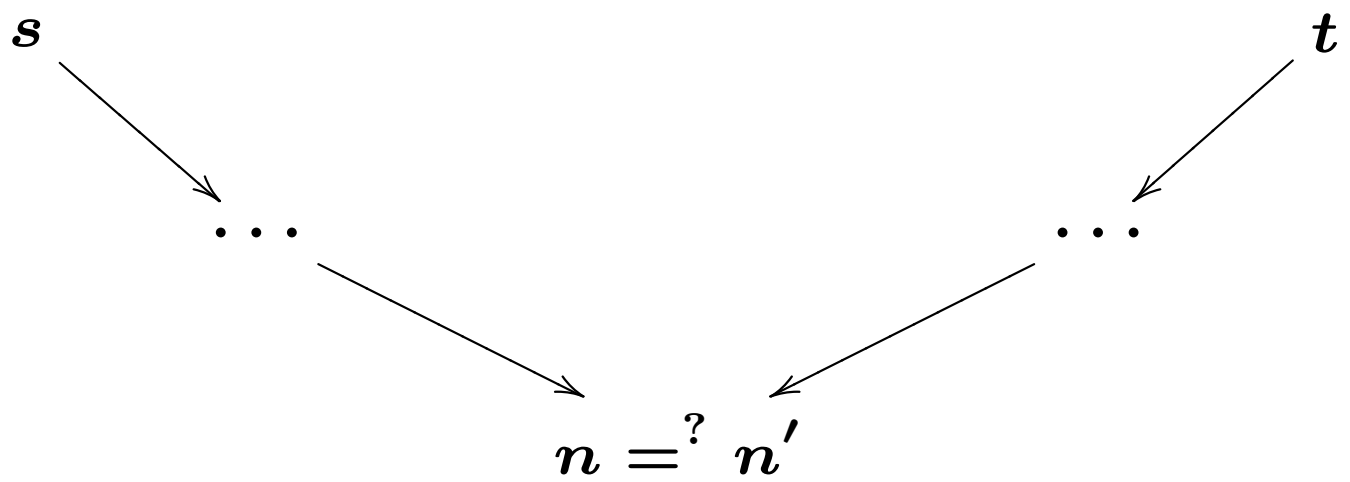
$$C \vdash s \stackrel{?}{=} t \quad \Leftrightarrow \quad \text{proof by rewriting}$$

Basic Idea: Proof by Rewriting

To prove

$$C \vdash s = t$$

is equivalent to showing



Normal forms

.

.

How to prove CR and SN?

How to Prove Monad Laws are SN (i.e. Terminating)

$$\begin{array}{lrcl}
\text{(unitL)} & \mathsf{return}(\boldsymbol{X}) \gg\!= \boldsymbol{y}.\boldsymbol{K}[\boldsymbol{y}] & \Rightarrow & \boldsymbol{K}[\boldsymbol{X}] \\
\text{(unitR)} & \boldsymbol{E} \gg\!= \boldsymbol{y}.\mathsf{return}(\boldsymbol{y}) & \Rightarrow & \boldsymbol{E} \\
\text{(assoc)} & (\boldsymbol{E} \gg\!= \boldsymbol{x}.\boldsymbol{K}[\boldsymbol{x}]) \gg\!= \boldsymbol{y}.\boldsymbol{L}[\boldsymbol{y}] & \Rightarrow & \boldsymbol{E} \gg\!= \boldsymbol{x}.(\boldsymbol{K}[\boldsymbol{x}] \gg\!= \boldsymbol{y}.\boldsymbol{L}[\boldsymbol{y}])
\end{array}$$

- ▷ Bad: No simple "weight" for term
- ▷ Standard: Reducibility method [Tait, Girard; Lindley, Stark'05...]
  – how to find suitable reducibility predicate
- ▷ SOL: The General Schema criterion by Blanqui [RTA'00,TCS'16]
  – based on computability
  – just checks (various) syntactic conditions
  positivity, accessibility, safe use of recursive calls, metavariables

How to Prove Monad Laws are SN by SOL

Step 1. Write a Haskell script for describing rewrite rules and
. signature using SOL's DSL (by Template Haskell)

```
monad = [rule|
 (unitL)  return(X) >>= y.K[y]       => K[X]
 (unitR)  E >>= y.return(y)          => E
 (assoc)  (E >>= x.K[x]) >>= y.L[y] => E >>= x.(K[x] >>= y.L[y])
 |]
```

How to Prove Monad Laws are SN by SOL

Step 2. Invoke the SOL's command `sn` in GHCi

```
*SOL> sn monad sigm

Found constructors: return
Checking type order >>OK
Checking positivity of constructors >>OK
Checking function dependency >>OK
Checking  (unitL) return(X) >>=y.K[y] => K[X]
 (meta K)[is acc in return(X),y.K[y]] [is positive in return(X)] [is acc in K[y]]
 (meta X)[is acc in return(X),y.K[y]] [is positive in return(X)] [is acc in X]  >>True
Checking  (unitR) N >>=y.return(y) => N
 (meta N)[is acc in N,y.return(y)] [is acc in N]  >>True
Checking  (assoc) (N >>=x.K[x]) >>=y.L[y] => N >>=x.(K[x] >>=y.L[y])
 (fun bind=bind) subterm comparison of args w. lexLR
 (meta N)[is acc in N >>=x.K[x],y.L[y]] [is positive in N >>=x.K[x]] [is acc in N]
 (fun bind=bind) subterm comparison of args w. lexLR
 (meta K)[is acc in N >>=x.K[x],y.L[y]] [is positive in N >>=x.K[x]] [is acc in K[x]]
 (meta L)[is acc in N >>=x.K[x],y.L[y]] [is positive in N >>=x.K[x]] [is acc in L[y]]  >>True
#SN!
```

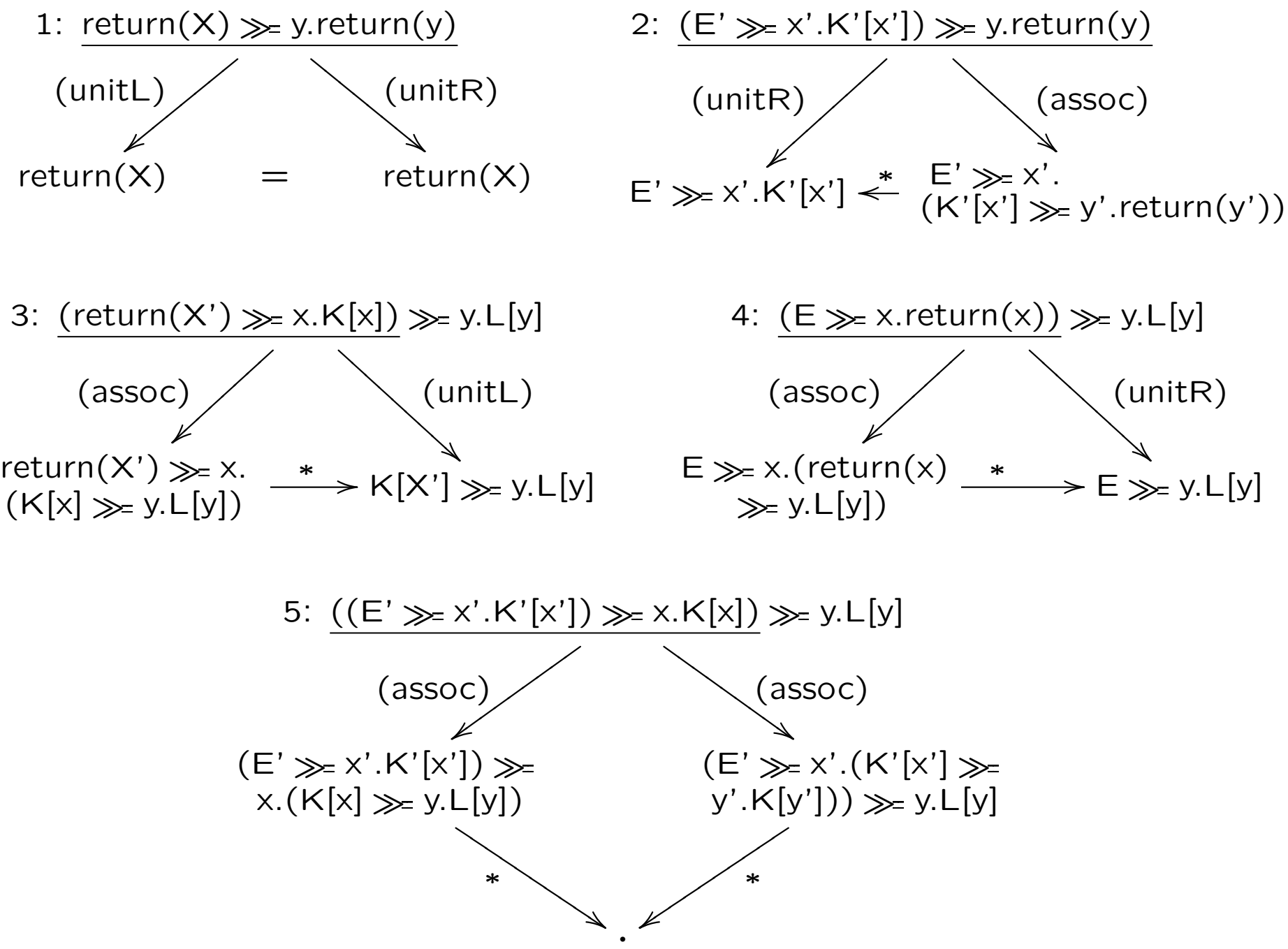
This shows termination.

How to Prove Monad Laws are CR by SOL

Step 3. Invoke the command `cri` for critical pair checking

```
*SOL> cri monad

1: Overlap (unitL)-(unitR)--- N'|-> return(X), K|-> z1.return(z1) ----------------
   L: (return(X) >>=y.K[y]) => K[X]
   R: N' >>=y'.return(y') => N'
                                    return(X) >>=y.return(y)
                                return(X) <-(unitL)-∧-(unitR)-> return(X)
                                    ---> return(X) =OK= return(X) <---
..
5: Overlap (assoc)-(assoc)--- N|-> N' >>=x'.K'[x'], L'|-> z1.K[z1] ---------------
   L: (N >>=x.K[x]) >>=y.L[y] => N >>=x.(K[x] >>=y.L[y])
   R: (N' >>=x'.K'[x']) >>=y'.L'[y'] => N' >>=x'.(K'[x'] >>=y'.L'[y'])
                                  ((N' >>=x'.K'[x']) >>=x.K[x]) >>=y.L[y]
 (N' >>=x'.K'[x']) >>=x.(K[x] >>=y.L[y]) <-(assoc)-∧-(assoc)-> (N' >>=x'.(K'[x'] >>=y'.K[y']))
    ---> N' >>=x.(K'[x] >>=y.(K[y] >>=y.L[y])) =E= N' >>=x.(K'[x] >>=x.(K[x] >>=y.L[y])) <---
#Joinable! (Total 5 CPs)
```



This shows local confluence, and with SN and Newman's lemma, we have CR.

Example: Monad

A monad T is a structure with two operations

$$\text{return} : a \to Ta \qquad\qquad \gg\!= \; : Ta \to (a \to Tb) \to Tb$$

satisfying

$$\begin{array}{lrcl}
\text{(unitL)} & \text{return}(x) \gg\!= \lambda y.k\,y & = & k\,x \\
\text{(unitR)} & e \gg\!= \lambda y.\text{return}(y) & = & e \\
\text{(assoc)} & (e \gg\!= \lambda x.k\,x) \gg\!= \lambda y.\ell\,y & = & e \gg\!= \lambda x.(k\,x \gg\!= \lambda y.\ell\,y)
\end{array}$$

The theory of monad is **decidable**.

Results: Decidability of Various Calculi — ICFP side

- Monad laws
- Algebraic theory of global state [Plotkin,Power'02]
- Various $\boldsymbol{\lambda}$-calculus: Call-by-name, Call-by-value [Plotkin TCS'75]
  Computational meta-language $\boldsymbol{\lambda_{ml}}$ [Moggi'88]
  Monadic calculus $\boldsymbol{\lambda_{ml*}}$ [Sabry,Wadler TOPLAS'97]
- Hasegawa's linear $\boldsymbol{\lambda}$-calculus [Ohta,Hasegwa RTA'06]
- Linear Term Calculus [Benton JFP'95]
- Algebraic theory of $\boldsymbol{\pi}$-calculus [Stark TCS'08]
- Asymmetric lens laws [Foster et al. TOPLAS'07]

How to Prove Your Calculus Is Decidable

Let's **Prove Your Calculus is Decidable** using

SOL

– I'm happy to assist you. Ask me directly, by email or Slack –