Proving Call-by-Value Termination of Constrained Higher-Order Rewriting by Dependency Pairs

Carsten Fuhs[1] Liye Guo[2] Cynthia Kop[2]

[1] Birkbeck, University of London

[2] Radboud University Nijmegen

Deduktionstreffen 2025
Stuttgart, Germany
1 August 2025

Proving Call-by-Value Termination of Constrained Higher-Order Rewriting by Dependency Pairs: Overview

Proving program termination:

Proving Call-by-Value Termination of Constrained Higher-Order Rewriting by Dependency Pairs: Overview

Proving program termination:

Many translations in the literature

- Prolog [van Raamsdonk, *ICLP '97*], [Giesl et al, *PPDP '12*]
- Java [Otto et al, *RTA '10*]
- Haskell [Giesl et al, *TOPLAS '11*]
- LLVM [Ströder et al, *JAR '17*]
- C [Fuhs, Kop, Nishida, *TOCL '17*]
- Jinja [Moser, Schaper, *IC '18*]
- Scala [Milovančević, Fuhs, Kunčak, *WPTE '25*]
- ...

Proving Call-by-Value Termination of Constrained Higher-Order Rewriting by Dependency Pairs: Overview

Proving program termination:

Proving Call-by-Value Termination of Constrained Higher-Order Rewriting by Dependency Pairs: Overview

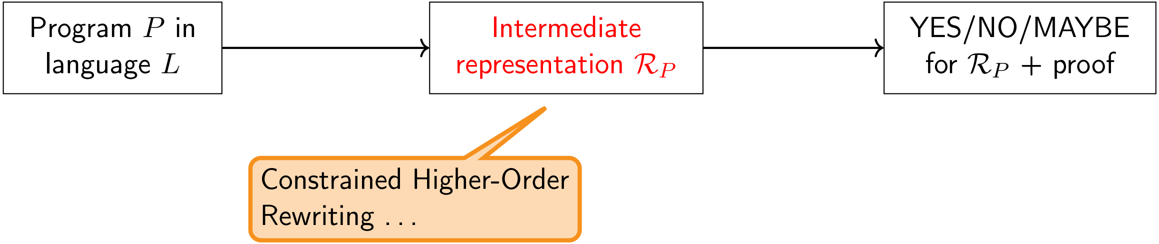
Proving program termination:

Intermediate representations based on

- Term Rewriting Systems: TRSs
- Integer Transition Systems: ITSs
- combinations and extensions: constrained rewriting

Proving Call-by-Value Termination of Constrained Higher-Order Rewriting by Dependency Pairs: Overview

Proving program termination:

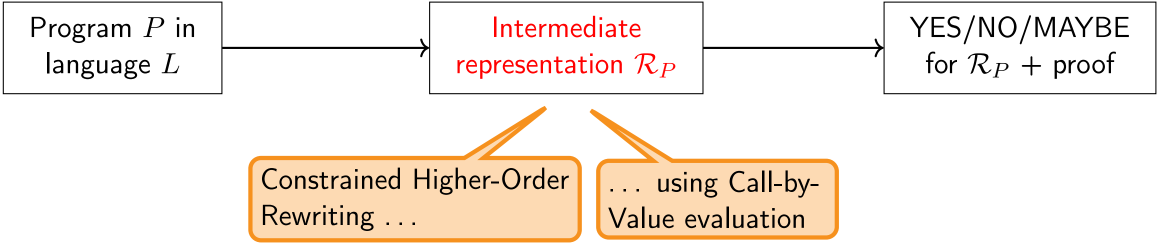


Intermediate representations based on

- Term Rewriting Systems: TRSs
- Integer Transition Systems: ITSs
- combinations and extensions: constrained rewriting

Proving Call-by-Value Termination of Constrained Higher-Order Rewriting by Dependency Pairs: Overview

Proving program termination:

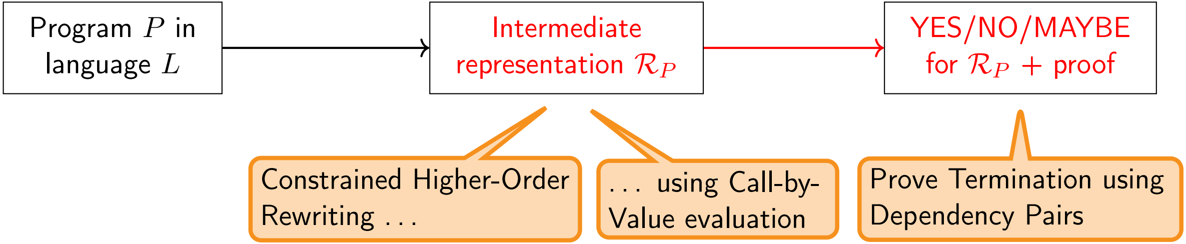


Intermediate representations based on

- Term Rewriting Systems: TRSs
- Integer Transition Systems: ITSs
- combinations and extensions: constrained rewriting

Proving Call-by-Value Termination of Constrained Higher-Order Rewriting by Dependency Pairs: Overview

Proving program termination:



Intermediate representations based on

- Term Rewriting Systems: TRSs
- Integer Transition Systems: ITSs
- combinations and extensions: constrained rewriting

Our intermediate representation: LCSTRSs

Ideal intermediate representation should

- be good for automated reasoning (no “lost in encoding”)
- express language features directly

Our intermediate representation: LCSTRSs

Ideal intermediate representation should

- be good for automated reasoning (no “lost in encoding”)
- express language features directly

What is available?

Our intermediate representation: LCSTRSs

Ideal intermediate representation should

- be good for automated reasoning (no "lost in encoding")
- express language features directly

$$\mathsf{length}\ \mathsf{nil} \to \mathsf{zero} \quad \mid \quad \mathsf{length}\ (\mathsf{cons}\ x\ xs) \to \mathsf{s}\ (\mathsf{length}\ xs)$$
$$\mathsf{plus}\ x\ \mathsf{zero} \to x \quad \mid \quad \mathsf{plus}\ x\ (\mathsf{s}\ y) \to \mathsf{s}\ (\mathsf{plus}\ x\ y)$$

What is available?

- Term Rewriting Systems aka TRSs: functions on algebraic data structures

Our intermediate representation: LCSTRSs

Ideal intermediate representation should

- be good for automated reasoning (no "lost in encoding")
- express language features directly

$$\mathsf{length}\ \mathsf{nil} \to \mathsf{zero} \qquad \mathsf{length}\ (\mathsf{cons}\ x\ xs) \to \mathsf{s}\ (\mathsf{length}\ xs)$$
$$\mathsf{plus}\ x\ \mathsf{zero} \to x \qquad \mathsf{plus}\ x\ (\mathsf{s}\ y) \to \mathsf{s}\ (\mathsf{plus}\ x\ y)$$

What is available?

- Term Rewriting Systems aka TRSs: functions on algebraic data structures
- Integer Transition Systems aka ITSs: functions/statements on integer data + arithmetic

$$\mathsf{gcd}\ m\ n \to \mathsf{gcd}\ (-m)\ n\ [m < 0] \qquad \mathsf{gcd}\ m\ n \to \mathsf{gcd}\ m\ (-n)\ [n < 0]$$
$$\mathsf{gcd}\ m\ 0 \to m\ [m \geq 0] \qquad \mathsf{gcd}\ m\ n \to \mathsf{gcd}\ n\ (m \bmod n)\ [m \geq 0 \wedge n > 0]$$

Our intermediate representation: LCSTRSs

Ideal intermediate representation should

- be good for automated reasoning (no "lost in encoding")
- express language features directly

$$\mathsf{length}\ \mathsf{nil} \to \mathsf{zero} \qquad \mathsf{length}\ (\mathsf{cons}\ x\ xs) \to \mathsf{s}\ (\mathsf{length}\ xs)$$
$$\mathsf{plus}\ x\ \mathsf{zero} \to x \qquad \mathsf{plus}\ x\ (\mathsf{s}\ y) \to \mathsf{s}\ (\mathsf{plus}\ x\ y)$$

What is available?

- Term Rewriting Systems aka TRSs: functions on algebraic data structures
- Integer Transition Systems aka ITSs: functions/statements on integer data + arithmetic

$$\mathsf{gcd}\ m\ n \to \mathsf{gcd}\ (-m)\ n\ [m < 0] \qquad \mathsf{gcd}\ m\ n \to \mathsf{gcd}\ m\ (-n)\ [n < 0]$$
$$\mathsf{gcd}\ m\ 0 \to m\ [m \geq 0] \qquad \mathsf{gcd}\ m\ n \to \mathsf{gcd}\ n\ (m \bmod n)\ [m \geq 0 \wedge n > 0]$$

- Logically Constrained TRSs aka LCTRSs [Kop, Nishida, *FroCoS '13*]:
  TRSs + ITSs + arbitrary logical theories (arrays, bitvectors, . . .)

Our intermediate representation: LCSTRSs

Ideal intermediate representation should

- be good for automated reasoning (no "lost in encoding")
- express language features directly

length nil → zero	length (cons x xs) → s (length xs)
plus x zero → x	plus x (s y) → s (plus x y)

What is available?

- Term Rewriting Systems aka TRSs: functions on algebraic data structures
- Integer Transition Systems aka ITSs: functions/statements on integer data + arithmetic

gcd m n → gcd $(-m)$ n $[m < 0]$	gcd m n → gcd m $(-n)$ $[n < 0]$
gcd m 0 → m $[m \geq 0]$	gcd m n → gcd n $(m \bmod n)$ $[m \geq 0 \wedge n > 0]$

- Logically Constrained TRSs aka LCTRSs [Kop, Nishida, *FroCoS '13*]: TRSs + ITSs + arbitrary logical theories (arrays, bitvectors, . . .)
- Logically Constrained Simply-typed TRSs aka LCSTRSs [Guo, Kop, *ESOP '24*]: LCTRSs + higher-order functions (but no λ)

gcdlist : intlist → int, fold : (int → int → int) → int → intlist → int

gcdlist → fold gcd 0 | fold f y nil → y | fold f y (cons x l) → f x (fold f y l)

Call-by-value (cbv) and innermost rewriting for LCSTRSs

Evaluating with an LCSTRS

$$\mathsf{fact}\ 0 \to 1$$
$$\mathsf{fact}\ x \to x * \mathsf{fact}\ (x - 1)\ [x > 0]$$
$$\mathsf{g}\ x \to \mathsf{g}\ (\mathsf{fact}\ {-1})$$

Call-by-value (cbv) and innermost rewriting for LCSTRSs

Evaluating with an LCSTRS

$\mathsf{fact}\ 0 \to 1$
$\mathsf{fact}\ x \to x * \mathsf{fact}\ (x-1)\ [x > 0]$
$\mathsf{g}\ x \to \mathsf{g}\ (\mathsf{fact}\ {-1})$

Cbv rewriting

Proper subterms of redex:
ground values

$$\mathsf{g}\ \underline{(\mathsf{fact}\ 1)} \xrightarrow{\text{v}} \mathsf{g}\ (1 * \underline{\mathsf{fact}\ 0}) \xrightarrow{\text{v}} \mathsf{g}\ \underline{(1*1)} \xrightarrow{\text{v}} \underline{\mathsf{g}\ 1} \xrightarrow{\text{v}} \mathsf{g}\ (\mathsf{fact}\ {-1}) \not\xrightarrow{\text{v}}$$

Call-by-value (cbv) and innermost rewriting for LCSTRSs

Evaluating with an LCSTRS

$\mathsf{fact}\ 0 \to 1$
$\mathsf{fact}\ x \to x * \mathsf{fact}\ (x - 1)\ [x > 0]$
$\mathsf{g}\ x \to \mathsf{g}\ (\mathsf{fact}\ {-1})$

Cbv rewriting

Proper subterms of redex: ground values

$$\mathsf{g}\ \underline{(\mathsf{fact}\ 1)} \xrightarrow{v} \mathsf{g}\ (1 * \underline{\mathsf{fact}\ 0}) \xrightarrow{v} \mathsf{g}\ \underline{(1 * 1)} \xrightarrow{v} \underline{\mathsf{g}\ 1} \xrightarrow{v} \mathsf{g}\ (\mathsf{fact}\ {-1}) \not\xrightarrow{v}$$

- Cbv: used in programming languages
- Want to prove termination of cbv rewriting!

Call-by-value (cbv) and innermost rewriting for LCSTRSs

Evaluating with an LCSTRS

$$\mathsf{fact}\ 0 \to 1$$
$$\mathsf{fact}\ x \to x * \mathsf{fact}\ (x - 1)\ [x > 0]$$
$$\mathsf{g}\ x \to \mathsf{g}\ (\mathsf{fact}\ {-1})$$

Cbv rewriting

Proper subterms of redex: ground values

$$\mathsf{g}\ \underline{(\mathsf{fact}\ 1)} \xrightarrow{\text{v}} \mathsf{g}\ (1 * \underline{\mathsf{fact}\ 0})$$
$$\xrightarrow{\text{v}} \mathsf{g}\ \underline{(1 * 1)} \xrightarrow{\text{v}} \underline{\mathsf{g}\ 1}$$
$$\xrightarrow{\text{v}} \mathsf{g}\ (\mathsf{fact}\ {-1}) \not\xrightarrow{\text{v}}$$

- Cbv: used in programming languages
- Want to prove termination of cbv rewriting!
- Literature on termination: focus on innermost rewriting

Call-by-value (cbv) and innermost rewriting for LCSTRSs

Evaluating with an LCSTRS

$$\mathsf{fact}\ 0 \to 1$$
$$\mathsf{fact}\ x \to x * \mathsf{fact}\ (x-1)\ [x > 0]$$
$$\mathsf{g}\ x \to \mathsf{g}\ (\mathsf{fact}\ {-1})$$

Cbv rewriting

Proper subterms of redex: ground values

$$\mathsf{g}\ \underline{(\mathsf{fact}\ 1)} \xrightarrow{\text{v}} \mathsf{g}\ (1 * \underline{\mathsf{fact}\ 0}) \xrightarrow{\text{v}} \mathsf{g}\ \underline{(1*1)} \xrightarrow{\text{v}} \underline{\mathsf{g}\ 1} \xrightarrow{\text{v}} \mathsf{g}\ (\mathsf{fact}\ {-1}) \not\xrightarrow{\text{v}}$$

Innermost rewriting

Proper subterms of redex: normal forms

$$\mathsf{g}\ \underline{(\mathsf{fact}\ 1)} \xrightarrow{\text{i}} \mathsf{g}\ (1 * \underline{\mathsf{fact}\ 0}) \xrightarrow{\text{i}} \mathsf{g}\ \underline{(1*1)} \xrightarrow{\text{i}} \underline{\mathsf{g}\ 1} \xrightarrow{\text{i}} \underline{\mathsf{g}\ (\mathsf{fact}\ {-1})} \xrightarrow{\text{i}} \underline{\mathsf{g}\ (\mathsf{fact}\ {-1})} \xrightarrow{\text{i}} \dots$$

- Cbv: used in programming languages
- Want to prove termination of cbv rewriting!
- Literature on termination: focus on innermost rewriting

Call-by-value (cbv) and innermost rewriting for LCSTRSs

Evaluating with an LCSTRS

$$\mathsf{fact}\ 0 \to 1$$
$$\mathsf{fact}\ x \to x * \mathsf{fact}\ (x - 1)\ [x > 0]$$
$$\mathsf{g}\ x \to \mathsf{g}\ (\mathsf{fact}\ -1)$$

Cbv rewriting

Proper subterms of redex: ground values

$$\mathsf{g}\ \underline{(\mathsf{fact}\ 1)} \xrightarrow{\text{v}} \mathsf{g}\ (1 * \underline{\mathsf{fact}\ 0}) \xrightarrow{\text{v}} \mathsf{g}\ \underline{(1 * 1)} \xrightarrow{\text{v}} \underline{\mathsf{g}\ 1} \xrightarrow{\text{v}} \mathsf{g}\ (\mathsf{fact}\ -1) \not\xrightarrow{\text{v}}$$

Innermost rewriting

Proper subterms of redex: normal forms

$$\mathsf{g}\ \underline{(\mathsf{fact}\ 1)} \xrightarrow{\text{i}} \mathsf{g}\ (1 * \underline{\mathsf{fact}\ 0}) \xrightarrow{\text{i}} \mathsf{g}\ \underline{(1 * 1)} \xrightarrow{\text{i}} \underline{\mathsf{g}\ 1} \xrightarrow{\text{i}} \underline{\mathsf{g}\ (\mathsf{fact}\ -1)} \xrightarrow{\text{i}} \underline{\mathsf{g}\ (\mathsf{fact}\ -1)} \xrightarrow{\text{i}} \dots$$

- Cbv: used in programming languages
- Want to prove termination of cbv rewriting!
- Literature on termination: focus on innermost rewriting
- Solution [Fuhs, Guo, Kop, *FSCD '25*]: mention $x : \mathsf{int}$ in constraint $\Rightarrow$ x must be value!
- int is inextensible theory sort

Call-by-value (cbv) and innermost rewriting for LCSTRSs

Evaluating with an LCSTRS

$$\mathsf{fact}\ 0 \to 1$$
$$\mathsf{fact}\ x \to x * \mathsf{fact}\ (x-1)\ [x > 0]$$
$$\mathsf{g}\ x \to \mathsf{g}\ (\mathsf{fact}\ -1)$$

Cbv rewriting

Proper subterms of redex: ground values

$$\mathsf{g}\ \underline{(\mathsf{fact}\ 1)} \xrightarrow{\text{v}} \mathsf{g}\ (1 * \underline{\mathsf{fact}\ 0}) \xrightarrow{\text{v}} \mathsf{g}\ \underline{(1*1)} \xrightarrow{\text{v}} \underline{\mathsf{g}\ 1} \xrightarrow{\text{v}} \mathsf{g}\ (\mathsf{fact}\ -1) \not\xrightarrow{\text{v}}$$

Innermost rewriting

Proper subterms of redex: normal forms

$$\mathsf{g}\ \underline{(\mathsf{fact}\ 1)} \xrightarrow{\text{i}} \mathsf{g}\ (1 * \underline{\mathsf{fact}\ 0}) \xrightarrow{\text{i}} \mathsf{g}\ \underline{(1*1)} \xrightarrow{\text{i}} \underline{\mathsf{g}\ 1} \xrightarrow{\text{i}} \underline{\mathsf{g}\ (\mathsf{fact}\ -1)} \xrightarrow{\text{i}} \underline{\mathsf{g}\ (\mathsf{fact}\ -1)} \xrightarrow{\text{i}} \ldots$$

- Cbv: used in programming languages
- Want to prove termination of cbv rewriting!
- Literature on termination: focus on innermost rewriting
- Solution [Fuhs, Guo, Kop, *FSCD '25*]: mention x : int in constraint ⇒ x must be value!
- int is inextensible theory sort

$$\mathsf{fact}\ 0 \to 1$$
$$\mathsf{fact}\ x \to x * \mathsf{fact}\ (x-1)\ [x > 0]$$
$$\mathsf{g}\ x \to \mathsf{g}\ (\mathsf{fact}\ -1)\quad [x \equiv x]$$

Call-by-value (cbv) and innermost rewriting for LCSTRSs

Evaluating with an LCSTRS

$\mathsf{fact}\ 0 \to 1$
$\mathsf{fact}\ x \to x * \mathsf{fact}\ (x - 1)\ [x > 0]$
$\mathsf{g}\ x \to \mathsf{g}\ (\mathsf{fact}\ {-1})$

Cbv rewriting

Proper subterms of redex: ground values

$$\mathsf{g}\ \underline{(\mathsf{fact}\ 1)} \xrightarrow{v} \mathsf{g}\ (1 * \underline{\mathsf{fact}\ 0}) \xrightarrow{v} \mathsf{g}\ \underline{(1 * 1)} \xrightarrow{v} \underline{\mathsf{g}\ 1} \xrightarrow{v} \mathsf{g}\ (\mathsf{fact}\ {-1}) \not\xrightarrow{v}$$

Innermost rewriting

Proper subterms of redex: normal forms

$$\mathsf{g}\ \underline{(\mathsf{fact}\ 1)} \xrightarrow{i} \mathsf{g}\ (1 * \underline{\mathsf{fact}\ 0}) \xrightarrow{i} \mathsf{g}\ \underline{(1 * 1)} \xrightarrow{i} \underline{\mathsf{g}\ 1} \xrightarrow{i} \underline{\mathsf{g}\ (\mathsf{fact}\ {-1})} \xrightarrow{i} \underline{\mathsf{g}\ (\mathsf{fact}\ {-1})} \xrightarrow{i} \dots$$

- Cbv: used in programming languages
- Want to prove termination of cbv rewriting!
- Literature on termination: focus on innermost rewriting
- Solution [Fuhs, Guo, Kop, *FSCD '25*]: mention $x : \mathsf{int}$ in constraint $\Rightarrow$ x must be value!
- int is inextensible theory sort

$\mathsf{fact}\ 0 \to 1$
$\mathsf{fact}\ x \to x * \mathsf{fact}\ (x - 1)\ [x > 0]$
$\mathsf{g}\ x \to \mathsf{g}\ (\mathsf{fact}\ {-1}) \quad [x \equiv x]$

$\Rightarrow$ Terminates also for innermost rewriting!

Static Dependency Pairs

$\mathcal{R}_{\text{gcd}}$

$\text{gcdlist} \to \text{fold gcd } 0$

$\text{fold } f\ y\ \text{nil} \to y \quad [y \equiv y] \quad | \quad \text{fold } f\ y\ (\text{cons } x\ l) \to f\ x\ (\text{fold } f\ y\ l) \quad [x \equiv x \wedge y \equiv y]$

$\text{gcd } m\ n \to \text{gcd } (-m)\ n\ [m < 0 \wedge n \equiv n] \quad | \quad \text{gcd } m\ n \to \text{gcd } m\ (-n)\ [n < 0 \wedge m \equiv m]$

$\text{gcd } m\ 0 \to m\ [m \geq 0] \quad | \quad \text{gcd } m\ n \to \text{gcd } n\ (m \bmod n)\ [m \geq 0 \wedge n > 0]$

Static Dependency Pairs

$\mathcal{R}_{\mathsf{gcd}}$

$\mathsf{gcdlist} \to \mathsf{fold}\ \mathsf{gcd}\ 0$

$\mathsf{fold}\ f\ y\ \mathsf{nil} \to y\quad [y \equiv y]$ | $\mathsf{fold}\ f\ y\ (\mathsf{cons}\ x\ l) \to f\ x\ (\mathsf{fold}\ f\ y\ l)\quad [x \equiv x \land y \equiv y]$

$\mathsf{gcd}\ m\ n \to \mathsf{gcd}\ (-m)\ n\ [m < 0 \land n \equiv n]$ | $\mathsf{gcd}\ m\ n \to \mathsf{gcd}\ m\ (-n)\quad [n < 0 \land m \equiv m]$

$\mathsf{gcd}\ m\ 0 \to m\quad [m \geq 0]$ | $\mathsf{gcd}\ m\ n \to \mathsf{gcd}\ n\ (m \bmod n)\ [m \geq 0 \land n > 0]$

Prove termination by Static Dependency Pairs for LCSTRSs [Guo, Hagens, Kop, Vale, *MFCS '24*]

- For LCSTRS $\mathcal{R}$ build dependency pairs $\mathcal{P} = \mathsf{SDP}(\mathcal{R}_{\mathsf{gcd}})$ (∼ function calls)

Static Dependency Pairs

$\mathcal{R}_{\text{gcd}}$

$\mathsf{gcdlist} \to \mathsf{fold}\ \mathsf{gcd}\ 0$

$\mathsf{fold}\ f\ y\ \mathsf{nil} \to y\ \ [y \equiv y]\quad |\quad \mathsf{fold}\ f\ y\ (\mathsf{cons}\ x\ l) \to f\ x\ (\mathsf{fold}\ f\ y\ l)\ \ [x \equiv x \wedge y \equiv y]$

$\mathsf{gcd}\ m\ n \to \mathsf{gcd}\ (-m)\ n\ [m < 0 \wedge n \equiv n]\quad |\quad \mathsf{gcd}\ m\ n \to \mathsf{gcd}\ m\ (-n)\ [n < 0 \wedge m \equiv m]$

$\mathsf{gcd}\ m\ 0 \to m\ [m \geq 0]\quad |\quad \mathsf{gcd}\ m\ n \to \mathsf{gcd}\ n\ (m \bmod n)\ [m \geq 0 \wedge n > 0]$

Prove termination by Static Dependency Pairs for LCSTRSs [Guo, Hagens, Kop, Vale, *MFCS '24*]

- For LCSTRS $\mathcal{R}$ build dependency pairs $\mathcal{P} = \text{SDP}(\mathcal{R}_{\text{gcd}})$ ($\sim$ function calls)
- Show: No ∞ call sequence with $\mathcal{P}$ (eval of $\mathcal{P}$'s args via $\mathcal{R}$)

Static Dependency Pairs

$\mathcal{R}_{\text{gcd}}$

$\mathsf{gcdlist} \to \mathsf{fold}\ \mathsf{gcd}\ 0$

$\mathsf{fold}\ f\ y\ \mathsf{nil} \to y\ \ [y \equiv y]$ | $\mathsf{fold}\ f\ y\ (\mathsf{cons}\ x\ l) \to f\ x\ (\mathsf{fold}\ f\ y\ l)\ \ [x \equiv x \wedge y \equiv y]$

$\mathsf{gcd}\ m\ n \to \mathsf{gcd}\ (-m)\ n\ [m < 0 \wedge n \equiv n]$ | $\mathsf{gcd}\ m\ n \to \mathsf{gcd}\ m\ (-n)\ [n < 0 \wedge m \equiv m]$

$\mathsf{gcd}\ m\ 0 \to m\ [m \geq 0]$ | $\mathsf{gcd}\ m\ n \to \mathsf{gcd}\ n\ (m \bmod n)\ [m \geq 0 \wedge n > 0]$

Prove termination by Static Dependency Pairs for LCSTRSs [Guo, Hagens, Kop, Vale, *MFCS '24*]

- For LCSTRS $\mathcal{R}$ build dependency pairs $\mathcal{P} = \mathsf{SDP}(\mathcal{R}_{\text{gcd}})$ ($\sim$ function calls)
- Show: No ∞ call sequence with $\mathcal{P}$ (eval of $\mathcal{P}$'s args via $\mathcal{R}$)

$\mathsf{SDP}(\mathcal{R}_{\text{gcd}})$

$\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{fold}^\sharp\ \mathsf{gcd}\ 0\ l'$

$\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{gcd}^\sharp\ m'\ n'$

$\mathsf{fold}^\sharp\ f\ y\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{fold}^\sharp\ f\ y\ l\ [x \equiv x \wedge y \equiv y]$

$\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ (-m)\ n\ [m < 0 \wedge n \equiv n]$

$\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ m\ (-n)\ [n < 0 \wedge m \equiv m]$

$\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ n\ (m \bmod n)\ [m \geq 0 \wedge n > 0]$

Dependency Pair Framework

- Works on DP problems $(\mathcal{P}, \mathcal{R})$
- DP framework:

  $S := \{(\mathsf{SDP}(\mathcal{R}), \mathcal{R})\}$
  while $S = S' \uplus \{(\mathcal{P}, \mathcal{R})\}$
  $\qquad S := S' \cup \rho(\mathcal{P}, \mathcal{R})$ for a DP processor ρ
  print "YES"

Dependency Pair Framework

- Works on DP problems $(\mathcal{P}, \mathcal{R})$
- DP framework:

  $S := \{(\mathsf{SDP}(\mathcal{R}), \mathcal{R})\}$
  while $S = S' \uplus \{(\mathcal{P}, \mathcal{R})\}$
  $\quad\quad S := S' \cup \rho(\mathcal{P}, \mathcal{R})$ for a DP processor ρ
  print "YES"

Existing DP processors for LCSTRSs [Guo, Hagens, Kop, Vale, *MFCS '24*]

- Graph processor
- Subterm criterion processor
- Integer mapping processor

Dependency Pair Framework

- Works on DP problems $(\mathcal{P}, \mathcal{R})$
- DP framework:

$S := \{(\mathsf{SDP}(\mathcal{R}), \mathcal{R})\}$
while $S = S' \uplus \{(\mathcal{P}, \mathcal{R})\}$
    $S := S' \cup \rho(\mathcal{P}, \mathcal{R})$ for a DP processor ρ
print "YES"

Existing DP processors for LCSTRSs [Guo, Hagens, Kop, Vale, *MFCS '24*]

- Graph processor
- Subterm criterion processor
- Integer mapping processor

New **innermost** DP processors for LCSTRSs [Fuhs, Guo, Kop, *FSCD '25*]

- Usable rules processor
- Reduction pair processor with usable rules wrt argument filtering
- Chaining processor

Also for **compositional termination analysis** via universal computability!

Existing DP processors for LCSTRSs

$\mathcal{P}$

(1) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{fold}^\sharp\ \mathsf{gcd}\ 0\ l'$

(2) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{gcd}^\sharp\ m'\ n'$

(3) $\mathsf{fold}^\sharp\ f\ y\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{fold}^\sharp\ f\ y\ l\ [x \equiv x \wedge y \equiv y]$

(4) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ (-m)\ n\ [m < 0 \wedge n \equiv n]$

(5) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ m\ (-n)\ [n < 0 \wedge m \equiv m]$

(6) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ n\ (m \bmod n)\ [m \geq 0 \wedge n > 0]$

$\mathcal{P}$

(1) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{fold}^\sharp\ \mathsf{gcd}\ 0\ l'$

(2) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{gcd}^\sharp\ m'\ n'$

(3) $\mathsf{fold}^\sharp\ f\ y\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{fold}^\sharp\ f\ y\ l\ [x \equiv x \wedge y \equiv y]$

(4) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ (-m)\ n\ [m < 0 \wedge n \equiv n]$

(5) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ m\ (-n)\ [n < 0 \wedge m \equiv m]$

(6) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ n\ (m \bmod n)\ [m \geq 0 \wedge n > 0]$

$\mathcal{R}$

$\ldots$

$\mathcal{P}$

(1) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{fold}^\sharp\ \mathsf{gcd}\ 0\ l'$

(2) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{gcd}^\sharp\ m'\ n'$

(3) $\mathsf{fold}^\sharp\ f\ y\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{fold}^\sharp\ f\ y\ l\ [x \equiv x \wedge y \equiv y]$

(4) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ (-m)\ n\ [m < 0 \wedge n \equiv n]$

(5) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ m\ (-n)\ [n < 0 \wedge m \equiv m]$

(6) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ n\ (m \bmod n)\ [m \geq 0 \wedge n > 0]$

$\mathcal{R}$

...

- **Dependency Graph**:
  which calls may follow one another?

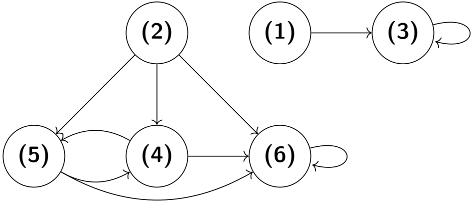
$\mathcal{P}$

(1) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{fold}^\sharp\ \mathsf{gcd}\ 0\ l'$
(2) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{gcd}^\sharp\ m'\ n'$
(3) $\mathsf{fold}^\sharp\ f\ y\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{fold}^\sharp\ f\ y\ l\ [x \equiv x \wedge y \equiv y]$
(4) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ (-m)\ n\ [m < 0 \wedge n \equiv n]$
(5) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ m\ (-n)\ [n < 0 \wedge m \equiv m]$
(6) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ n\ (m \bmod n)\ [m \geq 0 \wedge n > 0]$

$\mathcal{R}$

$\ldots$

- **Dependency Graph**:
  which calls may follow one another?
- Approximation
  [Guo, Hagens, Kop, Vale, *MFCS '24*]:



$\mathcal{P}$

(1) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{fold}^\sharp\ \mathsf{gcd}\ 0\ l'$

(2) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{gcd}^\sharp\ m'\ n'$

(3) $\mathsf{fold}^\sharp\ f\ y\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{fold}^\sharp\ f\ y\ l\ [x \equiv x \wedge y \equiv y]$

(4) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ (-m)\ n\ [m < 0 \wedge n \equiv n]$

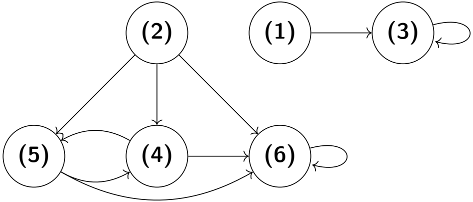
(5) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ m\ (-n)\ [n < 0 \wedge m \equiv m]$

(6) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ n\ (m \bmod n)\ [m \geq 0 \wedge n > 0]$

$\mathcal{R}$

...

- **Dependency Graph**: which calls may follow one another?
- Approximation [Guo, Hagens, Kop, Vale, *MFCS '24*]:



- **Graph processor**: decompose $\mathcal{P}$ into non-trivial Strongly Connected Components

$\mathcal{P}$

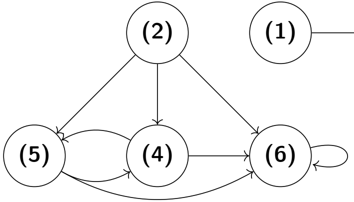
(1) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{fold}^\sharp\ \mathsf{gcd}\ 0\ l'$
(2) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{gcd}^\sharp\ m'\ n'$
(3) $\mathsf{fold}^\sharp\ f\ y\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{fold}^\sharp\ f\ y\ l\ [x \equiv x \wedge y \equiv y]$

(4) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ (-m)\ n\ [m < 0 \wedge n \equiv n]$
(5) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ m\ (-n)\ [n < 0 \wedge m \equiv m]$
(6) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ n\ (m \bmod n)\ [m \geq 0 \wedge n > 0]$

$\mathcal{R}$

...

- **Dependency Graph**: which calls may follow one another?
- Approximation [Guo, Hagens, Kop, Vale, *MFCS '24*]:



- **Graph processor**: decompose $\mathcal{P}$ into non-trivial Strongly Connected Components
- Here:

$$(\{\textbf{(3)}\}, \mathcal{R})$$
$$(\{\textbf{(6)}\}, \mathcal{R})$$
$$(\{\textbf{(4)}, \textbf{(5)}\}, \mathcal{R})$$

$\mathcal{P}$

(3) $\mathsf{fold}^\sharp\ f\ y\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{fold}^\sharp\ f\ y\ l\ [x \equiv x \wedge y \equiv y]$

$\mathcal{R}$

$\ldots$

$\mathcal{P}$

(3) $\mathsf{fold}^\sharp\ f\ y\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{fold}^\sharp\ f\ y\ l\ [x \equiv x \wedge y \equiv y]$

$\mathcal{R}$

$\ldots$

Subterm criterion processor [Guo, Hagens, Kop, Vale, *MFCS '24*]

- Detect structural decrease in argument
- Use projection $\nu(\mathsf{fold}^\sharp) = 3$
- Get $\mathsf{cons}\ x\ l \rhd l$

$\Rightarrow$ Remove **(3)**

$\Rightarrow$ $(\emptyset, \mathcal{R})$ deleted by graph processor

$\mathcal{P}$

(6) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ n\ (m \bmod n)\ [m \geq 0 \wedge n > 0]$

$\mathcal{R}$

$\ldots$

$\mathcal{P}$

(6) $\text{gcd}^\sharp\ m\ n \Rightarrow \text{gcd}^\sharp\ n\ (m \bmod n)\ [m \geq 0 \land n > 0]$

$\mathcal{R}$

$\ldots$

Integer mapping processor [Guo, Hagens, Kop, Vale, *MFCS '24*]

- Detect integer value decrease in argument
- Use projection $\nu(\text{gcd}^\sharp) = 2$
- Get $\quad m \geq 0 \land n > 0 \models n > m \bmod n$
  and $\quad m \geq 0 \land n > 0 \models n \geq 0$

$\Rightarrow$ Remove **(6)**

$\Rightarrow$ $(\emptyset, \mathcal{R})$ deleted by graph processor

$\mathcal{P}$

(6) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ n\ (m \bmod n)\ [m \geq 0 \wedge n > 0]$

$\mathcal{R}$

...

Integer mapping processor [Guo, Hagens, Kop, Vale, *MFCS '24*]

- Detect integer value decrease in argument
- Use projection $\nu(\mathsf{gcd}^\sharp) = 2$
- Get $m \geq 0 \wedge n > 0 \models n > m \bmod n$
  and $m \geq 0 \wedge n > 0 \models n \geq 0$

$\Rightarrow$ Remove **(6)**

$\Rightarrow$ $(\emptyset, \mathcal{R})$ deleted by graph processor

$(\{\mathbf{(4)}, \mathbf{(5)}\}, \mathcal{R})$ handled by integer mapping processor + graph processor

$\Rightarrow$ termination of $\mathcal{R}_{\mathsf{gcd}}$ proved!

New DP processors for LCSTRSs

Usable rules processor

$\mathcal{R}_{\mathsf{dfoldr}}$

drop : int → alist → alist
dfoldr : (a → b → b) → b → int → alist → b

drop n l	$\to$	l	$[n \leq 0]$
drop n nil	$\to$	nil	$[n \equiv n]$
drop n (cons x l)	$\to$	drop $(n - 1)$ l	$[n > 0]$
dfoldr f y n nil	$\to$	y	$[n \equiv n]$
dfoldr f y n (cons x l)	$\to$	f x (dfoldr f y n (drop n l))	$[n \equiv n]$

Usable rules processor

$\mathcal{R}_{\mathsf{dfoldr}}$

drop : int → alist → alist
dfoldr : (a → b → b) → b → int → alist → b

$$
\begin{array}{lll}
\mathsf{drop}\ n\ l & \to\ l & [n \leq 0] \\
\mathsf{drop}\ n\ \mathsf{nil} & \to\ \mathsf{nil} & [n \equiv n] \\
\mathsf{drop}\ n\ (\mathsf{cons}\ x\ l) & \to\ \mathsf{drop}\ (n-1)\ l & [n > 0]
\end{array}
$$

$$
\begin{array}{lll}
\mathsf{dfoldr}\ f\ y\ n\ \mathsf{nil} & \to\ y & [n \equiv n] \\
\mathsf{dfoldr}\ f\ y\ n\ (\mathsf{cons}\ x\ l) & \to\ f\ x\ (\mathsf{dfoldr}\ f\ y\ n\ (\mathsf{drop}\ n\ l)) & [n \equiv n]
\end{array}
$$

- Troublesome DP problem:

$$
(\ \{\ \mathsf{dfoldr}^{\sharp}\ f\ y\ n\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{dfoldr}^{\sharp}\ f\ y\ n\ (\mathsf{drop}\ n\ l)\ [n \equiv n]\ \},\ \mathcal{R}_{\mathsf{dfoldr}}\ )
$$

Usable rules processor

$\mathcal{R}_{\text{dfoldr}}$

drop : int → alist → alist
dfoldr : (a → b → b) → b → int → alist → b

$$
\begin{array}{lll}
\text{drop}\ n\ l & \to\ l & [n \leq 0] \\
\text{drop}\ n\ \text{nil} & \to\ \text{nil} & [n \equiv n] \\
\text{drop}\ n\ (\text{cons}\ x\ l) & \to\ \text{drop}\ (n-1)\ l & [n > 0] \\
\text{dfoldr}\ f\ y\ n\ \text{nil} & \to\ y & [n \equiv n] \\
\text{dfoldr}\ f\ y\ n\ (\text{cons}\ x\ l) & \to\ f\ x\ (\text{dfoldr}\ f\ y\ n\ (\text{drop}\ n\ l)) & [n \equiv n]
\end{array}
$$

- Troublesome DP problem:

$$
(\ \{\ \ \text{dfoldr}^\sharp\ f\ y\ n\ (\text{cons}\ x\ l) \Rightarrow \text{dfoldr}^\sharp\ f\ y\ n\ (\text{drop}\ n\ l)\ [n \equiv n]\ \ \},\ \ \mathcal{R}_{\text{dfoldr}}\ \ )
$$

- Reduction pair processor can show $\text{cons}\ x\ l\ \succ\ \text{drop}\ n\ l$
- But cannot show $\text{dfoldr}\ f\ y\ n\ (\text{cons}\ x\ l)\ \succsim\ f\ x\ (\text{dfoldr}\ f\ y\ n\ (\text{drop}\ n\ l))[n \equiv n]$

Usable rules processor

$\mathcal{R}_{\text{dfoldr}}$

$\text{drop} : \text{int} \to \text{alist} \to \text{alist}$
$\text{dfoldr} : (\text{a} \to \text{b} \to \text{b}) \to \text{b} \to \text{int} \to \text{alist} \to \text{b}$

$$\begin{array}{lcll}
\text{drop}\ n\ l & \to & l & [n \leq 0] \\
\text{drop}\ n\ \text{nil} & \to & \text{nil} & [n \equiv n] \\
\text{drop}\ n\ (\text{cons}\ x\ l) & \to & \text{drop}\ (n-1)\ l & [n > 0] \\
\text{dfoldr}\ f\ y\ n\ \text{nil} & \to & y & [n \equiv n] \\
\text{dfoldr}\ f\ y\ n\ (\text{cons}\ x\ l) & \to & f\ x\ (\text{dfoldr}\ f\ y\ n\ (\text{drop}\ n\ l)) & [n \equiv n]
\end{array}$$

- Troublesome DP problem:

$$(\ \{\ \text{dfoldr}^\sharp\ f\ y\ n\ (\text{cons}\ x\ l) \Rightarrow \text{dfoldr}^\sharp\ f\ y\ n\ (\text{drop}\ n\ l)\ [n \equiv n]\ \},\ \mathcal{R}_{\text{dfoldr}}\ )$$

- Reduction pair processor can show $\text{cons}\ x\ l \succ \text{drop}\ n\ l$
- But cannot show $\text{dfoldr}\ f\ y\ n\ (\text{cons}\ x\ l) \succsim f\ x\ (\text{dfoldr}\ f\ y\ n\ (\text{drop}\ n\ l))[n \equiv n]$
- **Usable rules processor**: keep only **usable** rules, called from DPs
- Here: rules for drop

Usable rules processor

$\mathcal{R}$

drop : int $\rightarrow$ alist $\rightarrow$ alist
dfoldr : (a $\rightarrow$ b $\rightarrow$ b) $\rightarrow$ b $\rightarrow$ int $\rightarrow$ alist $\rightarrow$ b

$$\begin{array}{lcll} \mathsf{drop}\ n\ l & \rightarrow & l & [n \leq 0] \\ \mathsf{drop}\ n\ \mathsf{nil} & \rightarrow & \mathsf{nil} & [n \equiv n] \\ \mathsf{drop}\ n\ (\mathsf{cons}\ x\ l) & \rightarrow & \mathsf{drop}\ (n-1)\ l & [n > 0] \end{array}$$

- Troublesome DP problem:

$$(\ \{\ \mathsf{dfoldr}^\sharp\ f\ y\ n\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{dfoldr}^\sharp\ f\ y\ n\ (\mathsf{drop}\ n\ l)\ [n \equiv n]\ \},\ \mathcal{R}_{\mathsf{dfoldr}}\ )$$

- Reduction pair processor can show $\mathsf{cons}\ x\ l \succ \mathsf{drop}\ n\ l$
- But cannot show $\mathsf{dfoldr}\ f\ y\ n\ (\mathsf{cons}\ x\ l) \succsim f\ x\ (\mathsf{dfoldr}\ f\ y\ n\ (\mathsf{drop}\ n\ l))[n \equiv n]$
- **Usable rules processor**: keep only **usable** rules, called from DPs
- Here: rules for drop

Reduction pair processor with usable rules wrt an argument filtering

$\mathcal{R}_{\mathsf{dfoldl}}$

$$
\begin{array}{lcll}
\mathsf{drop}\ n\ l & \to & l & [n \leq 0] \\
\mathsf{drop}\ n\ \mathsf{nil} & \to & \mathsf{nil} & [n \equiv n] \\
\mathsf{drop}\ n\ (\mathsf{cons}\ x\ l) & \to & \mathsf{drop}\ (n-1)\ l & [n > 0] \\
\\
\mathsf{dfoldl}\ f\ y\ n\ \mathsf{nil} & \to & y & [n \equiv n] \\
\mathsf{dfoldl}\ f\ y\ n\ (\mathsf{cons}\ x\ l) & \to & \mathsf{dfoldl}\ f\ (f\ y\ x)\ n\ (\mathsf{drop}\ n\ l) & [n \equiv n]
\end{array}
$$

Reduction pair processor with usable rules wrt an argument filtering

$\mathcal{R}_{\textsf{dfoldl}}$

$$
\begin{array}{lcll}
\textsf{drop}\ n\ l & \to & l & [n \le 0] \\
\textsf{drop}\ n\ \textsf{nil} & \to & \textsf{nil} & [n \equiv n] \\
\textsf{drop}\ n\ (\textsf{cons}\ x\ l) & \to & \textsf{drop}\ (n-1)\ l & [n > 0] \\
\textsf{dfoldl}\ f\ y\ n\ \textsf{nil} & \to & y & [n \equiv n] \\
\textsf{dfoldl}\ f\ y\ n\ (\textsf{cons}\ x\ l) & \to & \textsf{dfoldl}\ f\ (f\ y\ x)\ n\ (\textsf{drop}\ n\ l) & [n \equiv n]
\end{array}
$$

- Troublesome DP problem:

$$
(\ \{\ \textsf{dfoldl}^\sharp\ f\ y\ n\ (\textsf{cons}\ x\ l) \Rightarrow \textsf{dfoldl}^\sharp\ f\ (f\ y\ x)\ n\ (\textsf{drop}\ n\ l)\ [n \equiv n]\ \},\ \mathcal{R}_{\textsf{dfoldl}}\ )
$$

Reduction pair processor with usable rules wrt an argument filtering

$\mathcal{R}_{\mathsf{dfoldl}}$

$$
\begin{array}{lcll}
\mathsf{drop}\ n\ l & \to & l & [n \leq 0] \\
\mathsf{drop}\ n\ \mathsf{nil} & \to & \mathsf{nil} & [n \equiv n] \\
\mathsf{drop}\ n\ (\mathsf{cons}\ x\ l) & \to & \mathsf{drop}\ (n-1)\ l & [n > 0] \\
\mathsf{dfoldl}\ f\ y\ n\ \mathsf{nil} & \to & y & [n \equiv n] \\
\mathsf{dfoldl}\ f\ y\ n\ (\mathsf{cons}\ x\ l) & \to & \mathsf{dfoldl}\ f\ (f\ y\ x)\ n\ (\mathsf{drop}\ n\ l) & [n \equiv n]
\end{array}
$$

- Troublesome DP problem:

$$
(\ \{\ \mathsf{dfoldl}^\sharp\ f\ y\ n\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{dfoldl}^\sharp\ f\ (f\ y\ x)\ n\ (\mathsf{drop}\ n\ l)\ [n \equiv n]\ \},\ \mathcal{R}_{\mathsf{dfoldl}}\ )
$$

- All rules are usable!

Reduction pair processor with usable rules wrt an argument filtering

$\mathcal{R}_{\mathsf{dfoldl}}$

$\mathsf{drop}\ n\ l$	$\rightarrow$	l	$[n \leq 0]$
$\mathsf{drop}\ n\ \mathsf{nil}$	$\rightarrow$	nil	$[n \equiv n]$
$\mathsf{drop}\ n\ (\mathsf{cons}\ x\ l)$	$\rightarrow$	$\mathsf{drop}\ (n-1)\ l$	$[n > 0]$
$\mathsf{dfoldl}\ f\ y\ n\ \mathsf{nil}$	$\rightarrow$	y	$[n \equiv n]$
$\mathsf{dfoldl}\ f\ y\ n\ (\mathsf{cons}\ x\ l)$	$\rightarrow$	$\mathsf{dfoldl}\ f\ (f\ y\ x)\ n\ (\mathsf{drop}\ n\ l)$	$[n \equiv n]$

- Troublesome DP problem:

$$(\ \{\ \mathsf{dfoldl}^\sharp\ f\ y\ n\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{dfoldl}^\sharp\ f\ (f\ y\ x)\ n\ (\mathsf{drop}\ n\ l)\ [n \equiv n]\ \},\ \mathcal{R}_{\mathsf{dfoldl}}\ )$$

- All rules are usable!
- **Reduction pair processor with usable rules wrt argument filtering**:

  temporarily disregard arguments, calculate usable rules, use reduction pair (HORPO, . . .)

Reduction pair processor with usable rules wrt an argument filtering

$\mathcal{R}_{\mathsf{dfoldl}}$

$\mathsf{drop}\ n\ l$	$\rightarrow$	l	$[n \leq 0]$
$\mathsf{drop}\ n\ \mathsf{nil}$	$\rightarrow$	nil	$[n \equiv n]$
$\mathsf{drop}\ n\ (\mathsf{cons}\ x\ l)$	$\rightarrow$	$\mathsf{drop}\ (n-1)\ l$	$[n > 0]$
$\mathsf{dfoldl}\ f\ y\ n\ \mathsf{nil}$	$\rightarrow$	y	$[n \equiv n]$
$\mathsf{dfoldl}\ f\ y\ n\ (\mathsf{cons}\ x\ l)$	$\rightarrow$	$\mathsf{dfoldl}\ f\ (f\ y\ x)\ n\ (\mathsf{drop}\ n\ l)$	$[n \equiv n]$

- Troublesome DP problem:

$$(\ \{\ \mathsf{dfoldl}^\sharp\ f\ y\ n\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{dfoldl}^\sharp\ f\ (f\ y\ x)\ n\ (\mathsf{drop}\ n\ l)\ [n \equiv n]\ \},\ \mathcal{R}_{\mathsf{dfoldl}}\ )$$

- All rules are usable!
- **Reduction pair processor with usable rules wrt argument filtering**:

  temporarily disregard arguments, calculate usable rules, use reduction pair (HORPO, . . .)
- $\mathrm{regard}(\mathsf{dfoldl}^\sharp) = \{4\} \quad \Rightarrow$ use first-order RPO!

Reduction pair processor with usable rules wrt an argument filtering

$\mathcal{R}_{\mathsf{dfoldl}}$

$\mathsf{drop}\ n\ l$	$\to$	l	$[n \leq 0]$
$\mathsf{drop}\ n\ \mathsf{nil}$	$\to$	nil	$[n \equiv n]$
$\mathsf{drop}\ n\ (\mathsf{cons}\ x\ l)$	$\to$	$\mathsf{drop}\ (n-1)\ l$	$[n > 0]$
$\mathsf{dfoldl}\ f\ y\ n\ \mathsf{nil}$	$\to$	y	$[n \equiv n]$
$\mathsf{dfoldl}\ f\ y\ n\ (\mathsf{cons}\ x\ l)$	$\to$	$\mathsf{dfoldl}\ f\ (f\ y\ x)\ n\ (\mathsf{drop}\ n\ l)$	$[n \equiv n]$

- Troublesome DP problem:

$$(\ \{\ \mathsf{dfoldl}^\sharp \qquad (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{dfoldl}^\sharp \qquad (\mathsf{drop}\ n\ l)\ [n \equiv n]\ \},\ \mathcal{R}_{\mathsf{dfoldl}}\ )$$

- All rules are usable!
- **Reduction pair processor with usable rules wrt argument filtering**:

  temporarily disregard arguments, calculate usable rules, use reduction pair (HORPO, ...)
- $\mathrm{regard}(\mathsf{dfoldl}^\sharp) = \{4\} \quad \Rightarrow$ use first-order RPO!

Reduction pair processor with usable rules wrt an argument filtering

$\mathcal{R}_{\text{dfoldl}}$

drop n l	$\rightarrow$	l	$[n \leq 0]$
drop n nil	$\rightarrow$	nil	$[n \equiv n]$
drop n (cons x l)	$\rightarrow$	drop $(n-1)$ l	$[n > 0]$

- Troublesome DP problem:

$$(\ \{\ \text{dfoldl}^\sharp \quad (\text{cons}\ x\ l) \Rightarrow \text{dfoldl}^\sharp \quad (\text{drop}\ n\ l)\ [n \equiv n]\ \},\ \mathcal{R}_{\text{dfoldl}}\ )$$

- All rules are usable!
- **Reduction pair processor with usable rules wrt argument filtering**:

  temporarily disregard arguments, calculate usable rules, use reduction pair (HORPO, . . .)
- $\text{regard}(\text{dfoldl}^\sharp) = \{4\} \quad \Rightarrow$ use first-order RPO!

Chaining processor

(S)DPs from imperative program [Fuhs, Kop, Nishida, *TOCL '17*]

```
def fact(x):
  z = 1           # fact
  i = 1           # u1
  while i <= x:   # u2
    z = z * i     # u3
    i = i + 1     # u4
                  # u5
```

Chaining processor

(S)DPs from imperative program [Fuhs, Kop, Nishida, *TOCL '17*]

```
def fact(x):
  z = 1            # fact
  i = 1            # u1
  while i <= x:    # u2
    z = z * i      # u3
    i = i + 1      # u4
                   # u5
```

$$
\begin{array}{llll}
\mathsf{fact}^\sharp\ x & \Rightarrow & \mathsf{u_1}^\sharp\ x\ 1 & [x \equiv x] \\
\mathsf{u_1}^\sharp\ x\ z & \Rightarrow & \mathsf{u_2}^\sharp\ x\ z\ 1 & [x \equiv x \wedge z \equiv z] \\
\mathsf{u_2}^\sharp\ x\ z\ i & \Rightarrow & \mathsf{u_3}^\sharp\ x\ z\ i & [i \leq x \wedge z \equiv z] \\
\mathsf{u_3}^\sharp\ x\ z\ i & \Rightarrow & \mathsf{u_4}^\sharp\ x\ (z * i)\ i & [x \equiv x \wedge z \equiv z \wedge i \equiv i] \\
\mathsf{u_4}^\sharp\ x\ z\ i & \Rightarrow & \mathsf{u_2}^\sharp\ x\ z\ (i + 1) & [x \equiv x \wedge z \equiv z \wedge i \equiv i] \\
\mathsf{u_2}^\sharp\ x\ z\ i & \Rightarrow & \mathsf{u_5}^\sharp\ x\ z & [\neg(i \leq x) \wedge z \equiv z]
\end{array}
$$

- Automated translations $\Rightarrow$ DPs with many small steps
- Can be hard to analyse!

Chaining processor

(S)DPs from imperative program [Fuhs, Kop, Nishida, *TOCL '17*]

```
def fact(x):
  z = 1            # fact
  i = 1            # u1
  while i <= x:    # u2
    z = z * i      # u3
    i = i + 1      # u4
                   # u5
```

$\mathsf{fact}^\sharp\ x$	$\Rightarrow$	$\mathsf{u_1}^\sharp\ x\ 1$	$[x \equiv x]$
$\mathsf{u_1}^\sharp\ x\ z$	$\Rightarrow$	$\mathsf{u_2}^\sharp\ x\ z\ 1$	$[x \equiv x \wedge z \equiv z]$
$\mathsf{u_2}^\sharp\ x\ z\ i$	$\Rightarrow$	$\mathsf{u_3}^\sharp\ x\ z\ i$	$[i \leq x \wedge z \equiv z]$
$\mathsf{u_3}^\sharp\ x\ z\ i$	$\Rightarrow$	$\mathsf{u_4}^\sharp\ x\ (z * i)\ i$	$[x \equiv x \wedge z \equiv z \wedge i \equiv i]$
$\mathsf{u_4}^\sharp\ x\ z\ i$	$\Rightarrow$	$\mathsf{u_2}^\sharp\ x\ z\ (i + 1)$	$[x \equiv x \wedge z \equiv z \wedge i \equiv i]$
$\mathsf{u_2}^\sharp\ x\ z\ i$	$\Rightarrow$	$\mathsf{u_5}^\sharp\ x\ z$	$[\neg(i \leq x) \wedge z \equiv z]$

- Automated translations $\Rightarrow$ DPs with many small steps
- Can be hard to analyse!
- **Chaining processor**: remove intermediate symbols $\mathsf{u_1}^\sharp$

Chaining processor

(S)DPs from imperative program [Fuhs, Kop, Nishida, *TOCL '17*]

```
def fact(x):
  z = 1              # fact
  i = 1              # u1
  while i <= x:      # u2
    z = z * i        # u3
    i = i + 1        # u4
                     # u5
```

$$
\begin{array}{llll}
\mathsf{fact}^\sharp\ x & \Rightarrow & & \\
 & & \mathsf{u_2}^\sharp\ x\ 1\ 1 & [x \equiv x] \\
\mathsf{u_2}^\sharp\ x\ z\ i & \Rightarrow & \mathsf{u_3}^\sharp\ x\ z\ i & [i \leq x \wedge z \equiv z] \\
\mathsf{u_3}^\sharp\ x\ z\ i & \Rightarrow & \mathsf{u_4}^\sharp\ x\ (z * i)\ i & [x \equiv x \wedge z \equiv z \wedge i \equiv i] \\
\mathsf{u_4}^\sharp\ x\ z\ i & \Rightarrow & \mathsf{u_2}^\sharp\ x\ z\ (i+1) & [x \equiv x \wedge z \equiv z \wedge i \equiv i] \\
\mathsf{u_2}^\sharp\ x\ z\ i & \Rightarrow & \mathsf{u_5}^\sharp\ x\ z & [\neg(i \leq x) \wedge z \equiv z]
\end{array}
$$

- Automated translations $\Rightarrow$ DPs with many small steps
- Can be hard to analyse!
- **Chaining processor**: remove intermediate symbols $\mathsf{u_1}^\sharp$, $\mathsf{u_3}^\sharp$

Chaining processor

(S)DPs from imperative program [Fuhs, Kop, Nishida, *TOCL '17*]

```
def fact(x):
  z = 1              # fact
  i = 1              # u1
  while i <= x:      # u2
    z = z * i        # u3
    i = i + 1        # u4
                     # u5
```

$$\mathsf{fact}^\sharp\ x \Rightarrow \mathsf{u_2}^\sharp\ x\ 1\ 1 \quad [x \equiv x]$$

$$\mathsf{u_2}^\sharp\ x\ z\ i \Rightarrow \mathsf{u_4}^\sharp\ x\ (z * i)\ i \quad [i \leq x \land x \equiv x \land z \equiv z \land i \equiv i]$$

$$\mathsf{u_4}^\sharp\ x\ z\ i \Rightarrow \mathsf{u_2}^\sharp\ x\ z\ (i + 1) \quad [x \equiv x \land z \equiv z \land i \equiv i]$$

$$\mathsf{u_2}^\sharp\ x\ z\ i \Rightarrow \mathsf{u_5}^\sharp\ x\ z \quad [\neg(i \leq x) \land z \equiv z]$$

- Automated translations $\Rightarrow$ DPs with many small steps
- Can be hard to analyse!
- **Chaining processor**: remove intermediate symbols $\mathsf{u_1}^\sharp$, $\mathsf{u_3}^\sharp$, $\mathsf{u_4}^\sharp$

Chaining processor

(S)DPs from imperative program [Fuhs, Kop, Nishida, *TOCL '17*]

```
def fact(x):
  z = 1            # fact
  i = 1            # u1
  while i <= x:    # u2
    z = z * i      # u3
    i = i + 1      # u4
                   # u5
```

$\mathsf{fact}^\sharp\ x \Rightarrow \mathsf{u_2}^\sharp\ x\ 1\ 1 \quad [x \equiv x]$

$\mathsf{u_2}^\sharp\ x\ z\ i \Rightarrow \mathsf{u_2}^\sharp\ x\ z\ (i+1) \quad [i \leq x \wedge x \equiv x \wedge z \equiv z \wedge i \equiv i]$

$\mathsf{u_2}^\sharp\ x\ z\ i \Rightarrow \mathsf{u_5}^\sharp\ x\ z \quad [\neg(i \leq x) \wedge z \equiv z]$

- Automated translations $\Rightarrow$ DPs with many small steps
- Can be hard to analyse!
- **Chaining processor**: remove intermediate symbols $\mathsf{u_1}^\sharp$, $\mathsf{u_3}^\sharp$, $\mathsf{u_4}^\sharp$

Chaining processor

(S)DPs from imperative program [Fuhs, Kop, Nishida, *TOCL '17*]

```
def fact(x):
  z = 1            # fact
  i = 1            # u1
  while i <= x:    # u2
    z = z * i      # u3
    i = i + 1      # u4
                   # u5
```

$$\mathsf{fact}^\sharp\ x \Rightarrow \mathsf{u_2}^\sharp\ x\ 1\ 1 \quad [x \equiv x]$$
$$\mathsf{u_2}^\sharp\ x\ z\ i \Rightarrow \mathsf{u_2}^\sharp\ x\ z\ (i+1) \quad [i \leq x \land z \equiv z]$$
$$\mathsf{u_2}^\sharp\ x\ z\ i \Rightarrow \mathsf{u_5}^\sharp\ x\ z \quad [\neg(i \leq x) \land z \equiv z]$$

- Automated translations $\Rightarrow$ DPs with many small steps
- Can be hard to analyse!
- **Chaining processor**: remove intermediate symbols $\mathsf{u_1}^\sharp$, $\mathsf{u_3}^\sharp$, $\mathsf{u_4}^\sharp$

Chaining processor

(S)DPs from imperative program [Fuhs, Kop, Nishida, *TOCL '17*]

```
def fact(x):
  z = 1           # fact
  i = 1           # u1
  while i <= x:   # u2
    z = z * i     # u3
    i = i + 1     # u4
                  # u5
```

$$\mathsf{fact}^\sharp\ x \Rightarrow \mathsf{u_2}^\sharp\ x\ 1\ 1 \quad [x \equiv x]$$
$$\mathsf{u_2}^\sharp\ x\ z\ i \Rightarrow \mathsf{u_2}^\sharp\ x\ z\ (i+1) \quad [i \leq x \wedge z \equiv z]$$
$$\mathsf{u_2}^\sharp\ x\ z\ i \Rightarrow \mathsf{u_5}^\sharp\ x\ z \quad [\neg(i \leq x) \wedge z \equiv z]$$

- Automated translations $\Rightarrow$ DPs with many small steps
- Can be hard to analyse!
- **Chaining processor**: remove intermediate symbols $\mathsf{u_1}^\sharp$, $\mathsf{u_3}^\sharp$, $\mathsf{u_4}^\sharp$
- Integer mapping processor + graph processor prove termination

Compositional analysis from Universal Computability

- Goal: compositional **open-world** program analysis

Compositional analysis from Universal Computability

- Goal: compositional **open-world** program analysis
- For termination analysis: Universal Computability [Guo, Hagens, Kop, Vale, *MFCS '24*]

Compositional analysis from Universal Computability

- Goal: compositional **open-world** program analysis
- For termination analysis: Universal Computability [Guo, Hagens, Kop, Vale, *MFCS '24*]
- Analyse LCSTRS for use in context of larger program

Compositional analysis from Universal Computability

- Goal: compositional **open-world** program analysis
- For termination analysis: Universal Computability [Guo, Hagens, Kop, Vale, *MFCS '24*]
- Analyse LCSTRS for use in context of larger program
- Usable rules + reduction pair processor available for innermost (and cbv) rewriting!

Implementation

- Implementation in open-source tool Cora: https://github.com/hezzel/cora/
- HORPO as reduction pair
- Z3 as SMT solver

Experiments (1/3)

Experiments using 60 seconds timeout

Experiments (1/3)

Experiments using 60 seconds timeout

275 inputs: integer TRSs + λ-free HO-TRSs from TPDB + own benchmarks

Experiments (1/3)

Experiments using 60 seconds timeout

275 inputs: integer TRSs + λ-free HO-TRSs from TPDB + own benchmarks

Cora (innermost/cbv) v Cora (full) [Guo, Hagens, Kop, Vale, *MFCS '24*]

	Termination			**Universal Computability**		
	Full	Innermost	Call-by-value	Full	Innermost	Call-by-value
Total yes	171	179	182	155	179	182

Experiments (2/3)

117 integer TRSs: Cora v AProVE [Giesl et al, *JAR '17*] [Fuhs et al, *RTA '09*]

	Cora innermost	Cora call-by-value	AProVE innermost
Total yes	72	73	102

Experiments (2/3)

117 integer TRSs: Cora v AProVE [Giesl et al, *JAR '17*] [Fuhs et al, *RTA '09*]

	Cora innermost	Cora call-by-value	AProVE innermost
Total yes	72	73	102

- AProVE has strong reduction pair processor with polynomial interpretations and usable rules
- AProVE can handle rules $\mathsf{f}(x) \to \mathsf{g}(x > 0, x),\ \mathsf{g}(\mathfrak{t}, x) \to r_1,\ \mathsf{g}(\mathfrak{f}, x) \to r_2$ well

Experiments (3/3)

140 λ-free HO-TRSs: Cora v WANDA [Kop, *FSCD '20*]

	Cora innermost / call-by-value	WANDA full termination
Total yes	79	105

Experiments (3/3)

140 λ-free HO-TRSs: Cora v WANDA [Kop, *FSCD '20*]

	Cora innermost / call-by-value	WANDA full termination
Total yes	79	105

- WANDA: Polynomial interpretations, dynamic DPs, delegation to first-order termination tool, ...

Conclusion

- Transformation for analysis of LCSTRSs with call-by-value via innermost strategy
- Three new processors: usable rules, reduction pair with temporary argument filtering, chaining
- Improved open-world termination analysis

Conclusion

- Transformation for analysis of LCSTRSs with call-by-value via innermost strategy
- Three new processors: usable rules, reduction pair with temporary argument filtering, chaining
- Improved open-world termination analysis
- Implementation: https://github.com/hezzel/cora/
- Evaluation page: https://www.cs.ru.nl/~cynthiakop/experiments/fscd25/

Conclusion

- Transformation for analysis of LCSTRSs with call-by-value via innermost strategy
- Three new processors: usable rules, reduction pair with temporary argument filtering, chaining
- Improved open-world termination analysis
- Implementation: https://github.com/hezzel/cora/
- Evaluation page: https://www.cs.ru.nl/~cynthiakop/experiments/fscd25/
- FSCD 2025 paper:

An Innermost DP Framework for Constrained Higher-Order Rewriting

Carsten Fuhs
Birkbeck, University of London, UK

Liye Guo
Radboud University, Nijmegen, The Netherlands

Cynthia Kop
Radboud University, Nijmegen, The Netherlands

Conclusion

- Transformation for analysis of LCSTRSs with call-by-value via innermost strategy
- Three new processors: usable rules, reduction pair with temporary argument filtering, chaining
- Improved open-world termination analysis
- Implementation: https://github.com/hezzel/cora/
- Evaluation page: https://www.cs.ru.nl/~cynthiakop/experiments/fscd25/
- FSCD 2025 paper:

An Innermost DP Framework for Constrained Higher-Order Rewriting

Carsten Fuhs
Birkbeck, University of London, UK

Liye Guo
Radboud University, Nijmegen, The Netherlands

Cynthia Kop
Radboud University, Nijmegen, The Netherlands

Thanks a lot for your attention!