Deciding Pattern Completeness in Non-Deterministic Polynomial Time

René Thiemann and Akihisa Yamada

Supported by the Austrian Science Fund (FWF) project I 5943

IWC 2025, September 2–3, 2025

Motivation: Certify Ground Confluence of TRSs

- ground confluence: given term rewrite system (TRS) $\mathcal{R}$ over $\mathcal{T}(\mathcal{F},\mathcal{V})$, determine whether $\mathcal{R}$ is confluent on ground terms
- example: signature $\mathcal{F} = \{\mathsf{isBool}, \mathsf{true}, \mathsf{false}, \mathsf{test}\}$, rules $\mathcal{R}$:

$$\begin{aligned}
\mathsf{isBool}(\mathsf{true}) &\to \mathsf{true}\\
\mathsf{isBool}(\mathsf{false}) &\to \mathsf{true}\\
\mathsf{test}(x) &\to \mathsf{isBool}(x)\\
\mathsf{test}(x) &\to \mathsf{true}
\end{aligned}$$

- state of the art in confluence analysis (CoCo 2023 full run)
  - best tool in CR: 272 yes + 205 no, 188 yes + 205 no (certified)
  - best tool in GCR: 350 yes + 126 no, 0 yes + 0 no (certified)
- aim: formalize ground confluence techniques in Isabelle/HOL
- method of AGCP [Aoto, Toyama] to ensure GCR is based on bounded ground convertibility, rewriting induction and sufficient completeness

Motivation: Certify Ground Confluence of TRSs

- fix TRS $\mathcal{R}$, fix reduction order $\succ$ such that $\mathcal{R} \subseteq \succ$
- bounded ground convertible
  - s and t are bounded ground convertible if for all $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F})$

$$s\sigma \leftrightarrow_{\mathcal{R}}^{*} t\sigma \qquad \text{such that } s\sigma \succeq u \text{ or } t\sigma \succeq u \text{ for all intermediate terms } u$$

- rewriting induction
  - splits signature into $\mathcal{F} = \mathcal{C} \uplus \mathcal{D}$
  - inference rules to ensure for given s and t that for all $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{C})$

$$s\sigma \leftrightarrow_{\mathcal{R}}^{*} t\sigma \qquad \text{such that } s\sigma \succeq u \text{ or } t\sigma \succeq u \text{ for all intermediate terms } u$$

- require sufficient completeness for switching from $\mathcal{F}$ to $\mathcal{C}$ in substitutions

$$\forall t \in \mathcal{T}(\mathcal{F}).\ \exists s \in \mathcal{T}(\mathcal{C}).\ t \to_{\mathcal{R}}^{*} s$$

- criterion for sufficient completeness: termination and pattern completeness

Pattern Completeness

- property to ensure that evaluation by pattern matching cannot get stuck
- formal definition of pattern completeness

$$\forall t \in \mathcal{B}(\mathcal{C},\mathcal{D}).\ \exists \ell \to r \in \mathcal{R}, \mu.\ t = \ell\mu$$

  with basic ground terms $\mathcal{B}(\mathcal{C},\mathcal{D}) = \{f(t_1,\dots,t_n) \mid f \in \mathcal{D}, t_1,\dots,t_n \in \mathcal{T}(\mathcal{C})\}$
- pattern completeness is decidable for left-linear TRSs
  - via tree-automata: "basic ground terms" $\subseteq$ "terms matched by some lhs"
  - via complement algorithm [Lazrek, Lescanne, Thiel]
- decision procedure for arbitrary TRSs presented at FSCD 2024
- pattern completeness of Haskell programs is checked by GHC compiler
- decision problem for pattern completeness is co-NP complete

Sorts Matter

- consider TRS with signature $\mathcal{D} = \{\mathsf{even}\}$ and $\mathcal{C} = \{\mathsf{true}, \mathsf{false}, \mathsf{0}, \mathsf{s}\}$

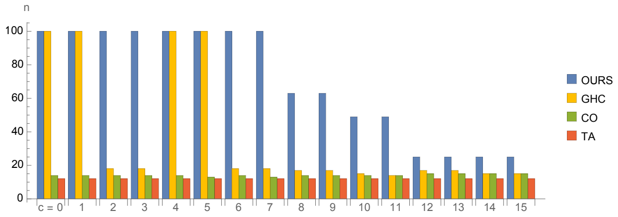
$$\begin{aligned}\mathsf{even}(\mathsf{0}) &\to \mathsf{true}\\ \mathsf{even}(\mathsf{s}(\mathsf{0})) &\to \mathsf{false}\\ \mathsf{even}(\mathsf{s}(\mathsf{s}(x))) &\to \mathsf{even}(x)\end{aligned}$$

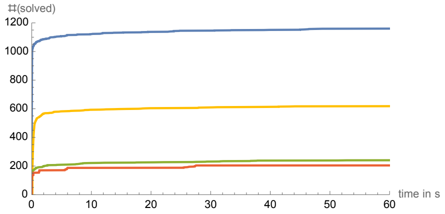
- not pattern complete: $\mathsf{even}(\mathsf{true})$
- TRS is pattern complete in sorted setting

$$\begin{aligned}\mathsf{even} &: \mathsf{Num} \to \mathsf{Bool}\\ \mathsf{true} &: \mathsf{Bool}\\ \mathsf{false} &: \mathsf{Bool}\\ \mathsf{0} &: \mathsf{Num}\\ \mathsf{s} &: \mathsf{Num} \to \mathsf{Num}\end{aligned}$$

FSCD Decision Procedure for Pattern Completeness

- main features
  - faster than existing algorithms
  - easy to implement, inspired by matching algorithm
  - arbitrary patterns, no restriction to left-linearity
  - verified in Isabelle/HOL
- performance on synthetic benchmark (size n, configuration $c \in \{0, \dots, 15\}$)





This Talk

- problem: FSCD algorithm requires exponential space on non-linear inputs
- aim: modify algorithm, so that membership in co-NP is guaranteed
- method
  1. change representation to non-deterministic one
  2. simplify pattern problems in several phases, finally translate to SMT problem
     - in each phase, at most polynomially many steps are performed
     - also the final encoding to SMT will have polynomial size
- results on 300 non-linear complete and 300 non-linear incomplete problems

	YES	NO	TIMEOUT
FSCD algorithm	120	123	357
NEW algorithm	211	300	89

The Algorithm – Problem Types

- (modified) matching problem
  - matching problem is finite set $mp = \{t_1 \sim \ell_1, \dots, t_n \sim \ell_n\}$ where t_i, ℓ_i are terms
  - ℓ_i is subterm of left-hand side of TRS: is t_i matched by ℓ_i
  - modification: vars in t_i represent constr. ground terms: is $t_i\sigma$ matched by ℓ_i
  - notion: mp is complete w.r.t. σ if there is μ such that $t_i\sigma = \ell_i\mu$ for all i
  - $\bot_{mp}$ is special matching problem that is never complete
- pattern problem = disjunctive combination of matching problems
  - pattern problem is finite set $pp = \{mp_1, \dots, mp_k\}$ of matching problems
  - pp is complete if for each constructor ground substitution σ, at least one of the matching problems mp_i is complete w.r.t. σ
- example for even-TRS
  - basic-terms represented by $\mathsf{even}(y)$
  - pattern completeness of TRS expressed as completeness of $pp_{\mathsf{even}} := \{\{\mathsf{even}(y) \sim \mathsf{even}(0)\}, \{\mathsf{even}(y) \sim \mathsf{even}(\mathsf{s}(0))\}, \{\mathsf{even}(y) \sim \mathsf{even}(\mathsf{s}(\mathsf{s}(x)))\}\}$

Phase 1: Simplification Rules for Matching Problems

- we transform matching problems via simplification relation $\to$
- $\to$ does not change completeness

$$\{f(t_1,\dots,t_n) \sim f(\ell_1,\dots,\ell_n)\} \uplus mp \to \{t_1 \sim \ell_1,\dots,t_n \sim \ell_n\} \cup mp \quad \text{(decompose)}$$

$$\{f(\dots) \sim g(\dots)\} \uplus mp \to \bot_{mp} \quad \text{if } f \neq g \quad \text{(clash)}$$

$$\{t \sim x\} \uplus mp \to mp \quad \text{if “}x \notin mp\text{”} \quad \text{(match)}$$

$$\{t \sim x, t' \sim x\} \uplus mp \to \bot_{mp} \quad \text{if } t \text{ and } t' \text{ do not unify} \quad \text{(clash')}$$

$$\{y \sim x, t \sim x\} \uplus mp \to \bot_{mp} \quad \text{if } t \notin \mathcal{T}(\mathcal{C},\mathcal{V}) \quad \text{(no-constructor)}$$

$$\{f(t_{i1},\dots,t_{in}) \sim x \mid i \in I\} \uplus mp \to \{t_{ij} \sim z_j \mid i \in I, 1 \leqslant j \leqslant n\} \cup mp$$
$$\text{if “}x \notin mp\text{” and } z_1,\dots,z_n \text{ are fresh variables} \quad \text{(decompose')}$$

- new rules in comparison to FSCD algorithm

Phase 1: Simplification Rules for Pattern Problems

- $\Rightarrow$ transforms a pattern problems into a set of pattern problems
  - equivalence: if $pp \Rightarrow Q$ then pp is complete iff all $qq \in Q$ are complete
- difference to FSCD: $\Rightarrow_{nd}$ makes non-deterministic choices
  - $pp \Rightarrow_{nd} qq$ iff $pp \Rightarrow Q$ and $qq \in Q$

$$\{mp\} \uplus pp \Rightarrow \{\{mp'\} \cup pp\} \qquad \text{if } mp \rightarrow mp' \qquad \text{(simp-mp)}$$

$$\{mp\} \uplus pp \Rightarrow \{pp\} \qquad \text{if } mp \rightarrow \bot_{mp} \qquad \text{(remove-mp)}$$

$$\{\emptyset\} \uplus pp \Rightarrow \emptyset \qquad \text{(success)}$$

$$pp \Rightarrow \mathrm{Inst}(pp, x) \qquad \text{if } mp \in pp \text{ and } x \sim f(\ldots) \in mp \qquad \text{(instantiate)}$$

$$pp \uplus pp' \Rightarrow \{pp'\} \ \text{ if } pp \neq \emptyset\text{, all variables in } pp' \text{ are of finite sort, and}$$
$$\forall mp \in pp.\ \exists x, y, t, \iota.\ \{y \sim x, t \sim x\} \subseteq mp \wedge y : \iota \in \mathcal{V} \wedge y \neq t \wedge |\iota| = \infty$$
$$\text{(inf-diff)}$$

- generalized rule in comparison to FSCD algorithm

$$\begin{tikzcd}
 & pp = \{\{\mathsf{even}(y) \sim \mathsf{even}(0)\}, \{\mathsf{even}(y) \sim \mathsf{even}(\mathsf{s}(0))\}, \{\mathsf{even}(y) \sim \mathsf{even}(\mathsf{s}(\mathsf{s}(x)))\}\} \arrow[d, no head] & \\
 & \{\{y \sim 0\}, \{y \sim \mathsf{s}(0)\}, \{y \sim \mathsf{s}(\mathsf{s}(x))\}\} \arrow[dl, no head, "y/0"'] \arrow[dr, no head, "y/\mathsf{s}(z)"] & \\
\{\{0 \sim 0\}, \{0 \sim \mathsf{s}(0)\}, \{0 \sim \mathsf{s}(\mathsf{s}(x))\}\} \arrow[d, no head] & & \{\{\mathsf{s}(z) \sim 0\}, \{\mathsf{s}(z) \sim \mathsf{s}(0)\}, \{\mathsf{s}(z) \sim \mathsf{s}(\mathsf{s}(x))\}\} \arrow[d, no head] \\
\{\emptyset, \bot_{mp}, \bot_{mp}\} \arrow[d, no head] & & \{\bot_{mp}, \{z \sim 0\}, \{z \sim \mathsf{s}(x)\}\} \arrow[d, no head] \\
\{\emptyset\} & & \{\{z \sim 0\}, \{z \sim \mathsf{s}(x)\}\} \arrow[dl, no head, "z/0"'] \arrow[d, no head, "z/\mathsf{s}(y)"] \\
 & \{\{0 \sim 0\}, \{0 \sim \mathsf{s}(x)\}\} \arrow[d, no head] & \{\{\mathsf{s}(y) \sim 0\}, \{\mathsf{s}(y) \sim \mathsf{s}(x)\}\} \arrow[d, no head] \\
 & \{\emptyset, \bot_{mp}\} \arrow[d, no head] & \{\bot_{mp}, \emptyset\} \arrow[d, no head] \\
 & \{\emptyset\} & \{\emptyset\}
\end{tikzcd}$$

Phase 1: Results

Theorem

- *the number of $\Rightarrow_{nd}$-steps is polynomially bounded* *(new)*
- *pp is incomplete iff there is some qq such that $pp \Rightarrow^!_{nd} qq$ and qq is incomplete.*
- *if pp is linear then pp is incomplete iff $pp \Rightarrow^!_{nd} \emptyset$*
- *if $pp \Rightarrow^!_{nd} qq$ then qq is in finite constructor form* *(new)*

- pp has finite constructor form iff each $t \sim \ell \in mp \in pp$ satisfies
  - t and ℓ have a finite sort, and
  - $t \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ and $\ell \in \mathcal{V}$
- Phase 2 eliminates constructors in t-terms to obtain finite variable form
- Phase 3 encodes completeness of finite variable form problems into SMT

Main Difference to FSCD Algorithm

- consider some non-linear problem $\{\{s \sim x, t \sim x\}\}$
- problem is complete iff $s\sigma = t\sigma$ for all constructor ground substitutions σ
- in general, $s\sigma$ and $t\sigma$ might become large, e.g., a full binary tree
- the FSCD algorithm will generate terms $s\sigma$ and $t\sigma$ starting from the root, until a clash is found or until $s\sigma = t\sigma$ (term size might be exponential)
- the new algorithm will non-deterministically search for a shared path of $s\sigma$ and $t\sigma$ until a clash is found (path length is polynomially bounded)
- moreover, the new algorithm will stop to explore term structure in case of variables, and instead switch to SMT
- in paper: details of Phase 2 (non-deterministic search with optimizations) and details of Phase 3 (SMT encoding)

Summary

- improved decision procedure for pattern completeness
    - theoretically optimal: co-NP
    - faster in experiments than FSCD algorithm
    - formally verified in Isabelle/HOL (including the required SMT solver)
- future work
    - formalize rewriting induction for certifying ground confluence proofs of AGCP

Thank you! Questions?

Examples in Performance-Tests: $f(x, x)$

```
(format MSTRS)
(sort s0) ... (sort s4)

; constructors
(fun d s0)
(fun c0 s0) ; include for incomplete, drop for complete
(fun c1 (-> s0 s0 s1))
...
(fun c4 (-> s3 s3 s4))

; defined
(fun f (-> s4 s4 s0))

(rule (f x x) d)
```

Examples in Performance-Tests: $f(\ldots,\_,x,c_i(x,x),\_,\ldots)$

```
; sorts as in previous example

(rule (f x0 (c1 x0 x0) x2 x3 x4) d)
(rule (f x0 x1 (c2 x1 x1) x3 x4) d)
(rule (f x0 x1 x2 (c3 x2 x2) x4) d)
(rule (f x0 x1 x2 x3 (c4 x3 x3)) d)
```

Examples in Performance-Tests: Pigeon-Hole Principle

```
(sort s)
(fun c0 s) ... (fun c3 s) ; add another constant for incomplete
(fun f (-> s s s s s s))
(rule (f y y x2 x3 x4) c0)
(rule (f y x1 y x3 x4) c0)
(rule (f y x1 x2 y x4) c0)
(rule (f y x1 x2 x3 y) c0)
(rule (f x0 y y x3 x4) c0)
(rule (f x0 y x2 y x4) c0)
(rule (f x0 y x2 x3 y) c0)
(rule (f x0 x1 y y x4) c0)
(rule (f x0 x1 y x3 y) c0)
(rule (f x0 x1 x2 y y) c0)
```