Automated Termination and Confluence Analysis of Rewrite System

Julian Nagele

University of Innsbruck, Austria

CHUM 4 @ IBM Watson September 8, 2015

Outline

- Motivation
- Termination
- Confluence
- Certification
- Conclusion

Formal Software Verification

- testing is not sufficient
- formal verification: prove correctness
- model of computation?

Programming Languages

Turing machines, register machines, two-counter automata, pointer machines, queue automata, μ-recursive functions, λ-calculus, Post canonical systems, combinatory logic, type-0 grammars, Semi-Thue systems, term rewriting, WHILE programs, Haskell, OCaml, Java, C, Prolog, ...

> *Well, when all is said and done, the only thing computers can do for us is to manipulate symbols and produce results of such manipulations. – Edsger W. Dijkstra, 1988*

Term Rewriting

Example

$$0 + y \to y \qquad 0 \times y \to 0$$
$$\mathsf{s}(x) + y \to \mathsf{s}(x + y) \qquad \mathsf{s}(x) \times y \to y + (x \times y)$$

$$\begin{aligned}\mathsf{s}(0) \times \mathsf{s}(\mathsf{s}(0)) &\to \mathsf{s}(\mathsf{s}(0)) + (0 \times \mathsf{s}(\mathsf{s}(0)))\\ &\to \mathsf{s}(\mathsf{s}(0)) + 0 \to \mathsf{s}(\mathsf{s}(0) + 0)\\ &\to \mathsf{s}(\mathsf{s}(0 + 0)) \to \mathsf{s}(\mathsf{s}(0))\end{aligned}$$

Properties

- Termination: Do all computations produce a result?
- Complexity: What is the cost of producing a result?
- Confluence: Are results unique?

- Automation: Automatic and reliable analysis

Basic Definitions

Rewrite Systems

- terms built from function symbols and variables $t ::= x \mid f(t_1, \ldots, t_n)$
- rewrite rule $\ell \to r$ is pair of terms with $\ell \notin \mathcal{V}$ and $\mathcal{V}\mathrm{ar}(r) \subseteq \mathcal{V}\mathrm{ar}(\ell)$
- rewrite system $\mathcal{R}$ is set of rewrite rules
- context C is term with one special symbol $\Box$
- $C[t]$ is replacement of $\Box$ by t in C
- substitution σ is mapping from variables to terms
- $t\sigma$ is application to t by homomorphic extension

Rewrite Relation

- $s \to_{\mathcal{R}} t$ if $s = C[\ell\sigma]$ and $t = C[r\sigma]$ for $\ell \to r \in \mathcal{R}$
- $\to_{\mathcal{R}}^*$ is reflexive transitive closure of $\to_{\mathcal{R}}$

Higher-Order Rewriting

Example

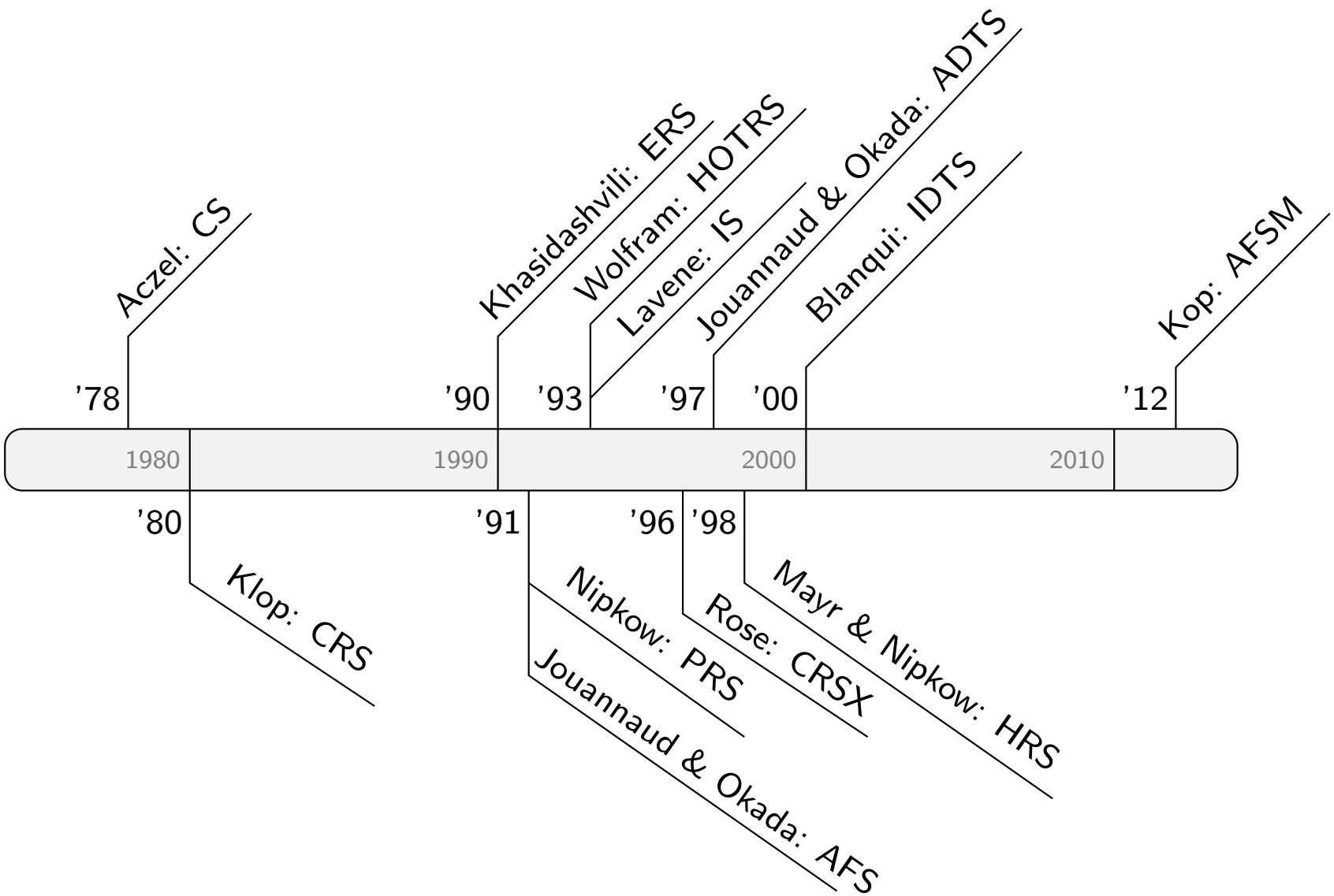
$$@([], ys) \to ys \qquad @(x : xs, ys) \to x : @(xs, ys)$$
$$\mathsf{rev}([]) \to [] \qquad \mathsf{rev}(x : xs) \to @(\mathsf{rev}(xs), x : [])$$

$$\mathsf{foldl}(\lambda x\, y.\, f(x, y), z, []) \to z$$
$$\mathsf{foldl}(\lambda x\, y.\, f(x, y), z, h : t) \to \mathsf{foldl}(\lambda x\, y.\, f(x, y), f(z, h), t)$$

$$\mathsf{foldl}(\lambda u\, v.\, u + v, 0, 1 : 2 : []) \to^* 3$$

- functional variables
- function abstraction as variable binder
- HOR = λ-calculus + term rewriting

History


Aczel: CS
Khasidashvili: ERS
Wolfram: HOTRS
Lavene: IS
Jouannaud & Okada: ADTS
Blanqui: IDTS
Kop: AFSM
'78
'90
'93
'97
'00
'12
1980
1990
2000
2010
'80
'91
'96
'98
Klop: CRS
Nipkow: PRS
Jouannaud & Okada: AFS
Rose: CRSX
Mayr & Nipkow: HRS

Outline

- Motivation
- Termination
- Confluence
- Certification
- Conclusion

Real World Programming Languages?

C

```
int main(){
    int x = 0;
    if (x == x++)
      while (1) ;
}
```

- terminates compiled with gcc version 5.2.0
- loops compiled with clang version 3.6.2

Rewriting

Definition

rewrite system $\mathcal{R}$ is terminating if there are no infinite $\to_{\mathcal{R}}$ sequences

Observation

$\mathcal{R}$ is terminating if and only if $\exists$ well-founded order on terms such that $s > t$ whenever $s \to_{\mathcal{R}} t$

Wanted

ways to find well-founded order $>$ such that $\ell > r$ for all $\ell \to r \in \mathcal{R}$ implies $s > t$ whenever $s \to_{\mathcal{R}} t$

Example

- interpretation into well-founded monotone algebra
- syntactic orders (LPO, KBO, ...)

Interpretations

Example

- TRS

$$0 + y \to y \qquad 0 \times y \to 0$$
$$\mathsf{s}(x) + y \to \mathsf{s}(x + y) \qquad \mathsf{s}(x) \times y \to y + (x \times y)$$

- polynomial interpretations in $\mathbb{N}$

$$0_{\mathcal{A}} = 1 \qquad +_{\mathcal{A}}(x, y) = 2x + y$$
$$\mathsf{s}_{\mathcal{A}}(x) = x + 1 \qquad \times_{\mathcal{A}}(x, y) = 2xy + x + y + 1$$

- $$\begin{array}{ccccccccc} \mathsf{s}(0) \times \mathsf{s}(\mathsf{s}(0)) & \to & \mathsf{s}(\mathsf{s}(0)) + (0 \times \mathsf{s}(\mathsf{s}(0))) & \to & \mathsf{s}(\mathsf{s}(0)) + 0 & \to & \mathsf{s}(\mathsf{s}(0) + 0) \\ 18 & > & 17 & > & 7 & > & 6 \\ & \to & \mathsf{s}(\mathsf{s}(0 + 0)) & \to & \mathsf{s}(\mathsf{s}(0)) & & \\ & > & 5 & > & 3 & & \end{array}$$

- constraints $\forall\, x\, y \in \mathbb{N}$

$$2 + y > y \qquad 3y + 2 > 1$$
$$2x + y + 2 > 2x + y + 1 \qquad 2xy + x + 3y + 2 > 2xy + x + 3y + 1$$

(Higher-Order) Recursive Path Ordering

- order on function symbols extended to terms by comparing root symbols and recursively comparing arguments

Example

$$\mathsf{map}(\lambda x.\, f(x), []) \to []$$
$$\mathsf{map}(\lambda x.\, f(x), h : t) \to f(h) : \mathsf{map}(\lambda x.\, f(x), t)$$

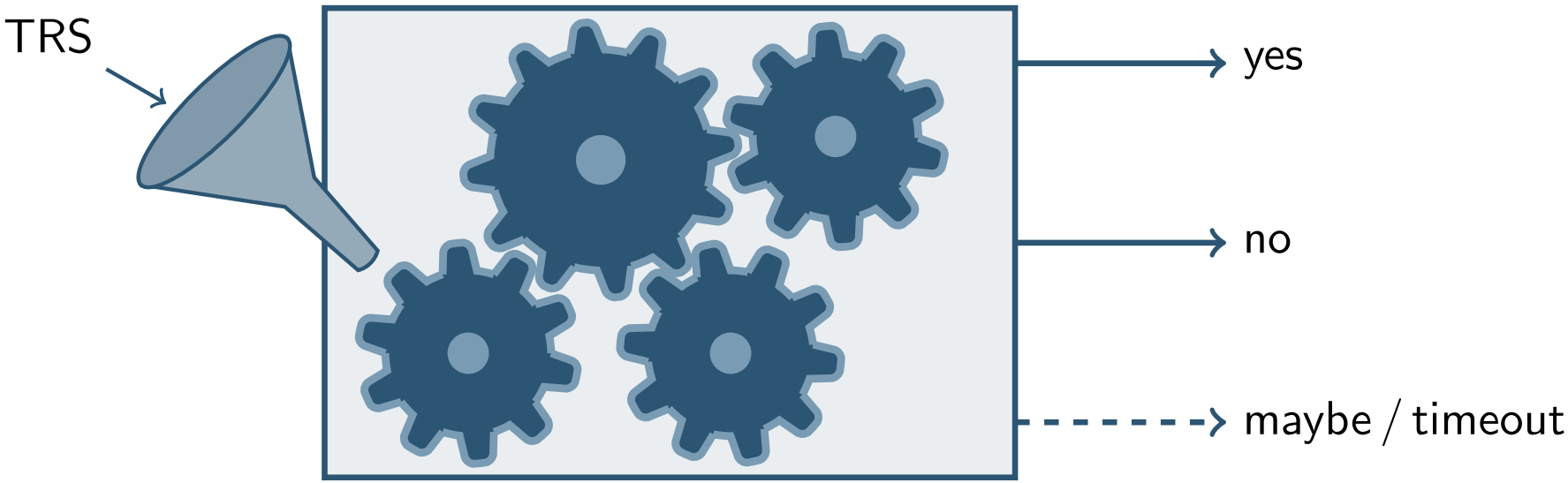
- $\mathsf{map}(\lambda x.\, f(x), []) \succ []$
- $\mathsf{map}(\lambda x.\, f(x), h : t) \succ f(h) : \mathsf{map}(\lambda x.\, f(x), t)$ by setting $\mathsf{map} \succ :$
- $\mathsf{map}(\lambda x.\, f(x), h : t) \succ f(h)$
- $\mathsf{map}(\lambda x.\, f(x), h : t) \succ \mathsf{map}(\lambda x.\, f(x), t)$ by comparing arguments
- $h : t \succ t$

Automation

Termination Methods

Knuth-Bendix order, polynomial interpretations, multiset order, simple path order, lexicographic path order, semantic path order, recursive decomposition order, multiset path order, recursive path order, transformation order, elementary interpretations, type introduction, well-founded monotone algebras, general path order, semantic labeling, dummy elimination, dependency pairs, freezing, top-down labeling, monotonic semantic path order, context-dependent interpretations, match-bounds, size-change principle, matrix interpretations, predictive labeling, uncurrying, bounded increase, quasi-periodic interpretations, arctic interpretations, increasing interpretations, root-labeling, ordinal interpretations, weighted path order, ...

Automation



Termination Tools

AProVE, Cariboo, CiME, Ctrl, HOT, Jambox, Matchbox, MuTerm, NaTT, Termptation, THOR, Torpa, T_TT_2, VMTL, WANDA, ...

termCOMP

- annually since 2004
- Java since 2009, C since 2014

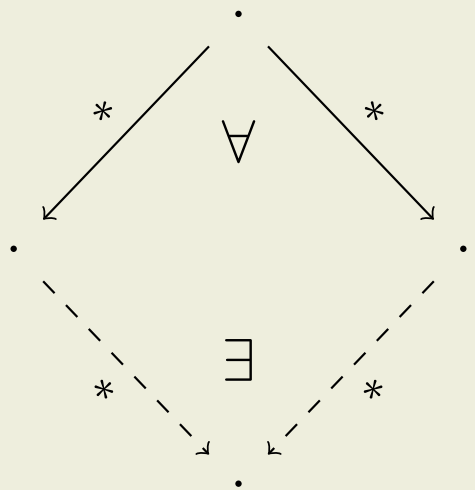
Outline

- Motivation
- Termination
- Confluence
- Certification
- Conclusion

Confluence

Definition

TRS is confluent if



- implies unique results
- corresponds to well-definedess of functions
- parallel and distributed systems
- consistency of logics/λ-calculi

Examples

Example

$$0 + y \to y \qquad x - 0 \to x$$
$$x + 0 \to x \qquad x - \mathsf{s}(y) \to \mathsf{p}(x - y)$$
$$\mathsf{s}(x) + y \to \mathsf{s}(x + y) \qquad \mathsf{s}(\mathsf{p}(x)) \to x$$
$$\mathsf{p}(\mathsf{s}(x)) \to x$$

$$\mathsf{p}(x - \mathsf{p}(y)) \leftarrow x - \mathsf{s}(\mathsf{p}(y)) \to x - y$$

Example

$$(\lambda x.\, M(x))\, N \to_\beta M(N) \qquad \lambda x.\, M\, x \to_\eta M$$

$$\lambda y.\, M(y) \;{}_\eta\!\leftarrow \lambda x.\, (\lambda y.\, M(y))\, x \to_\beta \lambda x.\, M(x)$$

Overlaps

Definition

- situation where two co-initial steps affect the same function symbol
- can be found by unification of left-hand sides
- for finite TRS only finitely many

Theorem (Knuth and Bendix, Huet)

If all overlaps of $\mathcal{R}$ can be joined then $\mathcal{R}$ is locally confluent.

Lemma (Newman)

If $\mathcal{R}$ is locally confluent and terminating then it is confluent.

Examples

Example

$$\mathsf{map}(\lambda x.\, f(x), []) \to []$$
$$\mathsf{map}(\lambda x.\, f(x), h : t) \to f(h) : \mathsf{map}(\lambda x.\, f(x), t)$$

is terminating and does not have overlaps: $[] \not\approx h : t \Longrightarrow$ confluent

Example (Huet)

$$f(x, x) \to a \qquad f(x, g(x)) \to b \qquad c \to g(c)$$

does not have overlaps, but is not confluent:

$$a \leftarrow f(c, c) \to f(c, g(c)) \to b$$

Theorem (Rosen)

Orthogonal rewrite systems are confluent.

Automation

Confluence Criteria

Knuth and Bendix, orthogonality, strongly/parallel/development closed critical pairs, decreasing diagrams (rule labeling), parallel and simultaneous critical pairs, divide and conquer techniques (commutation, layer preservation, order-sorted decomposition), decision procedures, depth/weight preservation, reduction-preserving completion, Church-Rosser modulo, relative termination and extended critical pairs, non-confluence techniques (tcap, tree automata, interpretation), ...

Confluence Tools

ACP, ACPH, AGCP, CO3, CoLL, ConCon, CoScart, CSI, CSIˆho, NoCo, Saigawa, ...

CoCo

- annually since 2012
- higher-order rewriting since 2015

Outline

- Motivation
- Termination
- Confluence
- Certification
- Conclusion

Reliable Analysis

- termination/confluence tools have become extremely complex
- we do formal verification
- why should we trust the output these tools?

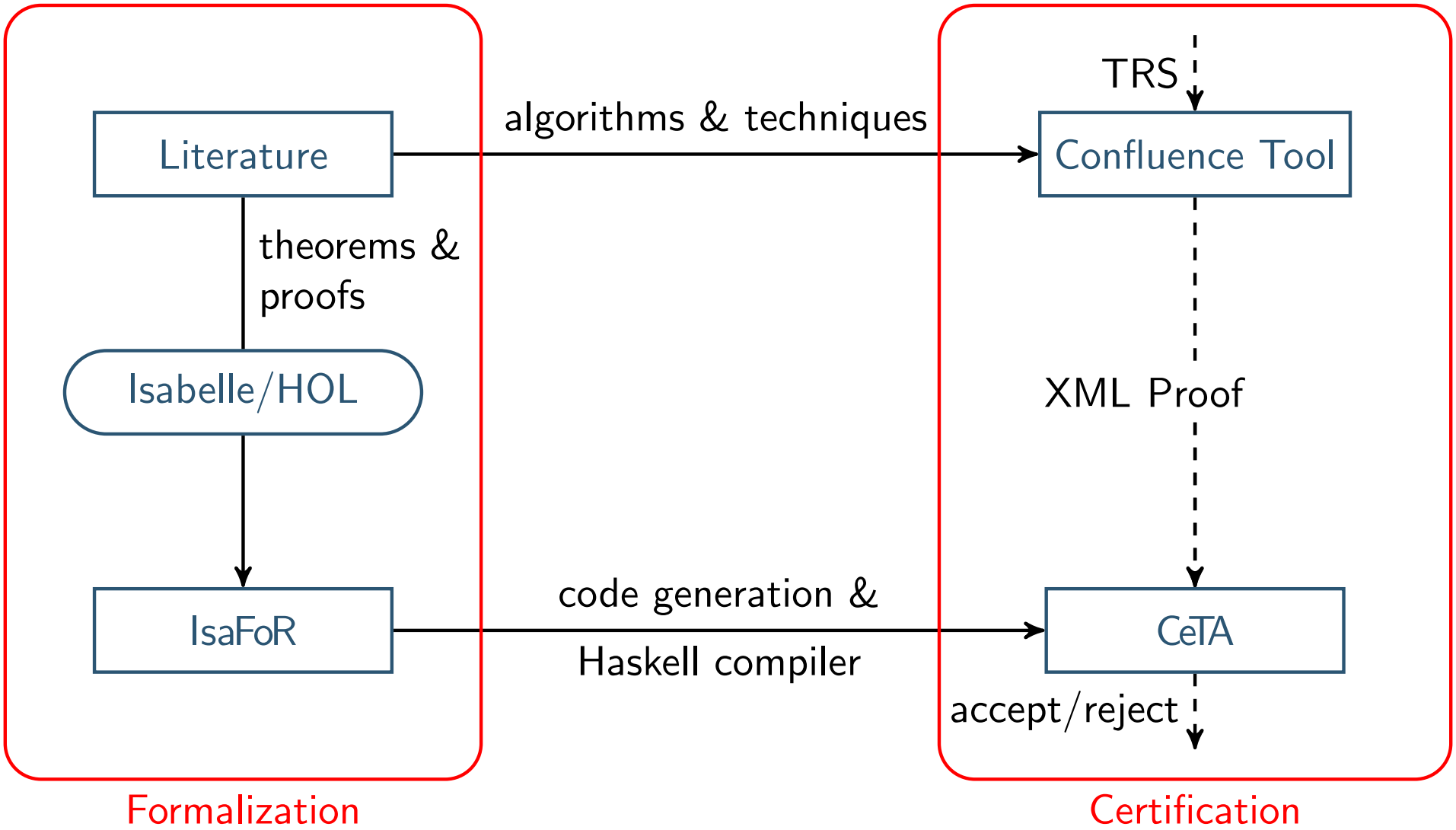
Trusted Tools

- prove correctness of termination/confluence tool
- changes in tool require adaptation of proof
- limits potential for optimization

Trusted Certifiers

- external highly trusted certifier
- checks output of unreliable tools
- checking proofs is usually much easier than finding them
- correctness proven in proof assistant (Isabelle, Coq, . . .)

IsaFoR/CeTA


Literature
algorithms & techniques
TRS
Confluence Tool
theorems &
proofs
Isabelle/HOL
XML Proof
IsaFoR
code generation &
Haskell compiler
CeTA
accept/reject
Formalization
Certification

Summary

Rewriting

- simple but powerful model of computation
- backend for analysis of real-world languages
- higher-order rewriting = term rewriting + λ-calculus
- termination and confluence analysis
- complex powerful tools exist
- . . . and compete in termCOMP and CoCo
- certification to fully trust verification