Towards a Verified Decision Procedure for Confluence of Ground Rewrite Systems in Isabelle/HOL

T. V. H. Prathamesh

joint work with

Bertram Felgenhauer, Aart Middeldorp, Franziska Rapp

University of Innsbruck

7 July 2018

supported by FWF project P 30301

Outline

- FORT and FORTissimo
- Dauchet-Tison Algorithm:Key Ideas
- Theory and Formalisation: An Outline
- CR Checker
- Future Work and Challenges

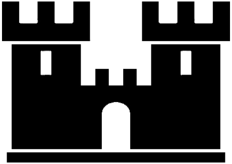
First Order Theory of Rewriting

- First-order logic $\mathcal{L}$ over a language with no function symbols.
- $\mathcal{L}$ consists of following symbols: $\to \quad \to^+ \quad \nrightarrow\!\!\!\!\!\twoheadrightarrow \quad \to_\epsilon \quad \leftrightarrow^* \quad =$
- Models of $\mathcal{L}$ are non-empty finite TRS's $(\mathcal{F}, \mathcal{R})$, where $\mathcal{R}$ is left-linear and right-ground.
- Set of ground terms serve as domain for variables.
- Standard interpretation of predicate symbols in TRS.
- Definable in this language:
  1. $s \downarrow t$: $\exists u.(s \to^* u \land t \to^* u)$.
  2. $CR(t)$: $\forall u.\forall v.(t \to^* u \land t \to^* v) \Rightarrow (u \downarrow v)$.
  3. CR : $\forall t.\ CR(t)$.

Remark

CR above refers only to ground confluence, since the variables range over only ground terms.

FORT

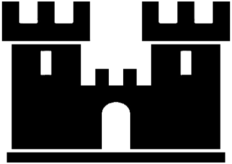
property →  ← TRS

decision mode

↓

yes | no | ?

FORT

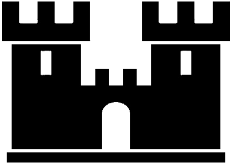
property →  ← TRS

decision mode

$$\forall s\, \exists t\, (s \to^* t \wedge \neg \exists u\, (t \to u)) \implies \exists v\, (s \not\twoheadrightarrow v \vee v \to_\epsilon t)$$

yes | no | ?

FORT

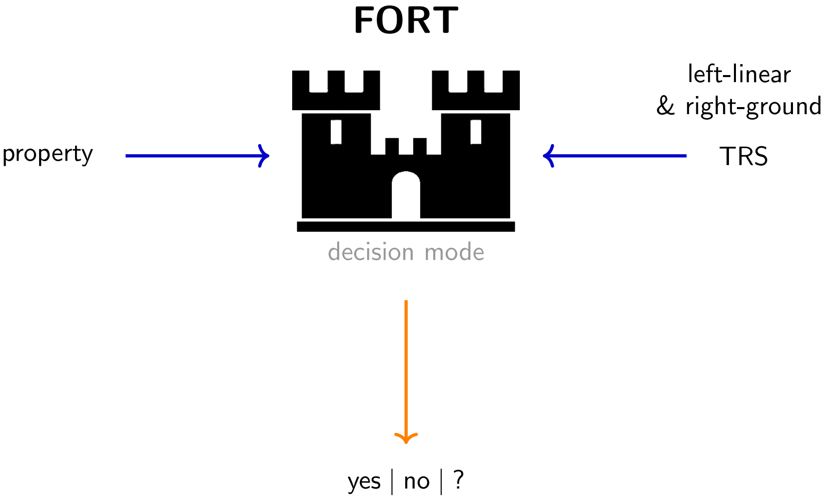
property →  ← TRS

left-linear
& right-ground

decision mode

↓

yes | no | ?



FORT is based on tree automata techniques (Dauchet & Tison, LICS 1990)

FORTissimo

Project Goals

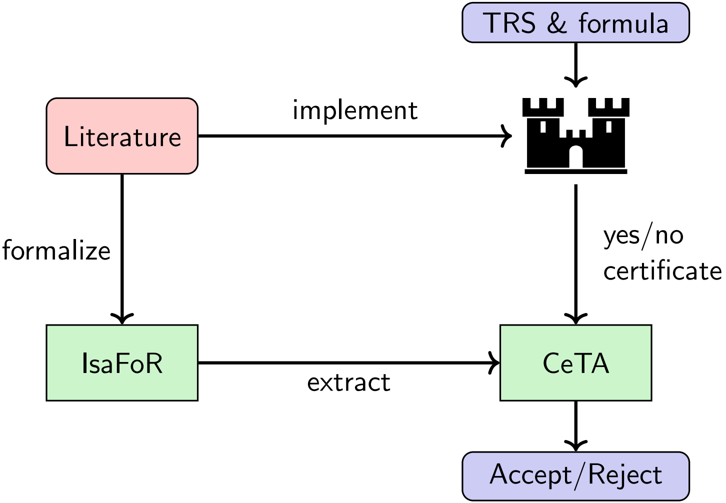
- To improve the efficiency of FORT.
- To find extensions of the decision procedure.
- To certify the output of FORT.
- Involves formalization of the theory of decision procedure in Isabelle/HOL.

FORTissimo

Project Goals

- To improve the efficiency of FORT.
- To find extensions of the decision procedure.
- To certify the output of FORT.
- Involves formalization of the theory of decision procedure in Isabelle/HOL.

Certification Workflow



First Order Theory of Rewriting

- First-order logic $\mathcal{L}$ over a language with no function symbols.
- $\mathcal{L}$ consists of following symbols: $\rightarrow \quad \rightarrow^{+} \quad \nrightarrow\!\!\!\!\rightarrow \quad \rightarrow_{\epsilon} \quad \leftrightarrow^{*} \quad =$
- Models of $\mathcal{L}$ are non-empty finite TRS's $(\mathcal{F}, \mathcal{R})$, where $\mathcal{R}$ is left-linear and right-ground.
- Set of ground terms serve as domain for variables.
- Interpretation as standard.
- Definable in this language:
  1. $s \downarrow t : \exists u.(s \rightarrow^{*} u \wedge t \rightarrow^{*} u)$.
  2. $CR(t) : \forall u.\forall v.(t \rightarrow^{*} u \wedge t \rightarrow^{*} v) \Rightarrow (u \downarrow v)$.
  3. $CR : \forall t.\ CR(t)$.

Remark

CR above refers only to ground confluence, since the variables range over only ground terms.

- **Goal**: To have a verified (ground) confluence checker for ground TRS.

- **Goal**: To have a verified (ground) confluence checker for ground TRS.
- **Motivation**:

- **Goal**: To have a verified (ground) confluence checker for ground TRS.
- **Motivation**:
  - Largely 'self contained fragment' of the decision procedure. (Dauchet-Tison, 87).
  - Test case for **FORT**.

Dauchet-Tison Algorithm: Key Ideas

1. **GTT relation** is a relation on ground terms, using a pair of *tree automata* called GTT (ground tree tranducers):

Dauchet-Tison Algorithm: Key Ideas

1. **GTT relation** is a relation on ground terms, using a pair of *tree automata* called GTT (ground tree tranducers):
   - $\twoheadrightarrow\!\!\!\!\!\!+\;$ is a GTT relation.

Dauchet-Tison Algorithm: Key Ideas

1. **GTT relation** is a relation on ground terms, using a pair of *tree automata* called GTT (ground tree tranducers):
   - $\twoheadrightarrow\!\!\!\!\!\!\!+$ is a GTT relation.
   - Transitive closure and Inverse of GTT relations are GTT relations.
   - Composition of GTT relations is a GTT relation.

Dauchet-Tison Algorithm: Key Ideas

1. **GTT relation** is a relation on ground terms, using a pair of *tree automata* called GTT (ground tree tranducers):
   - $\twoheadrightarrow\!\!\!\!\!\!\!\!+\;\;$ is a GTT relation.
   - Transitive closure and Inverse of GTT relations are GTT relations.
   - Composition of GTT relations is a GTT relation.
2. Associate to a TRS $\mathcal{R}$, a GTT $\mathcal{G}$. $\mathcal{R}(\mathcal{G})$ denotes the GTT relation.

$$\mathcal{R}(\mathcal{G}) = \{(s, t) \mid s \twoheadrightarrow\!\!\!\!\!\!\!+\;\; t\}$$

Dauchet-Tison Algorithm: Key Ideas

1. **GTT relation** is a relation on ground terms, using a pair of *tree automata* called GTT (ground tree trandducers):
   - $\nrightarrow\!\!\!\!\!\rightarrow$ is a GTT relation.
   - Transitive closure and Inverse of GTT relations are GTT relations.
   - Composition of GTT relations is a GTT relation.
2. Associate to a TRS $\mathcal{R}$, a GTT $\mathcal{G}$. $\mathcal{R}(\mathcal{G})$ denotes the GTT relation.

$$\mathcal{R}(\mathcal{G}) = \{(s,t) \mid s \nrightarrow\!\!\!\!\!\rightarrow t\}$$

3. From transitive closure of GTT relation: We get $\mathcal{G}^*$ such that:

$$\mathcal{R}(\mathcal{G}^*) = \{(s,t) \mid s \to^* t\}$$

4. From closure under inverse, obtain $\mathcal{G}^{*-}$ which recognizes $s \; {}^*\!\!\leftarrow t$.

$$\mathcal{R}(\mathcal{G}^{*-}) = \{(s,t) \mid s \; {}^*\!\!\leftarrow t\}$$

Dauchet-Tison Algorithm: Key Ideas(Contd)

- Compose $\mathcal{G}^*$ and $\mathcal{G}^{*-}$ to obtain two GTT's $\mathcal{G}_1$ and $\mathcal{G}_2$, which recognize relations:

$$\uparrow_{\mathcal{R}} = ({}_{\mathcal{R}}^{*}\!\leftarrow \cdot \rightarrow_{\mathcal{R}}^{*})$$

$$\downarrow_{\mathcal{R}} = (\rightarrow_{\mathcal{R}}^{*} \cdot {}_{\mathcal{R}}^{*}\!\leftarrow)$$

- Encode $\mathcal{G}_1$ and $\mathcal{G}_2$ into relations recognizable by a tree automata, using an encoding called RR_2 encoding.
- Do an inclusion check:

$$\mathcal{L}(\mathcal{G}_1) \subseteq \mathcal{L}(\mathcal{G}_2)$$

Remark

RR_2 encoding mentioned above is a special case of an RR_n encoding. RR_n encodings are used to encode **propositional operations**, **quantifiers and variables**, into *recognizable* relations on tree automata, thus leading to a decision procedure for the first-order theory of rewriting.

Decision Procedure: Formalization and Execution

Decision Procedure: Formalization and Execution

- The underlying theory of decision procedure, stands formalized.
- There exists an executable code, with some gaps.
- A sizeable portion of the formalization involved formalizing properties of ground tree tranducers.
- We illustrate some aspects of our formalization by considering the case of GTT composition and transitive closure.

- FORT and FORTissimo
- Dauchet-Tison Algorithm:Key Ideas
- Theory and Formalisation: An Outline
- CR Checker
- Future Work and Challenges

Tree Automata: A Quick Recap

Definition

A tree automata $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ is a quadruple, consisting of a finite signature $\mathcal{F}$, a set of states Q, a set of final states $Q_f \subseteq Q$, and a set of transition rules Δ of the following form:

- $f(q_1, q_2, \ldots, q_n) \to q$, where $f \in \mathcal{F}$ and $q_i, q \in Q$.
- $q \to q'$, where $q, q' \in Q$.

Ground Tree Tranducers

Definition

A *ground tree tranducer* is a pair $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ of tree automata over the same signature $\mathcal{F}$.

Definition

A relation R is *recognized* by the GTT $\mathcal{G}$ if

$$R = \{(s, t) \mid s \rightarrow^*_{\mathcal{A}} \cdot \; {}_{\mathcal{B}}^{*}\!\!\leftarrow t\}$$

Definition

A relation R is called a *GTT relation* if there exists a GTT $\mathcal{G}$ which recognizes R.

Definition

A relation R is called a *GTT relation* if there exists a GTT $\mathcal{G}$ which recognizes R.

Theorem

Composition of a GTT relations is a GTT relation. Transitive closure and inverse of a GTT relation is a GTT relation.

Proof Sketch: Composition

Definition (ϵ-transitions)

$$\Delta_\epsilon(\mathcal{A}, \mathcal{B}) = \{(q, q') \mid \exists \text{ ground } t.\ (t \to_{\mathcal{A}}^* q) \wedge (t \to_{\mathcal{B}}^* q')\ \}$$

Let

$$\mathcal{G}_1 = (\mathcal{F}, Q_1, \Delta_{A_1}, \Delta_{B_1})$$
$$\mathcal{G}_2 = (\mathcal{F}, Q_2, \Delta_{A_2}, \Delta_{B_2})$$

Define:

$$\Delta_A = \Delta_{A_1} \cup \Delta_{A_2} \cup \Delta_\epsilon(\mathcal{B}_1, \mathcal{A}_2)$$
$$\Delta_B = \Delta_{B_1} \cup \Delta_{B_2} \cup \Delta_\epsilon(\mathcal{A}_2, \mathcal{B}_1)$$
$$\mathcal{G} = (\mathcal{F}, Q_1 \cup Q_2, \Delta_A, \Delta_B)$$

The proof further consists of showing that

$$\mathcal{R}(\mathcal{G}) = \mathcal{R}(\mathcal{G}_1) \circ \mathcal{R}(\mathcal{G}_2)$$

Proof Sketch: Transitive Closure

- $\mathcal{G}_i = (\mathcal{F}, Q, \Delta_A^i, \Delta_B^i)$, for $i \geq 1$.
  1. $\Delta_A^1 = \Delta_A$; $\Delta_B^1 = \Delta_B$.
  2. $\Delta_A^{i+1} = \Delta_A^i \cup \Delta_\epsilon(\mathcal{B}_i, \mathcal{A}_i)$; $\Delta_B^{i+1} = \Delta_B^i \cup \Delta_\epsilon(\mathcal{A}_i, \mathcal{B}_i)$.
- There exists an N such that:

$$\forall i \geq N.\ \mathcal{G}_i = \mathcal{G}_N$$

- Remainder of the proof consists of showing that $\mathcal{G}_N$ recognizes the transitive closure of R.

Challenges

- Formal Proof Challenges:
  - Working with different equivalent definitions.

Challenges

- Formal Proof Challenges:
  - Working with different equivalent definitions. e.g. GTT acceptance using standard definition, and inductive definition involving functions and multihole contexts.

Challenges

- Formal Proof Challenges:
  - Working with different equivalent definitions. e.g. GTT acceptance using standard definition, and inductive definition involving functions and multihole contexts.
  - Converting pictorial arguments into formal algebraic arguments.

Challenges

- Formal Proof Challenges:
  - Working with different equivalent definitions. e.g. GTT acceptance using standard definition, and inductive definition involving functions and multihole contexts.
  - Converting pictorial arguments into formal algebraic arguments. e.g. reasoning about multihole contexts using pictorial properties of term trees.

Challenges

- Formal Proof Challenges:
  - Working with different equivalent definitions. e.g. GTT acceptance using standard definition, and inductive definition involving functions and multihole contexts.
  - Converting pictorial arguments into formal algebraic arguments. e.g. reasoning about multihole contexts using pictorial properties of term trees.
- Executable code for GTT composition:

Challenges

- Formal Proof Challenges:
  - Working with different equivalent definitions. e.g. GTT acceptance using standard definition, and inductive definition involving functions and multihole contexts.
  - Converting pictorial arguments into formal algebraic arguments. e.g. reasoning about multihole contexts using pictorial properties of term trees.
- Executable code for GTT composition:
  - Definitions convenient from the perspective of formal proofs, are not the most convenient from an executable code perspective.

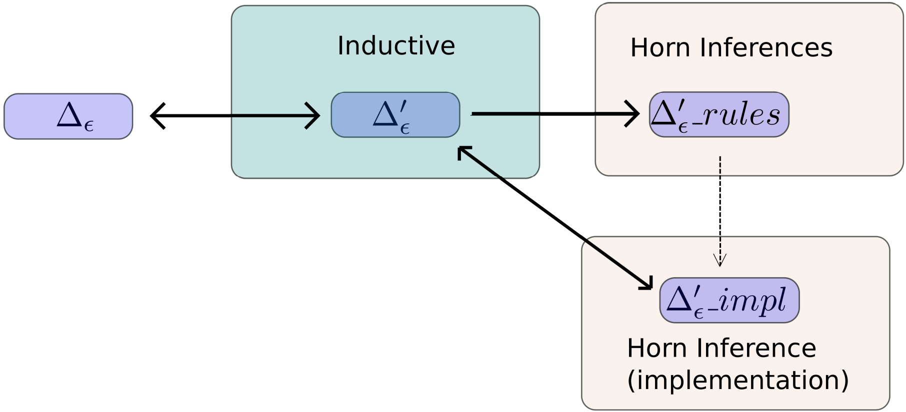
Challenges

- Formal Proof Challenges:
  - Working with different equivalent definitions. e.g. GTT acceptance using standard definition, and inductive definition involving functions and multihole contexts.
  - Converting pictorial arguments into formal algebraic arguments. e.g. reasoning about multihole contexts using pictorial properties of term trees.
- Executable code for GTT composition:
  - Definitions convenient from the perspective of formal proofs, are not the most convenient from an executable code perspective.
  - For instance, Δ_ϵ, as defined before, is not executable.

Challenges

- Formal Proof Challenges:
  - Working with different equivalent definitions. e.g. GTT acceptance using standard definition, and inductive definition involving functions and multihole contexts.
  - Converting pictorial arguments into formal algebraic arguments. e.g. reasoning about multihole contexts using pictorial properties of term trees.
- Executable code for GTT composition:
  - Definitions convenient from the perspective of formal proofs, are not the most convenient from an executable code perspective.
  - For instance, Δ_ϵ, as defined before, is not executable.
  - An implementable version of Δ_ϵ is constructed, which is then formally proved to be equal to Δ_ϵ.

Executable Δ_ϵ



Executable Code: Δ_ϵ

- A generic algorithm for Horn Clauses is defined, for which correctness and termination are proved.
- Δ'_ϵ is defined inductively, and proved to be equivalent to Δ_ϵ.

inductive_set $\Delta_\varepsilon{}'$ :: $('p, 'f)\ ta \Rightarrow ('q, 'f)\ ta \Rightarrow ('p \times 'q)\ set$ **for** $A\ B$ **where**
$\Delta_\varepsilon{}'$*_cong*: $f\ ps \to p \in ta\_rules\ A \Longrightarrow f\ qs \to q \in ta\_rules\ B \Longrightarrow length\ ps = length\ qs \Longrightarrow$
$(\bigwedge i.\ i < length\ qs \Longrightarrow (ps\ !\ i,\ qs\ !\ i) \in \Delta_\varepsilon{}'\ A\ B) \Longrightarrow (p, q) \in \Delta_\varepsilon{}'\ A\ B$
| $\Delta_\varepsilon{}'$*_eps1*: $(p, p') \in ta\_eps\ A \Longrightarrow (p, q) \in \Delta_\varepsilon{}'\ A\ B \Longrightarrow (p', q) \in \Delta_\varepsilon{}'\ A\ B$
| $\Delta_\varepsilon{}'$*_eps2*: $(q, q') \in ta\_eps\ B \Longrightarrow (p, q) \in \Delta_\varepsilon{}'\ A\ B \Longrightarrow (p, q') \in \Delta_\varepsilon{}'\ A\ B$

- The inductive rules of Δ'_ϵ are converted to Horn clauses.

definition $\Delta_\varepsilon{}'\_rules\ \mathcal{A}\ \mathcal{B} = \{uu\_.\ \exists f\ ps\ p\ qs\ q.\ uu\_ = zip\ ps\ qs \to_h (p, q) \wedge f\ ps \to p \in ta\_rules\ \mathcal{A} \wedge f\ qs \to q \in ta\_rules\ \mathcal{B} \wedge length\ ps = length\ qs\} \cup \{[(p, q)] \to_h (p', q)\ |p\ p'\ q.\ (p, p') \in ta\_eps\ \mathcal{A}\} \cup \{[(p, q)] \to_h (p, q')\ |p\ q\ q'.\ (q, q') \in ta\_eps\ \mathcal{B}\}$

Executable Code: Δ_ϵ

- We prove that Horn inferences characterise Δ'_ϵ.

sublocale *horn* $\Delta_\varepsilon{}'$*_rules* $\mathcal{A}$ $\mathcal{B}$ **.**

lemma $\Delta_\varepsilon{}'\ \mathcal{A}\ \mathcal{B} =$ *saturate* ($\Delta_\varepsilon{}'$*_rules* $\mathcal{A}$ $\mathcal{B}$)

- An implementable variant of Δ'_ϵ is defined. This involves using an implementable version of saturated function, and two other functions to generate the inferences.

definition $\Delta_\varepsilon{}'$*_impl* $\mathcal{A}$ $\mathcal{B}$ = *saturate_impl* ($\Delta_\varepsilon{}'$*_infer0* $\mathcal{A}$ $\mathcal{B}$) ($\Delta_\varepsilon{}'$*_infer1* $\mathcal{A}$ $\mathcal{B}$)

Proving Soundness

- Δ'_ϵ is equal to Δ_ϵ.

lemma $\Delta_\varepsilon\ \mathcal{A}\ \mathcal{B} = \Delta_\varepsilon{}'\ \mathcal{A}\ \mathcal{B}$

- Δ'_ϵ_*impl* computes Δ'_ϵ.

lemma $\Delta_\varepsilon{}'\_impl\ \mathcal{A}\ \mathcal{B} = Some\ xs \Longrightarrow set\ xs = \Delta_\varepsilon{}'\ (ta\_of\ \mathcal{A})\ (ta\_of\ \mathcal{B})$

Partially Verified CR Check

Verified executable parts

- TRS to GTT, GTT transitive closure, GTT composition
- GTT to RR_2 conversion
- tree automata language containment

Partially Verified CR Check

Verified executable parts

- TRS to GTT, GTT transitive closure, GTT composition
- GTT to RR_2 conversion
- tree automata language containment

Experiments on ground Cops

timeout	YES	NO	MAYBE	total
60s	14	39	45	98
600s	20	52	26	98

Partially Verified CR Check

Verified executable parts

- TRS to GTT, GTT transitive closure, GTT composition
- GTT to RR_2 conversion
- tree automata language containment

Experiments on ground Cops

timeout	YES	NO	MAYBE	total
60s	14	39	45	98
600s	20	52	26	98

Gaps

- compose correctness results, check side conditions
- signature extension
- ground CR $\neq$ CR

Conclusions and Future Work

- Total length of formalization: 7200 lines.
- Future Work: Certificates for FORT.
  - First-order formula manipulation.
  - Transition from GTT to RR_n.
  - Certifier reproduces tree automata.

Conclusions and Future Work

- Total length of formalization: 7200 lines.
- Future Work: Certificates for FORT.
  - First-order formula manipulation.
  - Transition from GTT to RR_n.
  - Certifier reproduces tree automata.

Thank You.