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: $\rightarrow$ $\rightarrow^+$ $\equiv$ $\rightarrow_\epsilon$ $\leftrightarrow^*$ $=$
- 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. $s \downarrow t : \exists u. (s \rightarrow^* u \land t \rightarrow^* u)$.
  2. $CR(t) : \forall u. \forall v. (t \rightarrow^* u \land t \rightarrow^* v) \Rightarrow (u \downarrow v)$.
  3. $CR : \forall t. CR(t)$.

Remark

$CR$ above refers only to ground confluence, since the variables range over only ground terms.
FORT is based on tree automata techniques (Dauchet & Tison, LICS 1990)

property → FORTissimo

decision mode

yes | no | ?

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

∀s ∃t (s →* t ∧ ¬∃u (t → u))
⇒ ∃v (s ⊢ v ∨ v →_ε t)

decision mode

property

yes | no | ?

TRS
FORT and FORTissimo

FORT

property

left-linear
& right-ground

TRS

decision mode

∀s ∃t (s → * t ∧ ¬∃u (t → u)) = ⇒ ∃v (s → /parallel.short v ∨ v → ϵ t)

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

Towards Verified CR for Ground TRSs
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.
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

1. Literature
2. Formalize
3. IsaFoR
4. Extract
5. CeTA
6. Implement
7. TRS & formula
8. Yes/No certificate
9. Accept/Reject
First Order Theory of Rewriting

- First-order logic $\mathcal{L}$ over a language with no function symbols.
- $\mathcal{L}$ consists of following symbols: $\rightarrow$ $\rightarrow^+$ $\Rightarrow$ $\rightarrow^\epsilon$ $\leftrightarrow^*$ $=$
- 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. $s \downarrow t : \exists u. (s \rightarrow^* u \land t \rightarrow^* u)$.
  2. $CR(t) : \forall u. \forall v. (t \rightarrow^* u \land t \rightarrow^* v) \Rightarrow (u \downarrow v)$.
  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.
FORT and FORTissimo

Dauchet-Tison Algorithm: Key Ideas

Theory and Formalisation: An Outline

CR Checker

Future Work and Challenges
Dauchet-Tison Algorithm: Key Ideas

1. **GTT relation** is a relation on ground terms, using a pair of *tree automata* called GTT (ground tree transducers):
Dauchet-Tison Algorithm: Key Ideas

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

\[
\mathcal{R}(\mathcal{G}) = \{(s, t) \mid s \rightarrow t\}
\]
1. **GTT relation** is a relation on ground terms, using a pair of *tree automata* called GTT (ground tree transducers):
   - $\leftrightarrow$ 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.

   $$\mathcal{R}(\mathcal{G}) = \{(s, t) \mid s \leftrightarrow t\}$$

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

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

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 $G^*$ and $G^{*-}$ to obtain two GTT's $G_1$ and $G_2$, which recognize relations:

  $\uparrow R = (\overleftarrow{R} \cdot \overrightarrow{R}^*)$

  $\downarrow R = (\overrightarrow{R}^* \cdot \overleftarrow{R})$

- Encode $G_1$ and $G_2$ into relations recognizable by a tree automata, using an encoding called $RR_2$ encoding.

- Do an inclusion check:

  $\mathcal{L}(G_1) \subseteq \mathcal{L}(G_2)$

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.
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 transducers. We illustrate some aspects of our formalization by considering the case of GTT composition and transitive closure.
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 transducers.

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 $\Delta$ of the following form:

- $f(q_1, q_2, \ldots, q_n) \rightarrow q$, where $f \in \mathcal{F}$ and $q_i, q \in Q$.
- $q \rightarrow q'$, where $q, q' \in Q$. 
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 $G$ which recognizes $R$. 
**Definition**

A relation $R$ is called a **GTT relation** if there exists a GTT $G$ 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 (\(\epsilon\)-transitions)

\[
\Delta_\epsilon(A, B) = \{ (q, q') \mid \exists \text{ ground } t. (t \rightarrow^*_A q) \wedge (t \rightarrow^*_B q') \}
\]

Let

\[
G_1 = (F, Q_1, \Delta_{A_1}, \Delta_{B_1}) \\
G_2 = (F, Q_2, \Delta_{A_2}, \Delta_{B_2})
\]

Define:

\[
\Delta_A = \Delta_{A_1} \cup \Delta_{A_2} \cup \Delta_\epsilon(B_1, A_2) \\
\Delta_B = \Delta_{B_1} \cup \Delta_{B_2} \cup \Delta_\epsilon(A_2, B_1) \\
G = (F, Q_1 \cup Q_2, \Delta_A, \Delta_B)
\]

The proof further consists of showing that

\[
R(G) = R(G_1) \circ R(G_2)
\]
Proof Sketch: Transitive Closure

\( G_i = (F, Q, \Delta^i_A, \Delta^i_B) \), for \( i \geq 1 \).

1. \( \Delta^1_A = \Delta_A; \Delta^1_B = \Delta_B \).
2. \( \Delta^{i+1}_A = \Delta^i_A \cup \Delta_\epsilon(B_i, A_i); \Delta^{i+1}_B = \Delta^i_B \cup \Delta_\epsilon(A_i, B_i) \).

There exists an \( N \) such that:

\[ \forall i \geq N. \ G_i = G_N \]

Remainder of the proof consists of showing that \( G_N \) 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, $\Delta_\epsilon$, 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, $\Delta_\epsilon$, as defined before, is not executable.
  - An implementable version of $\Delta_\epsilon$ is constructed, which is then formally proved to be equal to $\Delta_\epsilon$. 
Executable $\Delta_\epsilon$

- $\Delta_\epsilon$
- $\Delta_\epsilon'$
- $\Delta_\epsilon'$-rules
- $\Delta_\epsilon'$-impl

Inductive

Horn Inferences

Horn Inference (implementation)
A generic algorithm for Horn Clauses is defined, for which correctness and termination are proved.

\( \Delta'_\epsilon \) is defined inductively, and proved to be equivalent to \( \Delta_\epsilon \).

The inductive rules of \( \Delta'_\epsilon \) are converted to Horn clauses.
We prove that Horn inferences characterise $\Delta'_\varepsilon$.

**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 $\Delta'_\varepsilon$ 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

- $\Delta'_{\epsilon}$ is equal to $\Delta_{\epsilon}$.

  \[
  \text{lemma } \Delta_{\epsilon} \ A \ B = \Delta'_{\epsilon} \ A \ B
  \]

- $\Delta'_{\epsilon-impl}$ computes $\Delta'_{\epsilon}$.

  \[
  \text{lemma } \Delta'_{\epsilon-impl} \ A \ B = \text{Some } xs \implies \text{set } xs = \Delta'_{\epsilon}(\text{ta_of } A) \ (\text{ta_of } B)
  \]
- FORT and FORTissimo

- Dauchet-Tison Algorithm: Key Ideas

- Theory and Formalisation: An Outline

- CR Checker

- Future Work and Challenges
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**

<table>
<thead>
<tr>
<th>timeout</th>
<th>YES</th>
<th>NO</th>
<th>MAYBE</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>60s</td>
<td>14</td>
<td>39</td>
<td>45</td>
<td>98</td>
</tr>
<tr>
<td>600s</td>
<td>20</td>
<td>52</td>
<td>26</td>
<td>98</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>timeout</th>
<th>YES</th>
<th>NO</th>
<th>MAYBE</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>60s</td>
<td>14</td>
<td>39</td>
<td>45</td>
<td>98</td>
</tr>
<tr>
<td>600s</td>
<td>20</td>
<td>52</td>
<td>26</td>
<td>98</td>
</tr>
</tbody>
</table>

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