Reducing Joinability to Confluence: How to Preserve Shallowness and Linearity

Luis Moraes and Rakesh Verma

University of Houston

September 8th, 2016
Motivation

- We have a reduction: $A \leq_P B$
- How is it helpful?
  - $A$ is undecidable $\implies B$ is undecidable.
  - $B$ is decidable $\implies A$ is decidable.
- A result for one property can be reused for another.
Preliminaries

- **Joinability:** Given a TRS $\mathcal{R}$ and two terms $s, t$, does there exist a term $z$ such that $s \rightarrow^* z \leftarrow^* t$?

- **Confluence:** Given a TRS $\mathcal{R}$. For any two terms $s, t$ that have a common ancestor ($s \leftarrow^* a \rightarrow^* t$), does there exist a term $z$ such that $s \rightarrow^* z \leftarrow^* t$?
- **Linear TRS**: A variable may only appear once on each side of a rule.
- **Shallow TRS**: Variables can only appear at depth 0 or 1 in a rule.
Reduction

Joinability : \( R : s \downarrow t \) ?
\[ \Downarrow \]
Confluence : \( R' : \) confluent ?

**Challenge is insuring:** \( s \downarrow t \) under \( R \) \( \iff \) \( R' \) is confluent
Σ' = Σ ∪ \{h, h', a\}

\[ R_1 = \{c \rightarrow h'(h(s, t), c) | c \in \Sigma\} \]
\[ \cup \{f(x_1 \ldots x_n) \rightarrow h'(h(s, t), f(x_1 \ldots x_n))\} \]

\[ R' = R \cup R_1 \cup \{h(x, x) \rightarrow a\} \cup \{h'(a, x) \rightarrow a\} \]

**Note:** Any term \(u\) reaches \(h(h'(s, t), u)\).

**Note 2:** If \(s \downarrow t\) then \(h'(s, t) \rightarrow^* a\). Any two terms join.

\[ \Sigma' = \Sigma \cup \{ h, h', a \} \]

\[ \mathcal{R}_1 = \{ c \rightarrow h'(h(s, t), c) | c \in \Sigma \} \]
\[ \cup \{ f(x_1 \ldots x_n) \rightarrow h'(h(s, t), f(x_1 \ldots x_n)) \} \]

\[ \mathcal{R}' = \mathcal{R} \cup \mathcal{R}_1 \cup \{ h(x, x) \rightarrow a \} \cup \{ h'(a, x) \rightarrow a \} \]

Violates right-shallow restriction

\[ \Sigma' = \Sigma \cup \{ h, h', a \} \]

\[ R_1 = \{ c \to h'(h(s, t), c) | c \in \Sigma \} \]
\[ \cup \{ f(x_1 \ldots x_n) \to h'(h(s, t), f(x_1 \ldots x_n)) \} \]

\[ R' = R \cup R_1 \cup \{ h(x, x) \to a \} \cup \{ h'(a, x) \to a \} \]

Violates right-shallow restriction

Violates left-linear restriction
In Verma [2012], joinability was shown to be undecidable for linear and left-shallow TRS.
  ▶ Not able to determine confluence for the same class through the reduction.
Another Reduction.
Intuition

- Suppose that instead of $s, t$ we had $0, 1$.
- Suppose we assigned each function symbol a binary string.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Binary String</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>00</td>
</tr>
<tr>
<td>b</td>
<td>01</td>
</tr>
<tr>
<td>f</td>
<td>10</td>
</tr>
<tr>
<td>g</td>
<td>11</td>
</tr>
</tbody>
</table>
Suppose that instead of $s, t$ we had 0, 1.

Suppose we assigned each function symbol a binary string.

```
<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
<td>00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>11</td>
<td>01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
\begin{tikzpicture}
  \node (f) at (0,0) {f};
  \node (a) at (-1,-1) {a};
  \node (b) at (1,-1) {b};
  \draw (f) -- (a);
  \draw (f) -- (b);
  \node (g) at (0,-2) {g};
  \node (b) at (-1,-3) {b};
  \node (a) at (1,-3) {a};
  \draw (g) -- (b);
  \draw (g) -- (a);
  \node (a) at (0,-1) {00};
  \node (b) at (0,-2) {01};
  \node (c) at (0,-3) {10};
  \node (d) at (0,-4) {11};
\end{tikzpicture}
```
Intuition

- Suppose that instead of $s, t$ we had 0, 1.
- Suppose we assigned each function symbol a binary string.

```
<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>
```

if $0 \rightarrow z \leftarrow 1...$
Intuition

- Suppose that instead of \( s, t \) we had 0, 1.
- Suppose we assigned each function symbol a binary string.

\[
\begin{array}{c|c}
\text{a} & 00 \\
\text{b} & 01 \\
\text{f} & 10 \\
\text{g} & 11 \\
\end{array}
\]

if 0 \( \rightarrow \) z \( \leftarrow \) 1...
To use $s, t$ as 0’s and 1’s we must flatten them.

We introduce rules in a manner similar to tree automata.

An example can be found in Godoy et al. [2003].

$$s = f \quad c_s \quad \begin{array}{c}
\wedge \\
 a \\
 b
\end{array}$$
Flattening

- To use $s, t$ as 0’s and 1’s we must flatten them.
- We introduce rules in a manner similar to tree automata.
- An example can be found in Godoy et al. [2003].

\[
\begin{align*}
    s &= f \\
    c_s &\rightarrow f \\
    \begin{array}{c}
    \wedge \\
    a \\
    \wedge \\
    c_a \\
    b \\
    \wedge \\
    c_b \\
    \end{array} &\rightarrow f(c_a, c_b)
\end{align*}
\]
Flattening

- To use $s, t$ as 0’s and 1’s we must flatten them.
- We introduce rules in a manner similar to tree automata.
- An example can be found in Godoy et al. [2003].

$$s = f$$

$$\text{c}_s \rightarrow f \rightarrow f$$

$$\text{c}_s \rightarrow f(\text{c}_a, \text{c}_b)$$
$$\text{c}_a \rightarrow a$$
$$\text{c}_b \rightarrow b$$
We also add a common ancestor to $c_s$, $c_t$.

Thus, we now have the following rules:

\[
\Sigma_1 := \Sigma \cup \Sigma_{flat} \cup \{\alpha : 0\}
\]
\[
\mathcal{R}_1 := \mathcal{R} \cup \mathcal{R}_{flat} \cup \{\alpha \rightarrow c_s, \alpha \rightarrow c_t\}
\]
We use the first $B$ positions of the $h_i$ symbols to hold the binary string. $h_i$ varies from 0 to $M$ (max arity in $\Sigma_1$).

$$\Sigma_{code} := \{ h_i : B + i \mid 0 \leq i \leq M \}$$

$$R_{code} := \{ f(x_1 \cdots x_n) \rightarrow h_n(c_{f_1} \cdots c_{f_B}, x_1 \cdots x_n) \mid f \in \Sigma_1 \}$$

$$\Sigma_2 := \Sigma_1 \cup \Sigma_{code}$$

$$R_2 := R_1 \cup R_{code}$$
If $c_s \downarrow c_t$ then $f(a, b) \downarrow g(b, a)$
However, $f(a, b)$ still cannot join $a$ or $b$
Structural Equivalence

However, $f(a, b)$ still cannot join $a$ or $b$
Requires *structural equivalence*
  i.e. the same set of positions
We introduce a *dummy symbol* that will be used to generate new positions.

\[ R_{ex} := \{ h_n(x_1 \cdots x_{B+n}) \rightarrow h_{n+1}(x_1 \cdots x_{B+n}, \delta) \} \]

\[ \Sigma' := \Sigma_2 \cup \{ \delta : 0 \} \]

\[ R' := R_2 \cup R_{ex} \]
Extension Rules – In Practice

\[ a \xrightarrow{*} h_0 \]

\[ C_s \quad C_s \quad C_s \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>000</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>001</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>010</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>011</td>
<td></td>
</tr>
<tr>
<td>f'</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>(\delta)</td>
<td>101</td>
<td></td>
</tr>
</tbody>
</table>
Extension Rules – In Practice

\[ a \rightarrow^* h_1 \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>000</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>001</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>010</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>011</td>
<td></td>
</tr>
<tr>
<td>f'</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>δ</td>
<td>101</td>
<td></td>
</tr>
</tbody>
</table>
Extension Rules – In Practice

\[ a \xrightarrow{\ast} h_1 \]

\[ f'(b) \xrightarrow{\ast} h_1 \]

\begin{array}{|c|c|}
\hline
a & 000 \\
b & 001 \\
f' & 100 \\
\hline
\end{array}

\begin{array}{|c|c|}
\hline
f & 010 \\
g & 011 \\
\delta & 101 \\
\hline
\end{array}
Extension Rules – In Practice

\[ a \xrightarrow{\ast} h_1 \]

\[ f'(f'(b)) \xrightarrow{\ast} h_1 \]
Extension Rules – In Practice

\[ a \stackrel{\ast}{\rightarrow} h_1 \]

\[ c_s \quad c_s \quad c_s \quad h_0 \quad h_0 \]

\[ c_t \quad c_s \quad c_t \quad c_t \quad c_s \quad c_s \quad c_t \]

\[ g(a, b) \stackrel{\ast}{\rightarrow} h_1 \]

\[ c_t \quad c_s \quad c_s \quad h_0 \quad h_0 \]

\[ c_s \quad c_s \quad c_s \quad c_s \quad c_s \quad c_t \]
Proofs.
Proofs.

(Sketch)
Every Term Joins

**Lemma**
Every term $t \in \mathcal{T}(\Sigma', X)$ reaches a code term.

**Lemma**
Any pair of code terms can be rewritten into structurally equivalent code terms.

**Lemma**
If $c_s \Downarrow c_t$ then any two terms can be joined.
Minimal Proofs

Definition
A derivation is a sequence of terms obtained through successive rewrite steps: \( u_1 \rightarrow u_2 \rightarrow \cdots \rightarrow u_{n-1} \rightarrow u_n \).

Definition
A *minimal proof* of joinability between two terms \( t_1, t_2 \) is a pair of derivations demonstrating \( t_1 \xrightarrow{\ast} z \leftarrow t_2 \) for some \( z \) such that there exists no other pair with a fewer number of rewrite steps.
Lemma

A minimal proof of joinability for $c_s \downarrow c_t$ performs no rewrites on binary string subterms.
Lemma
A minimal proof of joinability for \( c_s \downarrow c_t \) performs no \( R_{ex} \) rewrites.

\[
\begin{align*}
  u_1 \to^* u_i \to^* z \leftarrow^* v_i & \leftarrow^* v_1 \\
  h_n & \quad h_n \\
  x_1 \cdots \delta & \quad x_1 \cdots \delta
\end{align*}
\]
Lemma

\[ c_s \downarrow c_t \text{ under } \mathcal{R}_1 \text{ iff } c_s \downarrow c_t \text{ under } \mathcal{R}'. \]

\[
\begin{align*}
  u_1 & \rightarrow u_2 \rightarrow u_i \rightarrow u_n \\
  \downarrow \pi \\
  u'_1 & \rightarrow u'_2 \rightarrow u'_i \rightarrow u'_n
\end{align*}
\]

Use mapping \( \pi \) (maps to “pure” terms) to obtain a proof in \( \mathcal{R}_1 \).
Lemma

c_s \downarrow c_t \text{ under } R_1 \iff c_s \downarrow c_t \text{ under } R'.

\[
\begin{align*}
\pi(u_1) & \mapsto \pi(u_2) \mapsto \pi(u_i) \mapsto \pi(u_n) \\
\downarrow \pi & \\
\pi(u_1') & \mapsto \pi(u_2') \mapsto \pi(u_i') \mapsto \pi(u_n')
\end{align*}
\]

\(R\) code is erased.
\(\pi(u_i) = \pi(u_{i+1})\).

\(R_1\) steps still valid.
\(\pi(u_i) \rightarrow \pi(u_{i+1})\).

Use mapping \(\pi\) (maps to “pure” terms) to obtain a proof in \(R_1\).
Conclusion

**Theorem**

*Joinability reduces to confluence while preserving linearity and shallowness restrictions.*

**Proof.**

(i) If \( s \downarrow t \) under \( \mathcal{R} \) then any two terms join under \( \mathcal{R}' \). In particular, terms with a common ancestor join. Thus, \( \mathcal{R}' \) is confluent. Since all the new rules are linear and flat, the resulting TRS preserves linearity and shallowness.

(ii) If \( \mathcal{R}' \) is confluent, then \( c_s \downarrow c_t \) since they have a common ancestor. We know \( s \downarrow t \) under \( \mathcal{R} \) (same as \( c_s \downarrow c_t \) under \( \mathcal{R}_1 \)).
References

