Reducing Joinability to Confluence: How to Preserve Shallowness and Linearity[1]

Luis Moraes and Rakesh Verma

University of Houston

September 8th, 2016

[1] Research supported in part by NSF Grants DUE 1241772 and CNS 1319212

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 $s \xrightarrow{*} z \xleftarrow{*} t$?
- **Confluence:** Given a TRS $\mathcal{R}$. For any two terms s, t that have a common ancestor ($s \xleftarrow{*} a \xrightarrow{*} t$), does there exist a term z such that $s \xrightarrow{*} z \xleftarrow{*} t$?

Preliminaries – cont.

- **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

$$\begin{array}{lcl} \text{Joinability} & : \mathcal{R} \quad : & s \downarrow t \ ? \\ & \Downarrow & \\ \text{Confluence} & : \mathcal{R}' : & \text{confluent} \ ? \end{array}$$

Challenge is insuring: $s \downarrow t$ under $\mathcal{R} \iff \mathcal{R}'$ is confluent

Previous Reduction – Verma [2009]

$$\Sigma' = \Sigma \cup \{h, h', a\}$$

$$\begin{aligned} \mathcal{R}_1 = &\{c \to h'(h(s,t), c) | c \in \Sigma\} \\ &\cup \{f(x_1 \ldots x_n) \to h'(h(s,t), f(x_1 \ldots x_n))\} \end{aligned}$$

$$\mathcal{R}' = \mathcal{R} \cup \mathcal{R}_1 \cup \{h(x,x) \to a\} \cup \{h'(a,x) \to a\}$$

Note: Any term u reaches $h(h'(s,t), u)$.
Note 2: If $s \downarrow t$ then $h'(s,t) \xrightarrow{*} a$. Any two terms join.

Previous Reduction – Verma [2009] – Problems

$$\Sigma' = \Sigma \cup \{h, h', a\}$$

$$\begin{aligned}\mathcal{R}_1 = {} & \{c \to h'(h(s,t), c) | c \in \Sigma\} \\ & \cup \{f(x_1 \dots x_n) \to h'(h(s,t), \boxed{f(x_1 \dots x_n))}\}\end{aligned}$$

$$\mathcal{R}' = \mathcal{R} \cup \mathcal{R}_1 \cup \{h(x,x) \to a\} \cup \{h'(a,x) \to a\}$$

Violates right-shallow restriction

Previous Reduction – Verma [2009] – Problems

$$\Sigma' = \Sigma \cup \{h, h', a\}$$

$$\begin{aligned}\mathcal{R}_1 = &\{c \to h'(h(s,t), c) | c \in \Sigma\} \\ &\cup \{f(x_1 \ldots x_n) \to h'(h(s,t), \boxed{f(x_1 \ldots x_n))}\}\end{aligned}$$

$$\mathcal{R}' = \mathcal{R} \cup \mathcal{R}_1 \cup \{\boxed{h(x,x)} \to a\} \cup \{h'(a,x) \to a\}$$

Violates right-shallow restriction

Violates left-linear restriction

Previous Reduction – Verma [2009] – Problems

- 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.

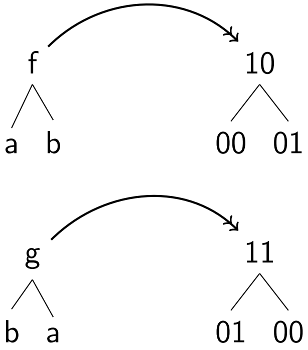
Intuition

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

a	00
b	01
f	10
g	11

Intuition

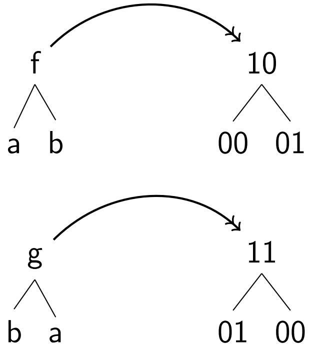
- Suppose that instead of s, t we had $0, 1$.
- Suppose we assigned each function symbol a binary string.



a	00
b	01
f	10
g	11

Intuition

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

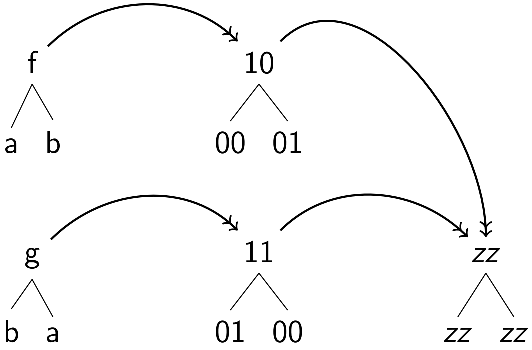


a	00
b	01
f	10
g	11

if $0 \xrightarrow{*} z \xleftarrow{*} 1 \ldots$

Intuition

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



a	00
b	01
f	10
g	11

if $0 \xrightarrow{*} z \xleftarrow{*} 1 \ldots$

Flattening

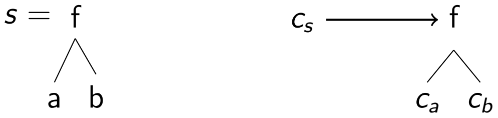
- To use s, t as 0's and 1's we must flatten them.
- We introduce rules in a manner similar to tree automata.
- An example can be found in Godoy et al. [2003].

$s =$

c_s

Flattening

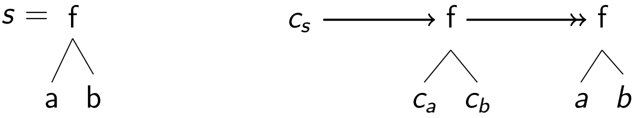
- To use s, t as 0's and 1's we must flatten them.
- We introduce rules in a manner similar to tree automata.
- An example can be found in Godoy et al. [2003].



$$c_s \to f(c_a, c_b)$$

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].



$$
\begin{aligned}
c_s &\to f(c_a, c_b)\\
c_a &\to a\\
c_b &\to b
\end{aligned}
$$

Flattening Rules and Common Ancestor

- We also add a common ancestor to c_s, c_t.
- Thus, we now have the following rules:

$$\Sigma_1 := \Sigma \cup \Sigma_{flat} \cup \{\alpha : 0\}$$
$$\mathcal{R}_1 := \mathcal{R} \cup \mathcal{R}_{flat} \cup \{\alpha \to c_s,\ \alpha \to c_t\}$$

Code Rules

- 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 Σ_1).

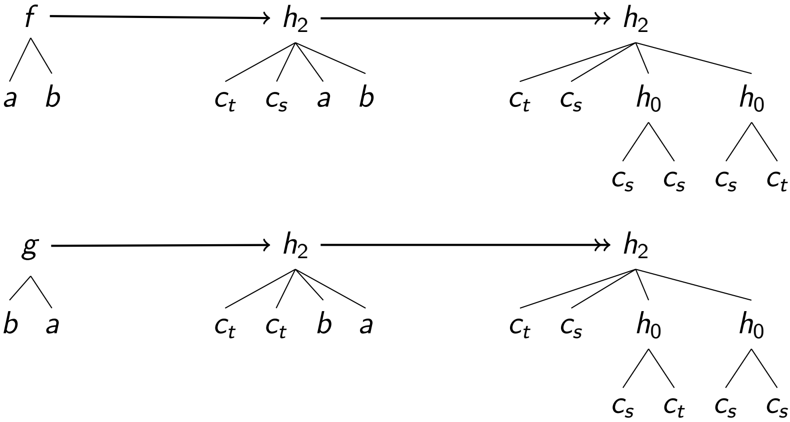
$$\Sigma_{code} := \{h_i : B + i \mid 0 \leq i \leq M\}$$

$$\mathcal{R}_{code} := \{f(x_1 \cdots x_n) \to h_n(c_{f_1} \cdots c_{f_B}, x_1 \cdots x_n) | f \in \Sigma_1\}$$

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

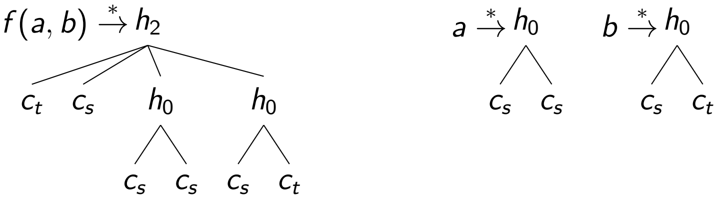
$$\mathcal{R}_2 := \mathcal{R}_1 \cup \mathcal{R}_{code}$$

Code Rules – In Practice



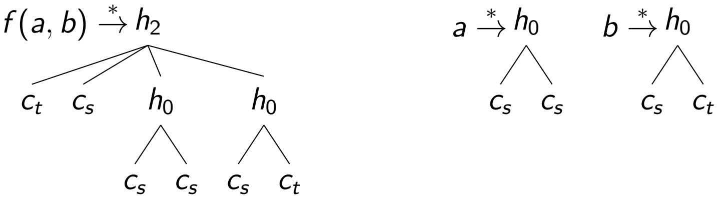
If $c_s \downarrow c_t$ then $f(a, b) \downarrow g(b, a)$

Structural Equivalence



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

Extension Rules

- We introduce a *dummy symbol* that will be used to generate new positions.

$$\mathcal{R}_{ex} := \{h_n(x_1 \cdots x_{B+n}) \to h_{n+1}(x_1 \cdots x_{B+n}, \delta)\}$$
$$\Sigma' := \Sigma_2 \cup \{\delta : 0\}$$
$$\mathcal{R}' := \mathcal{R}_2 \cup \mathcal{R}_{ex}$$

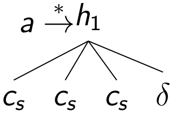
Extension Rules – In Practice

$a \xrightarrow{*} h_0$

c_s c_s c_s

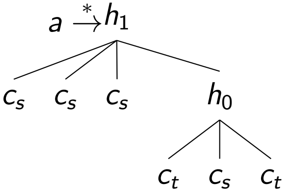
a	000
b	001
f	010
g	011
f'	100
δ	101

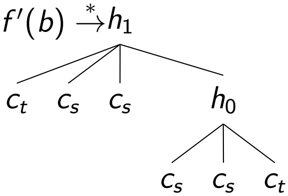
Extension Rules – In Practice



a	000
b	001
f	010
g	011
f'	100
δ	101

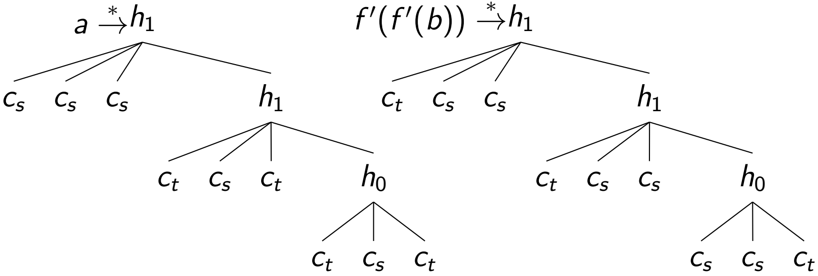
Extension Rules – In Practice



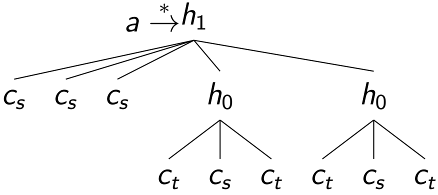


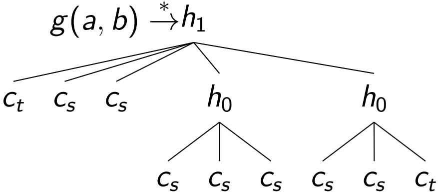
a	000
b	001
f	010
g	011
f'	100
δ	101

Extension Rules – In Practice



Extension Rules – In Practice





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: $u_1 \rightarrow u_2 \rightarrow \cdots \rightarrow u_{n-1} \rightarrow u_n$.

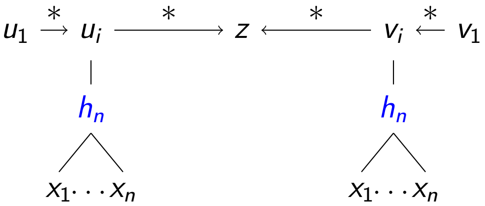
Definition

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

Minimal Proofs – cont

Lemma

A minimal proof of joinability for $c_s \downarrow c_t$ performs no rewrites on binary string subterms.



Minimal Proofs – cont

Lemma

A minimal proof of joinability for $c_s \downarrow c_t$ performs no $\mathcal{R}_{ex}$ rewrites.

$$
\begin{tikzcd}
u_1 \arrow[r, "*"] & u_i \arrow[rr, "*"] \arrow[d, dash] & & z & & v_i \arrow[ll, "*"'] \arrow[d, dash] & v_1 \arrow[l, "*"'] \\
& h_n \arrow[d, dash] & & & & h_n \arrow[d, dash] & \\
& x_1 \cdots \delta & & & & x_1 \cdots \delta &
\end{tikzcd}
$$

Minimal Proofs – cont

Lemma

$c_s \downarrow c_t$ under $\mathcal{R}_1$ iff $c_s \downarrow c_t$ under $\mathcal{R}'$.

$$
\begin{array}{c}
u_1 \xrightarrow{*} u_2 \xrightarrow{*} u_i \xrightarrow{*} u_n \\
\Downarrow \pi \\
u_1' \xrightarrow{*} u_2' \xrightarrow{*} u_i' \xrightarrow{*} u_n'
\end{array}
$$

Use mapping π (maps to "pure" terms) to obtain a proof in $\mathcal{R}_1$.

Minimal Proofs – cont

Lemma

$c_s \downarrow c_t$ under $\mathcal{R}_1$ iff $c_s \downarrow c_t$ under $\mathcal{R}'$.

$$\begin{array}{c} u_1 \stackrel{*}{\rightarrow} u_2 \stackrel{*}{\rightarrow} u_i \stackrel{*}{\rightarrow} u_n \\ \Downarrow \pi \\ u'_1 \stackrel{*}{\rightarrow} u'_2 \stackrel{*}{\rightarrow} u'_i \stackrel{*}{\rightarrow} u'_n \end{array}$$

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

$\mathcal{R}_1$ steps still valid.
$\pi(u_i) \rightarrow \pi(u_{i+1})$.

Use mapping π (maps to "pure" terms) to obtain a proof in $\mathcal{R}_1$.

Conclusion

Theorem

Joinability reduces to confluence while preserving linearity and shallowness restrictions.

Proof.

$(\implies)$ 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.

$(\impliedby)$ 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 $\mathcal{R}_1$). $\square$

References

Guillem Godoy, Ashish Tiwari, and Rakesh Verma. On the confluence of linear shallow term rewrite systems. In *STACS 2003, Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, 2003.

Rakesh Verma. Complexity of normal form properties and reductions for term rewriting problems. *Fundamenta Informaticae*, 2009.

Rakesh Verma. New undecidability results for properties of term rewrite systems. *Electronic Notes in Theoretical Computer Science*, 2012.