Equivalence proofs of simple deterministic grammars

Michael Färber

Université Bordeaux 1, LaBRI

27th May 2013

Scenario

Figure: Tram at Bordeaux. Copyright by Peter Gugerell.

A Matter of Trust

Premise

Many algorithms give only yes/no answers. We trust them usually because they are proven to be correct.

A Matter of Trust

Premise

Many algorithms give only yes/no answers. We trust them usually because they are proven to be correct.

Problems

- An algorithm might be correct, but its implementation might be erroneous.
- The machine executing an algorithm might misbehave.

A Matter of Trust

Premise

Many algorithms give only yes/no answers. We trust them usually because they are proven to be correct.

Problems

- An algorithm might be correct, but its implementation might be erroneous.
- The machine executing an algorithm might misbehave.

Solution: Proofs

An algorithm can certify its response with a proof, which the user can check afterwards.

Our problem

Task

Are two given simple deterministic grammars (sdgs) equivalent?

Existing solution

The equivalence of sdgs is decidable in P, but the algorithm [1] does not produce an equivalence proof.

Question

Can we find an algorithm to construct equivalence proofs of sdgs in time P?

Simple deterministic grammars

Grammar

A simple deterministic grammar is a context-free grammar in Greibach normal form such that for each terminal a, there exists at most one production rule $X \to a\alpha$.

Norm

$\|X\|$ equals the length of the shortest word producible by variable X, or ∞ if no such word exists.

Strictly normed grammars

A grammar is strictly normed iff all its variables have a finite norm.

Equivalence

Two variables are equivalent iff they generate the same language.

Example grammar

Example:

$$X \to a + bC_1$$
$$Y \to a + bC_2$$
$$C_1 \to c$$
$$C_2 \to c$$

$$\dfrac{\dfrac{}{C_1 \equiv c}\,\mathrm{gr} \qquad \dfrac{\dfrac{}{C_2 \equiv c}\,\mathrm{gr}}{c \equiv C_2}\,\mathrm{sym}}{C_1 \equiv C_2}\,\mathrm{trans}$$

Full example proof

$$\dfrac{\dfrac{}{X \equiv a + bC_1}\,\text{gr} \qquad \dfrac{\dfrac{}{Y \equiv a + bC_2}\,\text{gr} \qquad a + bC_2 \equiv a + bC_1}{Y \equiv a + bC_1}\,\text{trans}}{a + bC_1 \equiv Y}\,\text{sym}$$
$$X \equiv Y\ \text{trans}$$

Full example proof

$$\dfrac{\dfrac{\dfrac{}{a \equiv a}\,\text{refl} \qquad \dfrac{\dfrac{}{b \equiv b}\,\text{refl} \qquad C_2 \equiv C_1}{bC_2 \equiv bC_1}\,\times}{a + bC_2 \equiv a + bC_1}\,+ \qquad \dfrac{\dfrac{}{Y \equiv a + bC_2}\,\text{gr} \qquad a + bC_2 \equiv a + bC_1}{Y \equiv a + bC_1}\,\text{trans}}{\dfrac{}{X \equiv a + bC_1}\,\text{gr} \qquad \dfrac{Y \equiv a + bC_1}{a + bC_1 \equiv Y}\,\text{sym}}\,\text{trans}$$

Full example proof

$$\dfrac{\dfrac{\dfrac{}{X \equiv a + bC_1}\,\text{gr} \qquad \dfrac{\dfrac{\dfrac{}{Y \equiv a + bC_2}\,\text{gr} \qquad a + bC_2 \equiv a + bC_1}{Y \equiv a + bC_1}\,\text{trans}}{a + bC_1 \equiv Y}\,\text{sym}}{X \equiv Y}\,\text{trans}$$

$$\dfrac{\dfrac{}{a \equiv a}\,\text{refl} \qquad \dfrac{\dfrac{}{b \equiv b}\,\text{refl} \qquad C_2 \equiv C_1}{bC_2 \equiv bC_1}\,\times}{a + bC_2 \equiv a + bC_1}\,+$$

$$\dfrac{\dfrac{}{C_2 \equiv c}\,\text{gr} \qquad \dfrac{\dfrac{}{C_1 \equiv c}\,\text{gr}}{c \equiv C_1}\,\text{sym}}{C_2 \equiv C_1}\,\text{trans}$$

Recursive grammar

Example:

$$X \to a + bX$$
$$Y \to a + bY.$$

We see that $\mathcal{L}(X) = \mathcal{L}(Y) = b^*a$.

Implicit induction proof

$$\dfrac{\dfrac{}{X \equiv a + bX}\,\text{gr} \qquad \dfrac{\dfrac{\dfrac{}{Y \equiv a + bY}\,\text{gr} \qquad a + bY \equiv a + bX}{Y \equiv a + bX}\,\text{trans}}{a + bX \equiv Y}\,\text{sym}}{X \equiv Y}\,\text{trans}$$

Implicit induction proof

$$\dfrac{\dfrac{}{a \equiv a}\,\text{refl} \qquad \dfrac{\dfrac{}{b \equiv b}\,\text{refl} \qquad \dfrac{\boxed{X \equiv Y}}{Y \equiv X}\,\text{sym}}{bY \equiv bX}\,\times}{a + bY \equiv a + bX}\,+$$

$$\dfrac{\dfrac{}{X \equiv a + bX}\,\text{gr} \qquad \dfrac{\dfrac{}{Y \equiv a + bY}\,\text{gr} \qquad a + bY \equiv a + bX}{Y \equiv a + bX}\,\text{trans}}{a + bX \equiv Y}\,\text{sym}}{X \equiv Y}\,\text{trans}$$

What is not a proof?

Example

$$\dfrac{\dfrac{\boxed{X \equiv Y}}{Y \equiv X}\,\text{sym}}{X \equiv Y}\,\text{sym}$$

Why is that an invalid proof?

On paths between two equivalent rules, there must be at least one product ($\times$) rule.

Proof system

Schemes of non-strict rules:

- $\dfrac{y \equiv x}{x \equiv y}\ \mathrm{sym}$
- $\dfrac{x \equiv y \qquad y \equiv z}{x \equiv z}\ \mathrm{trans}$
- $\dfrac{x \equiv x' \qquad y \equiv y'}{x + y \equiv x' + y'}\ +$

Schemes of strict rules:

- $\dfrac{}{x \equiv x}\ \mathrm{refl}$
- $\dfrac{x \equiv x' \qquad y \equiv y'}{x \cdot y \equiv x' \cdot y'}\ \times$

Strict rules:

- $\dfrac{}{X \equiv \mathrm{Gr}\,(X)}\ \mathrm{gr}$

Figure: Proof rules.

Automatic proof construction

Algorithm

Depth-first search with information which rules have already been constructed, to avoid proving the same judgement multiple times.

Proving one judgement

Given a certain judgement j to prove, we construct a rule with this judgement j as conclusion. This new rule may have up to two premises, which in turn we try to prove recursively.

Obvious construction rules

- $$\frac{A \equiv \mathsf{Gr}\,(A) \qquad \mathsf{Gr}\,(A) \equiv B}{A \equiv B}\ \text{trans (pp)}$$
- $$\frac{a \equiv a \qquad P \equiv Q}{aP \equiv aQ}\ \times\ (\text{ss-}\times)$$
- $$\frac{a\alpha \equiv a\alpha' \qquad \beta \equiv \beta'}{a\alpha + \beta \equiv a\alpha' + \beta'}\ +$$

PP rules

What to do when we encounter products of variables on both sides of a judgement?

Example:

$$AB \equiv CD$$

Easy to treat when $\|A\| = \|C\|$ (application of $\times$ rule), but otherwise we have to rewrite with the trans rule.

Strategy 1: Grammar replacement

In case we encounter a term such as $AB \equiv CD$, we might replace a variable by its production rules; e.g. if $A \to aX + bY$, then we might apply

$$\frac{AB \equiv (aX + bY)\,B \qquad (aX + bY)\,B \equiv CD}{AB \equiv CD}\ \text{trans}$$

Problem

This yields exponential-size proofs even for relatively simple equivalences.

Example:

$$A_0 \to a \qquad A_0' \to a$$

$$A_{n+1} \to aA_nA_n \qquad A_{n+1}' \to aA_n'A_n'$$

Caucal base

For the next strategy, we use the Caucal base, which is a medium also used in the polynomial equivalence algorithm.

Idea

The base stores which variables are prefixes of other variables.
If Y is prefix of X, then $(X, Y\alpha) \in B$, where α is a postfix of X and $X \equiv Y\alpha$.

Caucal base

For the next strategy, we use the Caucal base, which is a medium also used in the polynomial equivalence algorithm.

Idea

The base stores which variables are prefixes of other variables.
If Y is prefix of X, then $(X, Y\alpha) \in B$, where α is a postfix of X and $X \equiv Y\alpha$.

Nice properties

- The size of the base is quadratic.
- Given X and $\|Y\|$, we can calculate α in time P such that $(X, Y\alpha) \in B$ iff Y is prefix of X.

Strategy 2: Base replacement

If we have to prove $AB \equiv CD$ and know that $\|A\| > \|C\|$, then we know that $(A, C\alpha) \in B$. We therefore replace A by $C\alpha$ in our resulting proof:

$$\dfrac{\dfrac{A \equiv C\alpha \qquad B \equiv B}{AB \equiv C\alpha B} \times \qquad \dfrac{C \equiv C \qquad \alpha B \equiv D}{C\alpha B \equiv CD} \times}{AB \equiv CD}\ \text{trans}$$

Examples:

$$A_0 \to a \qquad A_0' \to a$$
$$A_{n+1} \to aA_nA_n \qquad A_{n+1}' \to aA_n'A_n'$$

$$W \to aB \qquad Y \to a \qquad B \to b$$
$$X \to c \qquad Z \to bX$$

Example proof with BR

$$\dfrac{\dfrac{}{Y \equiv Y}\,\text{refl} \qquad \dfrac{\dfrac{}{B \equiv b}\,\text{gr} \qquad \dfrac{}{X \equiv X}\,\text{refl}}{BX \equiv bX}\,\times \qquad \dfrac{}{bX \equiv Z}\,\text{gr}}{\dfrac{BX \equiv Z}{YBX \equiv YZ}\,\text{trans}}\,\times$$

$$\dfrac{\dfrac{}{W \equiv aB}\,\text{gr} \qquad \dfrac{\dfrac{}{a \equiv Y}\,\text{gr} \qquad \dfrac{}{B \equiv B}\,\text{refl}}{aB \equiv YB}\,\times}{\dfrac{W \equiv YB}{WX \equiv YBX}\,\text{trans}} \qquad \dfrac{}{X \equiv X}\,\text{refl}\,\times$$

$$\dfrac{WX \equiv YBX \qquad YBX \equiv YZ}{WX \equiv YZ}\,\text{trans}$$

Base replacement problems

Base replacement produces proofs of exponential size; e.g. for proving $F^n \equiv G^n$ of following grammar:

$$
\begin{array}{rlrl}
A & \to a & A & \to a \\
B & \to b & B & \to b \\
A_B^0 & \to aB & B_A^0 & \to bA \\
A_B^{n+1} & \to aBA_B^n A_B^n & B_A^{n+1} & \to bAB_A^n B_A^n \\
F^n & \to aBA_B^n & G^n & \to aB_A^n B
\end{array}
$$

Decomposition

For the next strategy, we invented a new operation, which we called "Decomposition".

Idea

The decomposition calculates the prefix of a variable possessing a certain norm.
Given that α is a prefix of X and there exists a prefix β of Y such that $X \equiv \alpha\beta$, we can calculate β from $\|X\|$, $\|\alpha\|$ and Y in time P.

Limited domain

Unlike the base, decomposition is not always guaranteed to return successfully, as there does not always exist a polynomial-size prefix β of Y! In this case, we fall back to base replacement. However, decomposition always works if there are no sums in the grammar.

Strategy 3: Decomposition

If we have to prove $AB \equiv CD$ and know that $\|A\| > \|C\|$, then $\alpha = C$, and we try to calculate β from $\|A\|$, $\|C\|$ and D. If this calculation is successful, we replace A by $C\beta$ in our resulting proof:

$$\dfrac{\dfrac{A \equiv C\beta \qquad B \equiv B}{AB \equiv C\beta B} \times \qquad \dfrac{C \equiv C \qquad \beta B \equiv D}{C\beta B \equiv CD} \times}{AB \equiv CD}\ \text{trans}$$

This strategy succeeds in finding a polynomial-size proof for the counter-example of the base replacement strategy, but ...

It happened last week ...

It happened last week ...

Figure: An image sometimes says more than a thousand words ...

Decomposition problems

Decomposition produces proofs of exponential size; e.g. for proving $F^n \equiv G^n$ of following grammar:

$$
\begin{array}{rlrl}
A & \to a + b & A & \to a + b \\
B & \to b + a & B & \to b + a \\
A^0_B & \to (a + b)\, B & B^0_A & \to (b + a)\, A \\
A^{n+1}_B & \to (a + b)\, B A^n_B A^n_B & B^{n+1}_A & \to (b + a)\, A B^n_A B^n_A \\
F^n & \to (a + b)\, B A^n_B & G^n & \to (a + b)\, B^n_A B
\end{array}
$$

This is because the grammar of each variable is a sum, which is why decomposition always fails, and we have to fall back to base replacement, which we already claimed to produce exponential-size proofs for a similar grammar before.

Then what is decomposition good for, anyway?

So far, using the decomposition we did not find a single exponential-size proof when we looked at grammars without sums.

Further problem

It is not even clear whether the given proof system allows polynomial-size proofs for all possible kinds of grammars.

Outlook

- Find new strategy to prevent exponential-size proofs
- Adapt given proof system
- Find new proof system: A system more closely related to the existing polynomial algorithm would allow polynomial-size proofs, but proofs in it might not be as simple to verify. Possible candidate: $\lambda\Pi$ (first order dependent type theory)

Thank you for your attention! Questions?

Bibliography


Yoram Hirshfeld, Mark Jerrum, and Faron Moller.
A polynomial algorithm for deciding bisimilarity of normed context-free processes.
Theoretical Computer Science, 158:143–159, 1996.