Equivalence proofs of simple deterministic grammars

Michael Färber

Université Bordeaux 1, LaBRI

27th May 2013
Scenario

Figure: Tram at Bordeaux. Copyright by Peter Gugerell.
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 $X \rightarrow a\alpha$.

Norm
$\|X\|$ equals the length of the shortest word producible by variable $X$, or $\infty$ 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:

\[ X \rightarrow a + bC_1 \]
\[ Y \rightarrow a + bC_2 \]
\[ C_1 \rightarrow c \]
\[ C_2 \rightarrow c \]

\[ C_1 \equiv c \quad \text{gr} \]
\[ C_2 \equiv c \quad \text{gr} \]
\[ c \equiv C_2 \quad \text{sym} \]
\[ C_1 \equiv C_2 \quad \text{trans} \]
Full example proof

\[
\begin{align*}
\text{gr} & \quad Y \equiv a + bC_2 \\
\quad & \quad a + bC_2 \equiv a + bC_1 & \text{trans} \\
\text{gr} & \quad X \equiv a + bC_1 \\
\quad & \quad a + bC_1 \equiv Y & \text{sym} \\
\quad & \quad Y \equiv a + bC_1 & \text{trans} \\
\quad & \quad a + bC_1 \equiv Y & \text{trans} \\
\quad & \quad X \equiv Y
\end{align*}
\]
Full example proof

\[
\begin{align*}
    a \equiv a \quad \text{refl} & \quad b \equiv b \quad \text{refl} & \quad C_2 \equiv C_1 \\
    bC_2 \equiv bC_1 & \quad \times & \quad a + bC_2 \equiv a + bC_1 \\
    Y \equiv a + bC_2 \quad \text{gr} & \quad a + bC_2 \equiv a + bC_1 \quad \text{trans} \\
    X \equiv a + bC_1 \quad \text{gr} & \quad Y \equiv a + bC_1 \quad \text{sym} \\
    a + bC_1 \equiv Y & \quad \text{trans} \\
    X \equiv Y
\end{align*}
\]
Full example proof

\[ \begin{align*}
C_2 & \equiv c & C_2 & \equiv C_1 \\
& \quad \text{gr} & \quad \text{sym} & \quad \text{trans}
\end{align*} \]

\[ \begin{align*}
C_1 & \equiv c & c & \equiv C_1 \\
& \quad \text{trans}
\end{align*} \]

\[ \begin{align*}
C_2 & \equiv C_1 \\
& \quad \text{refl}
\end{align*} \]

\[ \begin{align*}
b & \equiv b & bC_2 & \equiv bC_1 \\
& \quad \text{refl} & \quad \text{trans}
\end{align*} \]

\[ \begin{align*}
a + bC_2 & \equiv a + bC_1 \\
& \quad +
\end{align*} \]

\[ \begin{align*}
Y & \equiv a + bC_2 & a + bC_2 & \equiv a + bC_1 \\
& \quad \text{gr} & \quad \text{trans}
\end{align*} \]

\[ \begin{align*}
X & \equiv a + bC_1 & Y & \equiv a + bC_1 \\
& \quad \text{gr} & \quad \text{sym} & \quad \text{trans}
\end{align*} \]

\[ \begin{align*}
X & \equiv Y
\end{align*} \]
Recursive grammar

Example:

\[ X \rightarrow a + bX \]
\[ Y \rightarrow a + bY. \]

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

\[
\begin{align*}
Y &\equiv a + bY \quad \text{gr} \\
\frac{a + bY \equiv a + bX}{Y \equiv a + bX} \quad \text{trans} \\
\frac{X \equiv a + bX \quad \text{gr}}{a + bX \equiv Y} \quad \text{sym} \\
\frac{a + bX \equiv Y}{X \equiv Y} \quad \text{trans}
\end{align*}
\]
Implicit induction proof

\[
\begin{align*}
X &\equiv Y \\
\text{sym} \\
\hline
b &\equiv b \\
\text{refl} \\
Y &\equiv X \\
\times \\
\hline
a &\equiv a \\
\text{refl} \\
bY &\equiv bX \\
\hline
a + bY &\equiv a + bX \\
+ \\
\hline
Y &\equiv a + bY \\
\text{gr} \\
\hline
a + bY &\equiv a + bX \\
\text{trans} \\
\hline
X &\equiv a + bX \\
\text{gr} \\
\hline
Y &\equiv a + bX \\
\text{sym} \\
\hline
a + bX &\equiv Y \\
\text{sym} \\
\hline
X &\equiv Y \\
\text{trans}
\end{align*}
\]
What is not a proof?

Example

\[
\begin{align*}
X & \equiv Y \\
\sym & \\
Y & \equiv X \\
\sym & \\
X & \equiv Y
\end{align*}
\]

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:

\[
\begin{align*}
\text{sym} & : \quad y \equiv x \\
& \quad \quad \quad \Rightarrow \quad x \equiv y \\
\text{trans} & : \quad x \equiv y, y \equiv z \\
& \quad \quad \quad \Rightarrow \quad x \equiv z \\
\text{+} & : \quad x \equiv x', y \equiv y' \\
& \quad \quad \quad \Rightarrow \quad x + y \equiv x' + y'
\end{align*}
\]

Schemes of strict rules:

\[
\begin{align*}
\text{refl} & : \quad x \equiv x \\
\text{×} & : \quad x \equiv x', y \equiv y' \\
& \quad \quad \quad \Rightarrow \quad x \cdot y \equiv x' \cdot y' \\
\text{gr} & : \quad X \equiv \text{Gr}(X)
\end{align*}
\]

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

\[
\begin{align*}
A \equiv \text{Gr} (A) & \quad \text{Gr} (A) \equiv B \quad \text{trans (pp)} \\
\frac{A \equiv B}{a \equiv a} & \quad P \equiv Q \quad \times (ss-\times) \\
\frac{aP \equiv aQ}{a \alpha \equiv a \alpha'} & \quad \beta \equiv \beta' \\
\frac{a \alpha + \beta \equiv a \alpha' + \beta'}{+}
\end{align*}
\]
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 $A \rightarrow aX + bY$, then we might apply

$$AB \equiv (aX + bY) B \quad (aX + bY) B \equiv CD \quad \text{trans}$$

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

Example:

$$A_0 \rightarrow a$$
$$A_{n+1} \rightarrow aA_nA_n$$
$$A'_0 \rightarrow a$$
$$A'_{n+1} \rightarrow aA'_nA'_n$$
Cauca base

For the next strategy, we use the Cauca 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 $\alpha$ 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 $\alpha$ 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 $\alpha$ 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:

$$\frac{A \equiv C\alpha \quad B \equiv B}{AB \equiv C\alpha B} \times \frac{C \equiv C \quad \alpha B \equiv D}{C\alpha B \equiv CD} \times \frac{AB \equiv CD}{\text{trans}}$$

Examples:

$$A_0 \rightarrow a \quad A'_0 \rightarrow a$$
$$A_{n+1} \rightarrow aA_nA_n \quad A'_{n+1} \rightarrow aA'_nA'_n$$

$$W \rightarrow aB \quad Y \rightarrow a \quad B \rightarrow b$$
$$X \rightarrow c \quad Z \rightarrow bX$$
Example proof with BR

\[
\begin{align*}
\text{refl} & \quad B \equiv b \\
\text{gr} & \quad X \equiv X \\
\text{refl} & \quad BX \equiv bX \\
\times & \quad B \equiv bX \\
\text{gr} & \quad bX \equiv Z \\
\text{trans} & \quad BX \equiv Z \\
\times & \quad YBX \equiv YZ
\end{align*}
\]

\[
\begin{align*}
\text{gr} & \quad W \equiv aB \\
\text{gr} & \quad a \equiv Y \\
\text{gr} & \quad B \equiv B \\
\times & \quad aB \equiv YB \\
\text{trans} & \quad W \equiv YB \\
\text{trans} & \quad WX \equiv YBX \\
\times & \quad X \equiv X \\
\times & \quad W \equiv YB \\
\times & \quad WX \equiv YBX \\
\times & \quad YBX \equiv YZ \\
\text{trans} & \quad WX \equiv YZ
\end{align*}
\]
Base replacement problems

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

\[
\begin{align*}
A & \rightarrow a & A & \rightarrow a \\
B & \rightarrow b & B & \rightarrow b \\
A_B^0 & \rightarrow aB & B_A^0 & \rightarrow bA \\
A_B^{n+1} & \rightarrow aBA_B^n A_B^n & B_A^{n+1} & \rightarrow bAB_A^n B_A^n \\
F^n & \rightarrow aBA_B^n & G^n & \rightarrow aB_A^n B
\end{align*}
\]
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 $\alpha$ is a prefix of $X$ and there exists a prefix $\beta$ of $Y$ such that $X \equiv \alpha \beta$, we can calculate $\beta$ 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 $\beta$ 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 $\beta$ from $\|A\|$, $\|C\|$ and $D$. If this calculation is successful, we replace $A$ by $C\beta$ in our resulting proof:

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

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:

\[
\begin{align*}
A & \rightarrow a + b & A & \rightarrow a + b \\
B & \rightarrow b + a & B & \rightarrow b + a \\
A_B^0 & \rightarrow (a + b) B & B_A^0 & \rightarrow (b + a) A \\
A_B^{n+1} & \rightarrow (a + b) BA_B^n A_B^n & B_A^{n+1} & \rightarrow (b + a) AB_A^n B_A^n \\
F^n & \rightarrow (a + b) BA_B^n & G^n & \rightarrow (a + b) B_A^n B
\end{align*}
\]

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?