Algebraic Proof Mining for Fast Decision Procedures

Aaron Stump

Computer Science and Engineering
Washington University
St. Louis, Missouri, USA

CSE Retreat ’05
CAREER: Semantic Programming

Develop language-based approaches to program verification.

Programmer knows why code is right.

Provide language constructs to encode that knowledge formally.

“Programming with proofs”:
  Code intertwined with proofs.
  Functions input proofs of pre-conditions.
  They output proofs of post-conditions.
  Proofs built by hand, with automated help.

Goal: make provably correct coding a practical reality!
Case Study: Decision Procedures

*Decision procedures* (DPs) check validity of logical formulas.

Can handle background theories: arithmetic, arrays, bitvectors.

Used for algorithmic verification.

Good case study for programming with proofs:
- Relations between proofs and DPs well understood.
- Proofs independently valuable.
- Proofs are about data (formulas), not executions.
- Proofs can be enormous, so good engineering required.

Leverages PI expertise in the area.
Decision procedures are hot in verification.

Increasing submissions at CAV and TACAS.

Inside a Decision Procedure

SAT solver: ANDs, ORs, and NOTs.

Theory solver: $3x + y < z$, \textit{write}(a, i, v) = b$, or just $a = b$.

SAT solver satisfies boolean structure, then calls theory solver.
Inside a Decision Procedure

SAT solver: ANDs, ORs, and NOTs.

Theory solver: $3x + y < z$, $\text{write}(a, i, v) = b$, or just $a = b$.

SAT solver satisfies boolean structure, then calls theory solver.
Inside a Decision Procedure

SAT solver: ANDs, ORs, and NOTs.

Theory solver: $3x + y < z$, $\text{write}(a, i, v) = b$, or just $a = b$.

SAT solver satisfies boolean structure, then calls theory solver.

SAT solver

Theory Reasoner

$a = b$
$d = b$
$b = c$
$a \neq c$
Inside a Decision Procedure

SAT solver: ANDs, ORs, and NOTs.

Theory solver: $3x + y < z$, $\text{write}(a, i, v) = b$, or just $a = b$.

SAT solver satisfies boolean structure, then calls theory solver.
Inside a Decision Procedure

SAT solver: ANDs, ORs, and NOTs.

Theory solver: $3x + y < z$, $\text{write}(a, i, v) = b$, or just $a = b$.

SAT solver satisfies boolean structure, then calls theory solver.

Conflict clause:

\[ a \neq b \lor b \neq c \lor a = c \]
Conflict Clauses

Conflict clauses prune later search.

Smaller clauses aren’t always better [Malik et al. 2001].

A subset of a clause is always better.

Suppose theory solver produces proof of contradiction.

Just return the assumptions used in that proof.

Assumptions: \( a = b, \quad d = b, \quad b = c, \quad a \neq c. \)

Proof:

\[
\begin{align*}
  a = b & \quad b = c \\
  \text{Trans} & \\
  a = c & \quad a \neq c
\end{align*}
\]

\[
\text{Contra}
\]

\[
\text{false}
\]
Proof Mining [Barrett,Dill,Stump 2002]

Suppose theory solver produces proof of contradiction.

Just return the assumptions used in that proof.

Assumptions: \( a = b, \ d = b, \ b = c, \ a \neq c \).

Proof:

\[
\frac{a = b \quad b = c}{\text{Trans} \quad a = c} \quad \frac{a \neq c}{\text{Contra} \quad \text{false}}
\]
Suppose theory solver produces proof of contradiction. Just return the assumptions used in that proof.

Assumptions: \(a = b, \ d = b, \ b = c, \ a \neq c\).

Proof:

\[
\begin{align*}
\begin{array}{cccc}
  & a = b & b = c & \\
\hline
  a & = c & & \\
\end{array}
\end{align*}
\]

\(a = c\) \hspace{1cm} \text{Trans} \hspace{1cm} a \neq c \hspace{1cm} \text{Contra}

false
Flabby Proofs from Union-Find

Assumptions: $a = b$, $a = c$, $a \neq c$.

Union-find structure: Action:
Flabby Proofs from Union-Find

Assumptions: \( a = b, \ a = c, \ a \neq c. \)

Union-find structure: Action: union(a,b)

\[
\begin{array}{c}
 b \\
 \uparrow \\
 a
\end{array}
\]
Flabby Proofs from Union-Find

Assumptions: \( a = b, \ a = c, \ a \neq c. \)

Union-find structure: Action: union(a,c)

```
    b
   / \
a   /  \
c
```
Flabby Proofs from Union-Find

Assumptions: \( a = b, \ a = c, \ a \neq c. \)

Union-find structure: \( \text{find}(a) = b, \ \text{find}(c) = b \)

\[
\begin{array}{c}
b \\
\uparrow & \downarrow \\
a & c
\end{array}
\]
Flabby Proofs from Union-Find

Assumptions: $a = b$, $a = c$, $a \neq c$.

Union-find structure: \[ \text{find}(a) = b, \text{find}(c) = b \]
\[ \begin{array}{c}
  \text{b} \\
  \text{a} \quad \text{c}
\end{array} \]

Resulting flabby proof:

\[
\begin{align*}
  a &= b \\
  b &= a & \text{Symm} \\
  a &= c & \text{Trans} \\
  b &= c & \text{Trans} \\
  a &= c & \text{Trans} \\
  a &= c & \text{Contra}
\end{align*}
\]
Flabby Proofs from Union-Find

Assumptions: \( a = b \), \( a = c \), \( a \neq c \).

Union-find structure: \( \text{find}(a) = b \), \( \text{find}(c) = b \)

\[
\begin{array}{cc}
\text{b} & \\
\uparrow & \downarrow \\
a & c
\end{array}
\]

Resulting flabby proof:

\[
\begin{align*}
  a &= b \\
  b &= a & \text{Symm} \\
  a &= c & \text{Trans} \\
  b &= c & \text{Trans} \\
  a &= c & \text{Trans} \\
  a \neq c & \text{Contra}
\end{align*}
\]
Idea: transform proofs to get rid of flab.

*Algebraic*: transformations based on equations between proofs.

\[
\begin{align*}
\mathcal{D}_1 \quad & \quad \mathcal{D}_2 \\
& \quad \frac{a = b \quad b = c}{a = c} \quad \text{Trans} \\
& \quad \frac{a = c}{a = d} \\
\mathcal{D}_3 \\
& \quad \frac{c = d}{a = d} \\
\mathcal{D}_2 \quad & \quad \mathcal{D}_3 \\
& \quad \frac{b = c \quad c = d}{a = d} \quad \text{Trans} \\
\end{align*}
\]

\[
\begin{align*}
\mathcal{D} \quad & \quad \mathcal{D}_1 \\
& \quad \frac{a = a}{a = b} \quad \text{Refl} \\
& \quad \frac{a = b}{a = b} \quad \text{Trans} \\
\mathcal{D} \\
& \quad \frac{a = b}{b = a} \quad \text{Symm} \\
& \quad \frac{b = a}{b = b} \quad \text{Trans} \\
\end{align*}
\]

\[
\begin{align*}
\mathcal{D} \quad & \quad \mathcal{D}_1 \\
& \quad \frac{a = b}{b = b} \quad \text{Trans} \\
& \quad \frac{b = b}{b = b} \quad \text{Refl} \\
\end{align*}
\]
An Equational Theory On Proof Terms

Equations

\begin{align*}
\text{Trans(Trans}(d_1,d_2),d_3) & \equiv \text{Trans}(d_1,\text{Trans}(d_2,d_3)) \\
\text{Trans(Refl,d)} & \equiv d \\
\text{Trans(Symm}(d),d) & \equiv \text{Refl}
\end{align*}

Which equational theory is it?
Compare these equations:

\[
\begin{align*}
\text{Trans(Trans(d₁,d₂),d₃)} & \mathbin{\cong} \text{Trans(d₁,Trans(d₂,d₃))} \\
\text{Trans(Refl,d)} & \mathbin{\cong} d \\
\text{Trans(Symm(d),d)} & \mathbin{\cong} \text{Refl}
\end{align*}
\]

With the group axioms:

\[
\begin{align*}
(d₁ \ast d₂) \ast d₃ & \mathbin{\cong} d₁ \ast (d₂ \ast d₃) \\
1 \ast d & \mathbin{\cong} d \\
d^{-1} \ast d & \mathbin{\cong} 1
\end{align*}
\]

Trans is \(\ast\), Symm is \((\cdot)^{-1}\), and Refl is 1.
Transforming Equality Proofs

Theorem (Knuth-Bendix, 1970)

*These rules put every group term into canonical form:*

1. \((x \star y) \star z \rightarrow x \star (y \star z)\)
2. \(x^{-1} \star x \rightarrow 1\)
3. \(x \star x^{-1} \rightarrow 1\)
4. \(x \star (x^{-1} \star y) \rightarrow y\)
5. \(x^{-1} \star (x \star y) \rightarrow y\)
6. \((x \star y)^{-1} \rightarrow y^{-1} \star x^{-1}\)
7. \(1 \star x \rightarrow x\)
8. \(x \star 1 \rightarrow x\)
9. \(1^{-1} \rightarrow 1\)
10. \((x^{-1})^{-1} \rightarrow x\)

Rewrite proofs to remove flab.
Better: mine assumptions without actually rewriting.
# Empirical Results in CVC

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>dec. orig</th>
<th>time orig (s)</th>
<th>dec. mining</th>
<th>time mining (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dlx-regfile</td>
<td>2807</td>
<td>2.1</td>
<td>2430</td>
<td>2.5</td>
</tr>
<tr>
<td>dlx-dmem</td>
<td>1336</td>
<td>1.0</td>
<td>1048</td>
<td>0.9</td>
</tr>
<tr>
<td>pp-regfile</td>
<td>115197</td>
<td>295.7</td>
<td>44547</td>
<td>121.9</td>
</tr>
<tr>
<td>pp-dmem</td>
<td>25928</td>
<td>68.1</td>
<td>11899</td>
<td>23.7</td>
</tr>
<tr>
<td>pp-bloaddata</td>
<td>4060</td>
<td>1.7</td>
<td>3461</td>
<td>2.1</td>
</tr>
<tr>
<td>pp-TakenBranch</td>
<td>15364</td>
<td>26.1</td>
<td>11928</td>
<td>24.6</td>
</tr>
</tbody>
</table>
Conclusion

Decision procedures: case study for programming with proofs.

Algebraic proof mining: mine info from transformed proofs.

For equality proofs, use rewrite rules for free group theory.

2x performance improvement on large benchmarks observed.

Future work: apply to other theories.

Future work: RVC decision procedure, written in RSP.
Congruence Rules

Congruence rules are commuting endomorphisms.

These proofs prove the same theorem:

\[
\begin{align*}
&\frac{a = b}{a = c} \quad \text{Trans} \quad \frac{b = c}{a = c} \\
&\frac{f(a, d) = f(c, d)}{f(a, d) = f(c, d)} \quad \text{Cong}_{f,1}
\end{align*}
\]

\[
\begin{align*}
&\frac{a = b}{f(a, d) = f(b, d)} \quad \text{Cong}_{f,1} \quad \frac{b = c}{f(b, d) = f(c, d)} \quad \text{Cong}_{f,1} \\
&\frac{f(a, d) = f(c, d)}{f(a, d) = f(c, d)} \quad \text{Trans}
\end{align*}
\]

Commutativity is also required:

\[
\text{Trans}(\text{Cong}_{f,1}(d1), \text{Cong}_{f,2}(d2)) \cong \text{Trans}(\text{Cong}_{f,2}(d2), \text{Cong}_{f,1}(d1))
\]
Rules for Commuting Endomorphisms

1. \((x \ast y) \ast z\) \quad \rightarrow \quad x \ast (y \ast z)
2. \(x^{-1} \ast x\) \quad \rightarrow \quad 1
3. \(x \ast x^{-1}\) \quad \rightarrow \quad 1
4. \(x \ast (x^{-1} \ast y)\) \quad \rightarrow \quad y
5. \(x^{-1} \ast (x \ast y)\) \quad \rightarrow \quad y
6. \((x \ast y)^{-1}\) \quad \rightarrow \quad y^{-1} \ast x^{-1}
7. \(1 \ast x\) \quad \rightarrow \quad x
8. \(x \ast 1\) \quad \rightarrow \quad x
9. \(1^{-1}\) \quad \rightarrow \quad 1
10. \((x^{-1})^{-1}\) \quad \rightarrow \quad x
11. \(f(1)\) \quad \rightarrow \quad 1
12. \((f(x))^{-1}\) \quad \rightarrow \quad f(x^{-1})
13. \(f(x) \ast f(y)\) \quad \rightarrow \quad f(x \ast y)
14. \(f(x) \ast (f(y) \ast z)\) \quad \rightarrow \quad f(x \ast y) \ast z
15. \(g(1)\) \quad \rightarrow \quad 1
16. \((g(x))^{-1}\) \quad \rightarrow \quad g(x^{-1})
17. \(g(x) \ast g(y)\) \quad \rightarrow \quad g(x \ast y)
18. \(g(x) \ast (g(y) \ast z)\) \quad \rightarrow \quad g(x \ast y) \ast z
19. \(f(x) \ast g(y)\) \quad \rightarrow \quad g(y) \ast f(x)
20. \(f(x) \ast (g(y) \ast z)\) \quad \rightarrow \quad g(y) \ast (f(x) \ast z)