Knuth-Bendix Completion with Modern Termination Checking

Ian Wehrman
Thesis Defense
July 26, 2006

Equational Automated Theorem Proving

- Want to solve the **word problem** automatically.
- Does a finite set of identities (a **theory**) entail another identity?

Example Theory: Groups

- For example, the theory of **groups** (G) is axiomatized by three identities:

$$x * 1 \approx x \quad x * x^{-1} \approx 1 \quad x * (y * z) \approx (x * y) * z$$

Word Problem for Groups

- The **word problem** for G: is an identity a consequence of the axioms of group theory?
- E.g., a left-inverse lemma:

$$\boxed{G \models x^{-1} * x \overset{?}{\approx} 1}$$

Proof about Groups

- Yes, there is a left inverse lemma! Here's the proof:

$$
\begin{array}{rcll}
x^{-1} * x & \approx & x^{-1} * (x * 1) & (1) \\
 & \approx & x^{-1} * (x * (x^{-1} * (x^{-1})^{-1})) & (1) \\
 & \approx & x^{-1} * ((x * x^{-1}) * (x^{-1})^{-1}) & (3) \\
 & \approx & x^{-1} * (1 * (x^{-1})^{-1}) & (2) \\
 & \approx & (x^{-1} * 1) * (x^{-1})^{-1} & (3) \\
 & \approx & x^{-1} * (x^{-1})^{-1} & (1) \\
 & \approx & 1 & (2)
\end{array}
$$

Automating Group Theory Proofs

- That proof looked a little tricky.
  - Q) How long did it take me to find it?
  - A) About 0.2s – I used an automated theorem prover! (Much longer with just my head.)

Group Theory Completion

- Used a tool called **Waldmeister** that implements an algorithm called **completion**.
  - Input: theory (finite set of identities).
  - Output: rewriting system (also called a **completion**) used to decide whether or not an identity holds.

Group Theory Completion

$$1 * x \approx x \quad x^{-1} * x \approx 1 \quad (x * y) * z \approx x * (y * z)$$

- Input: G
- Output: rewriting system equivalent to G.
- To prove an identity holds, rewrite both sides, then test for equality.

$$\begin{array}{lll}
1 * x \to x & x * 1 \to x & 1^{-1} \to 1 \\
(x^{-1})^{-1} \to x & (x * y)^{-1} \to x^{-1} * y^{-1} & (x * y) * z \to x * (y * z) \\
x * x^{-1} \to 1 & x^{-1} * x \to 1 & \\
x * (x^{-1} * y) \to y & x^{-1} * (x * y) \to y &
\end{array}$$

Group Theory Proofs Made Easy

- With a completion, it's easy to solve the word problem. Works every time.

$$
\begin{array}{lcl}
(y * x) * (x * y)^{-1} & \rightarrow & (y * x) * (x^{-1} * y^{1}) \\
 & \rightarrow & y * (x * (x^{-1} * y^{-1})) \\
 & \rightarrow & y * y^{-1} \\
 & \rightarrow & 1 \\
(y * x)^{-1} * (x * y) & \rightarrow & (y^{-1} * x^{-1}) * (x * y) \\
 & \rightarrow & y^{-1} * (x^{-1} * (x * y)) \\
 & \rightarrow & y^{-1} * y \\
 & \rightarrow & 1
\end{array}
$$

Another Completion

$$1 * x \approx x \qquad (x * y) * z \approx x * (y * z)$$
$$x^{-1} * x \approx 1 \qquad h(x * y) \approx h(x) * h(y)$$

- Input: groups + one endomorphism (GE_1).
- Output: completion for GE_1. Use this to solve the word problem for GE_1. Easy!

$$\begin{array}{ll}
x * 1 \to x & x * (y * z) \to (x * y) * z \\
1 * x \to x & (x * y)^{-1} \to x^{-1} * y^{-1} \\
x * x^{-1} \to 1 & (x * y) * y^{-1} \to x \\
x^{-1} * x \to 1 & (x * y^{-1}) * y \to x \\
1^{-1} \to 1 & h(x)^{-1} \to h(x^{-1}) \\
h(1) \to 1 & h(x) * h(y) \to h(x * y) \\
(x^{-1})^{-1} \to x & (x * h(y)) * h(z) \to x * h(y * z)
\end{array}$$

Completion Fails!

$$
\begin{array}{lll}
1 * x \approx x & x^{-1} * x \approx 1 & (x * y) * z \approx x * (y * z) \\
f(x * y) \approx f(x) * f(y) & g(x * y) \approx g(x) * g(y) & f(x) * g(y) \approx g(y) * f(x)
\end{array}
$$

- Input: theory of groups + two commuting endomorphisms (CGE_2).
- Output: ... **not a completion!**
- Without a completion, we must use our heads to prove identities hold in CGE_2.

Our Mission

- **Revise** the algorithm used by Waldmeister.
- Use it to **find a completion** for CGE_2.
- Solve the word problem for CGE_2 **without using our heads**.

But first...

- Waldmeister's algorithm relies on results in the exciting field of **term rewriting**.
- Today's agenda:
  - Cover important details about the word problem and term rewriting.
  - Describe **completion** (Waldmeister's algorithm).
  - See why completion fails and then **fix it**.

All About the Word Problem

$$u_1 \approx v_1, u_2 \approx v_2, \ldots, u_n \approx v_n \models t_1 \approx t_n$$

- It's **undecidable** (in general).
- Can decide the word problem for some theories, but not all.

Word Problem Proofs

- How do we know an identity **holds** in a theory? Find a proof.
- Proof is a sequence of terms: starting with one side of the identity and ending with the other side.
- Successive terms created by replacing instances of one side of the theory axioms with instances of the other.
- Easy to check, but hard to find.

Solving the Word Problem by Rewriting

- Idea: **orient** axioms – now called **rules**.
- Replace instances of lhs with instances of rhs – called **rewriting**.
- Rewrite terms to **normal form.**
- Two sides of identity have same normal form iff identity holds.

Rewriting to Normal Form

- To solve the word problem like this, normal forms must:
  - require finitely many reductions,
  - be unique – same end result regardless of reduction sequence.

Properties of Rewriting Systems

- Corresponds to the two most important properties of rewriting systems:
  - **Termination**: no infinitely long reduction sequences.
  - **Confluence**: if a term is rewritten to distinct terms, then those terms can be rewritten to a common term (**joined**).
- Termination + confluence = **convergence**.

Rewriting Example 1

- The non-confluent, terminating system

$$f(x,y) \to x \quad g(x) \to x \quad f(x,x) \to h(x)$$

applied to term *f(x,g(x))* yields any of these reduction sequences:

$$\begin{array}{ll} 1. & f(x,g(x)) \to x \\ 2. & f(x,g(x)) \to f(x,x) \to h(x) \end{array}$$

Rewriting Example 2

- The confluent, nonterminating system

$$f(x) \to g(h(x)) \quad g(x) \to f(x)$$

applied to term *f(x)* yields this looping reduction sequence:

$$\begin{aligned}
&f(x) \to g(h(x)) \to \\
&f(h(x)) \to g(h(h(x))) \to \\
&f(h(h(x))) \to g(h(h(h(x)))) \to \\
&f(h(h(h(x)))) \to g(h(h(h(h(x))))) \to \cdots
\end{aligned}$$

Rewriting Example 3

- The convergent system

$$
\begin{aligned}
&ack(0,n) \to n+1\\
&ack(m+1,0) \to ack(m,1)\\
&ack(m+1,n+1) \to ack(m,ack(m+1,n))
\end{aligned}
$$

applied to term *ack(3,3)* yields this long reduction sequence:

$$
\begin{aligned}
&ack(3,3) \to ack(2,ack(3,2)) \to ack(2,(ack(2,(ack(3,1))))) \to\\
&ack(2,(ack(2,(ack(2,ack(3,0)))))) \to ack(2,(ack(2,(ack(2,ack(2,1)))))) \to\\
&ack(2,(ack(2,(ack(2,ack(1,ack(2,0))))))) \to \cdots \to 61
\end{aligned}
$$

Proving Rewriting Properties

- To solve the word problem with rewriting, systems must be terminating and confluent.
  - How do we prove these properties?
  - What if we can't?

Proving Termination

- Prove a system is terminating with special well-founded ordering relation: a **reduction order**.
- Theorem: a system is terminating iff a compatible reduction order exists.
- An order $>$ is **compatible** with a rewriting system if $l > r$ for all rules $l \to r$.

Proving Termination

- Termination is undecidable (reduction from halting problem), so finding a compatible ordering is tough.
- Could also be impossible – e.g., any theory with the identity $x + y \approx y + x$ is not compatible with any reduction order.

Automated Termination Checkers

- Interesting aside: there are nifty tools to **automatically** prove termination.
- Works for systems that are compatible with any one of a variety of reduction orders.
- E.g., **AProVE**: fast, effective and produces human-readable proofs.
- Could be useful later...?

Proving Confluence

- Confluence is undecidable in general,
- But decidable for rewriting systems that are terminating.

Deciding Confluence for Terminating Systems

- Try to rewrite a common instance of two rules' lhs to different terms: $t_2 \leftarrow s_1 \rightarrow t_1$.
- Try to join those terms to a common term: $t_1 \rightarrow s_2 \leftarrow t_2$.
- (t_1, t_2) called a **critical pair**.
- Theorem: joinability of all critical pairs implies confluence **for terminating systems**.

Critical Pair Example 1

$$f(x, g(x)) \to x \qquad g(g(x)) \to x$$

- Common instances of rules' lhs rewrites two ways:

$$g(x) \leftarrow f(g(x), g(g(x))) \to f(g(x), x)$$

Non-Confluent Systems

- If system is not confluent, sometimes we can find an **equivalent** system that is.
- Systems are equivalent if an identity holds in one system iff it holds in the other.

Creating Confluent Systems

- Start with a terminating system, compatible with reduction order >.
- Calculate a non-joinable critical pair (t_1, t_2)
- If $t_1 > t_2$, then **add rule** $t_1 \rightarrow t_2$ to system.
- Continue until all critical pairs are joinable.

Critical Pair Example 2

$$f(x, g(x)) \to x \qquad g(g(x)) \to x$$

$$g(x) \leftarrow f(g(x), g(g(x))) \to f(g(x), x)$$

- Add unjoinable critical pair as rewrite rule. New, equivalent system:

$$f(x, g(x)) \to x \quad g(g(x)) \to x \quad f(g(x), x) \to g(x)$$

Completion

- Called **completion**, invented by Knuth.
- Completion can **solve the word problem**.
  - Use the equivalent, covergent rewrite system (the **completion**) to normalize both sides of any identity.
  - If normal forms are the same, identity holds, otherwise it doesn't.

Limits of Completion

- Completion doesn't always work:
  - An unorientable critical pair could be generated (completion **fails**);
  - Critical pair generation might not terminate.
- Fails only if reduction order is incompatible with the new rule.
- (Can show that "infinite" executions lead to semidecision procedure.)

Completion Specified Formally

- Completion typically specified as an **inference system**.
- Operates on tuples (E,R) – set of identities and rewrite system.
- Start with $(E_0,\emptyset)$ and finish with $(\emptyset,R_\infty)$.
- E_0 is the theory and R_∞ is an equivalent convergent system (a completion).

Completion as an Inference System

ORIENT:	$\dfrac{(E \cup \{s \mathrel{\dot{\approx}} t\}, R)}{(E, R \cup \{s \to t\})}$	if $s > t$
DEDUCE:	$\dfrac{(E, R)}{(E \cup \{s \approx t\}, R)}$	if $s \leftarrow_R u \to_R t$
DELETE:	$\dfrac{(E \cup \{s \approx s\}, R)}{(E, R)}$	
SIMPLIFY:	$\dfrac{(E \cup \{s \mathrel{\dot{\approx}} t\}, R)}{(E \cup \{u \mathrel{\dot{\approx}} t\}, R)}$	if $s \to_R u$
COMPOSE:	$\dfrac{(E, R \cup \{s \to t\})}{(E, R \cup \{s \to u\})}$	if $t \to_R u$
COLLAPSE:	$\dfrac{(E, R \cup \{s \to t\})}{(E \cup \{v \approx t\}, R)}$	if $s \overset{\sqsupset}{\to}_R v$

Correctness of Completion

- If executions eventually consider all critical pairs (are **fair**) and can orient every identity (is **non-failing**), completion succeeds.
- Theorem: a non-failing, fair execution with identities E yields a convergent, equivalent rewriting system R, which can be used to solve the word problem for E.

Completion and CGE_2

$$
\begin{array}{lll}
1 * x \approx x & x^{-1} * x \approx 1 & (x * y) * z \approx x * (y * z) \\
f(x * y) \approx f(x) * f(y) & g(x * y) \approx g(x) * g(y) & f(x) * g(y) \approx g(y) * f(x)
\end{array}
$$

- Recall: completion doesn't work with the **two commuting endomorphisms** (CGE_2) theory.
- Doesn't fail (technically) because it never starts.
- How to orient identities? What reduction order to use?

The Reduction Order Requirement

- Completion requires the user to provide a compatible reduction order.
- Can't find one. We've looked.
- Even if we found one, we couldn't specify it – no orders supported by tools (e.g. Waldmeister) are compatible.
- Without an order, completion is useless.

Issues with Completion

1. Compatible orders hard for the user to find and specify.
2. Implementations only implement a few classes, so even if an order exists, user can't make use of it.

The Orient Rule

$$\text{ORIENT:} \qquad \frac{(E \cup \{s \dot{\approx} t\}, R)}{(E, R \cup \{s \to t\})} \qquad \text{if } s > t$$

- Problems manifested in the **orient** rule – only place the presupposed order is mentioned.
- Completion would work for more theories if the system provided the order instead of the user.

A New Orient Rule

- Idea: what if we **use a termination checker** instead?
- New orient precondition: require that adding $s \rightarrow t$ preserves termination of the rewriting system.
- Implies the **existence** of a compatible reduction order.

Correctness of the New Orient Rule

- Different from standard completion in an important way –
- Termination implies the existence of a compatible order, but the **order could be different** each time the orient rule is applied.
- Like performing completion with **multiple orders**.

Completion with Multiple Orders

- A version of completion with multiple orders was used for years (without correctness proof).
- Changing orders is a useful feature.
- If an unorientable identity is encountered, just find another compatible order and keep going.

Multiple Orders Not Correct

- Correctness an open problem for years.
- Settled in the negative by Sattler-Klein in ‘94.
- Multiple orders can yield non-confluent, non-terminating systems.

A Correct Special Case

- But Sattler-Klein also proved that one kind of multi-ordered completion is correct:
- For finite executions without **compose** or **collapse**, completion works with multiple orders.

Compose and Collapse

$$\text{COMPOSE:} \quad \frac{(E, R \cup \{s \to t\})}{(E, R \cup \{s \to u\})} \quad \text{if } t \to_R u$$

$$\text{COLLAPSE:} \quad \frac{(E, R \cup \{s \to t\})}{(E \cup \{v \approx t\}, R)} \quad \text{if } s \overset{\sqsupset}{\to}_R v$$

- Why? These are the only rules that change or remove rules from the current rewriting system.
- Without these, the intermediate rewrite systems form an **increasing chain**.
- The **final** order could have been used from the start without failure.

Constraint System

- Could use new orient rule without compose and collapse, but they're good for performance.
- Instead: check termination of a **constraint** rewriting system not affected by compose and collapse.
- <u>Lemma</u>: Termination of constraint system implies termination of rewriting system and existence of increasing chain of reduction orders.

Revised Completion

ORIENT:	$\dfrac{(E \cup \{s \mathrel{\dot{\approx}} t\}, R, C)}{(E, R \cup \{s \to t\}, C \cup \{s \to t\})}$	if $C \cup \{s \to t\}$ terminates
DEDUCE:	$\dfrac{(E, R, C)}{(E \cup \{s \approx t\}, R, C)}$	if $s \leftarrow_R u \to_R t$
DELETE:	$\dfrac{(E \cup \{s \approx s\}, R, C)}{(E, R, C)}$	
SIMPLIFY:	$\dfrac{(E \cup \{s \mathrel{\dot{\approx}} t\}, R, C)}{(E \cup \{u \mathrel{\dot{\approx}} t\}, R, C)}$	if $s \to_R u$
COMPOSE:	$\dfrac{(E, R \cup \{s \to t\}, C)}{(E, R \cup \{s \to u\}, C)}$	if $t \to_R u$
COLLAPSE:	$\dfrac{(E, R \cup \{s \to t\}, C)}{(E \cup \{v \approx t\}, R, C)}$	if $s \overset{\sqsupset}{\to}_R v$

- Key differences: constraint system *C* and termination predicate in orient precondition.

Completion Search

- What if a if a rule can be oriented two different ways?
- Just try both. **Search** for a correct completion.
- (Search avoids pesky infinite executions mentioned earlier.)
- Breadth-first search guarantees that we will eventually find a completion.

Revised Completion

- Revised method is **correct**.
- Order is **discovered**, not provided.
- With perfect termination-checking ability, the method completes any theory compatible with some reduction order.
- With real termination-checking program that decides a class of orders *O*, then revised method completes any theory compatible with an order in *O*.

Slothrop

- Implementation of revised procedure: Slothrop.
- ~7000-line Ocaml program
- Integrated with AProVE termination checker with help from that team.

Completion of CGE_2

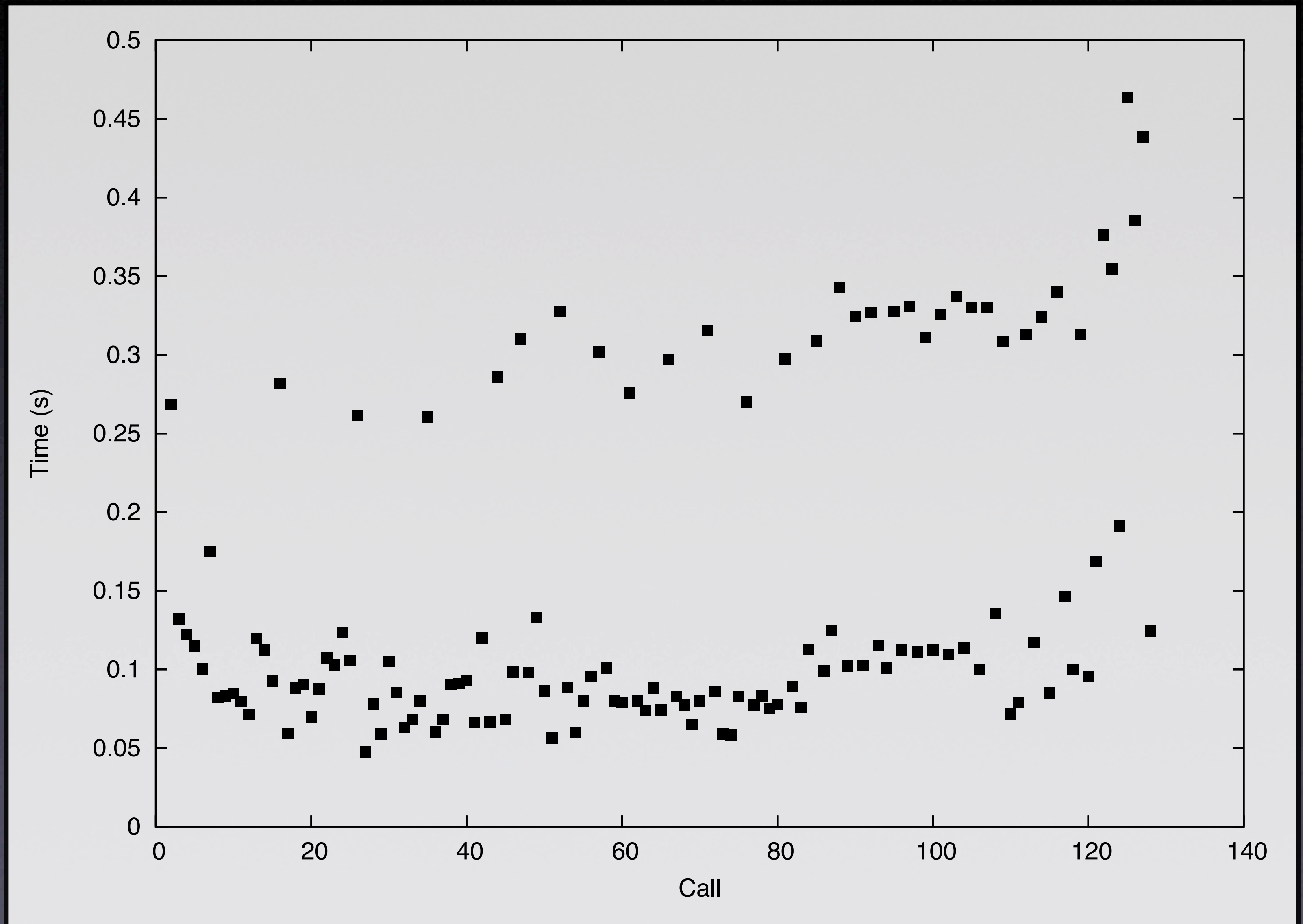
- Slothrop completes a variety of theories (e.g., groups and other algebraic structures).
- Completed CGE_2 – **first ever** automatic completion!

$(x * y) * z \to x * (y * z)$	$f(1) \to 1$
$x^{-1} * x \to 1$	$(f(x))^{-1} \to f(x^{-1})$
$x * x^{-1} \to 1$	$f(x) * f(y) \to f(x * y)$
$x * (x^{-1} * y) \to y$	$f(x) * (f(y) * z) \to f(x * y) * z$
$x^{-1} * (x * y) \to y$	$g(1) \to 1$
$(x * y)^{-1} \to y^{-1} * x^{-1}$	$(g(x))^{-1} \to g(x^{-1})$
$1 * x \to x$	$g(x) * g(y) \to g(x * y)$
$x * 1 \to x$	$g(x) * (g(y) * z) \to g(x * y) * z$
$1^{-1} \to 1$	$f(x) * g(y) \to g(y) * f(x)$
$(x^{-1})^{-1} \to x$	$f(x) * (g(y) * z) \to g(y) * (f(x) * z)$

Performance

- Time: 1m to find G completion, 2m for GE_1, 1.5h for CGE_2.
- Calls to AProVE: 40 calls to complete G, 130 for GE_1, 4000 for CGE_2.
- > 95% of runtime spent in AProVE, but most calls return in < 0.5s.

AProVE is Fast



Slothrop

- Efficiency is the only limitation of technique.
- Works well on small theories, but is slow on large theories.
- Improved termination checking will help, better search heuristics will help more.
- **Open question**: when is a partial completion nearly a completion?

Conclusion

- Thanks to:
  - Aaron Stump and Eddy Westbrook for big ideas and major contributions to correctness proof.
  - Everyone here for sitting through the whole dang talk.

Conclusion

- Fin.