Knuth-Bendix Completion with Modern Termination Checking

Ian Wehrman
ACL2 Seminar
August 20, 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:

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

Group Theory Proof

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

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

Automating Group Theory Proofs

- Found proof fully automatically using a tool called **Waldmeister**.
- Implements an algorithm called **completion**.
  - Input: theory (finite set of identities).
  - Output: rewriting system 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 syntactic 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

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

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

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

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!**

What Went Wrong?

- Beyond the theory, completion also needs an **order**, used to orient each identity.
- Waldmeister either accepts an order as input, or tries to guess one before starting.
- Works for simple theories; fails on CGE_2.
- Why? Waldmeister only looks for orders of a certain class, and no suitable order in that class exists for CGE_2.

Life without a Completion

- Without a completion, we must resort to other methods to solve word problem, e.g. paramodulation, or our heads.
- Other methods generally less efficient, and don't yield completions.
- Completions are preferred because they can lead to extremely efficient decision procedures and related algorithms (c.f., algebraic proof mining and SMT).

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 from term rewriting theory (particularly **proving termination**.)
  - Discuss **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.

$$\boxed{s_1 \longrightarrow s_2 \longrightarrow \cdots \longrightarrow s_n \stackrel{?}{=} t_n \longleftarrow \cdots \longleftarrow t_2 \longleftarrow t_1}$$

Rewriting to Normal Form

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

Properties of Rewriting Systems

- These requirements correspond to two 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))* could yield either reduction sequence:

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

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

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

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

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

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?

Termination Warm-up

- Answer to second question: if we can't prove termination, we're completely stuck.
- So let's try prove termination.
- Do the following TRS's terminate? Why?

1. $\{f(g(x)) \to f(h(f(x))) \qquad g(g(x)) \to f(g(h(x)))\}$
2. $\{f(f(x)) \to g(x) \qquad g(g(x)) \to f(x)\}$
3. $\{f(f(x,x),y) \to f(y,y)\}$

Decidability of Termination

- Termination not just undecidable (reduction from halting prob), it's also hard.
  - Undecidable classes are vast: 3-rule monadic systems, single-rule systems.
  - Decidable classes are modest: right-ground, LPO.

Reduction Orders

- The foundation of most termination proofs is a **reduction order**.
- Def: A reduction order is one that is 1) compatible with contexts; 2) closed under substitutions; and 3) well-founded.
- An order > is **compatible** with a rewriting system if $l > r$ for all rules $l \to r$.
- Thm: a system is terminating iff a compatible reduction order exists.

Proving Termination

- How to find a compatible reduction order?
- Methods divided into three classes:
  - Semantical – defined indirectly, by mapping terms into well-founded algebra.
  - Syntactical – defined directly, using the subterm relation (simplification orders).
  - Transformational – use a termination-preserving map ϕ; then termination of $\phi(R)$ implies termination of R.

Semantical Methods

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

- Idea: interpret terms in N; rewriting yields smaller interpretation; no infinite reduction sequences since N is well-founded.
- $f^N : N \to N$, $g^N : N \to N$, $h^N : N \to N$.
- $f^N(x) = x$, $g^N(x) = x+1$, $h^N(x) = x$.
- $[f(g(x))]^N = 1+x > x = [f(h(f(x)))]^N$
- $[g(g(x))]^N = 2+x > 1+x = [f(g(h(x)))]^N$

Σ-Algebras

- A **signature** $\Sigma = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- A Σ-**algebra** $\mathcal{A} = (A, \Sigma^{\mathcal{A}})$ s.t.
  $\Sigma^{\mathcal{A}} = \{f^{\mathcal{A}}:A^n \rightarrow A \mid f \in \Sigma^n\}$
- $(\mathcal{A},<)$ is a **well-founded** Σ-algebra if $<$ is well-founded on A.
- A **monotone** well-founded Σ-algebra has, $\forall\ f^{\mathcal{A}} \in \Sigma^{\mathcal{A}}; a,b \in A : a < b \rightarrow f^{\mathcal{A}}(a) < f^{\mathcal{A}}(b)$.

Semantical Methods

- Def: Let $\mathcal{A}$ be a well-founded monotone Σ-algebra. $t <^{\mathcal{A}} t'$ iff $t^{\mathcal{A}} < t'^{\mathcal{A}}$ for every variable assignment.
- Lem: $<^{\mathcal{A}}$ is a reduction order.
- Thm: A TRS terminates iff it admits a compatible well-founded monotone Σ-algebra.

Polynomial Orders

- Most common algebras used with semantic interpretation are polynomials with coefficients in N over N^+.
- Induces the class of reduction orders called **polynomial orders**.
- Decent heuristics exist for finding polynomials to associate with each function symbol.

Weaknesses of $>^{\mathcal{A}}$

- But, polynomial orders have drawbacks:
  - Finding a suitable algebra is undecidable.
  - Proving compatibility is undecidable.
  - Proof-theoretic strength severely limited. Can't prove termination if TRS:
    - has reduction sequences of length $> 2^{2^n}$
    - computes a super-polynomial number-theoretic function.

Syntactical Methods

- We can define reduction orders without relying on the existence of a well-founded algebra.
- Idea: $t_1 < t_2$ iff t_1 is a proper subterm of t_2. (But is it a reduction order? Yes.)
- Weak – length of reduction sequences limited by depth of term.
- Want to find an **extension** of this order.

Simplification Orders

- Thm: **any** rewrite order with the subterm property is well-founded (i.e., a reduction order) – called **simplification orders**.
- Extremely important result in termination analysis because simplification orders
  - are both powerful and decidable;
  - have simple recursive definitions;
  - are easily tested for compatibility.

Simplification Orders

- Most common: recursive path orders (RPO) and Knuth-Bendix orders (KBO).
- Idea behind recursive path orders:
  - define partial **precedence order** on Σ;
  - decide order by first comparing function symbols then recursively comparing arguments.
- RPO's differ in how the arguments are compared in aggregate.

Definition of LPO

Let Σ be a finite signature and $>$ be a strict order on Σ. Then $s >_{\text{lpo}} t$ iff

LPO1 $t = Var(s)$ and $s \neq t$; or

LPO2 $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$; and

- **LPO2a** $\exists 1 \leq i \leq m : s_i \geq_{\text{lpo}} t$; or
- **LPO2b** $f > g$ and $\forall 1 \leq j \leq n : s >_{\text{lpo}} t_j$; or
- **LPO2c** $f = g$, $\forall 1 \leq j \leq n : s >_{\text{lpo}} t_j$,
  and $\exists 1 \leq i \leq m : s_1 = t_1, \ldots, s_{i-1} = t_{i-1}$
  and $s_i >_{\text{lpo}} t_i$.

LPO Example 1

$$x + 0 \to x \qquad x + s(y) \to s(x + y)$$

- Above TRS shown LPO-terminating with precedence exp s > 0:

1. (LPO1) because rhs is a variable in lhs
2. (LPO2b) + > s in precedence, so check that lhs $>_{lpo}$ of rhs subterms. Yes for both subterms by (LPO1).

LPO Example 2

$$\begin{array}{ll} x + 0 \to x & x + s(y) \to s(x+y) \\ x * 0 \to 0 & x * s(y) \to x + (x * y) \\ exp(x,0) \to s(0) & exp(x, s(y)) \to x * exp(x,y) \end{array}$$

- Above TRS shown LPO-terminating with precedence exp > * > + > s > 0.
- Explanation of multiplication rules:

1. (LPO2b) * > 0, and 0 has no subterms.
2. (LPO2b) * > +, so check that lhs $>_{lpo}$ of rhs subterms; yes by (LPO1) again.

LPO Example 3

$$ack(0, x) \rightarrow s(x)$$
$$ack(s(x), y) \rightarrow ack(x, s(0))$$
$$ack(s(x), s(y)) \rightarrow ack(x, ack(s(x), y))$$

- Above TRS is shown terminating with precedence ack > s > 0

1. (LPO2b) ack > s; (LPO1) ack(0,x) > x
2. (LPO2c) ack(s(x),y) > x,s(0) and s(x) > x
3. (LPO2c) same.

LPO is pretty great

- Much stronger than polynomial orders – proves termination of functions that grow faster than any primitive recursive function.
- Compatibility easily checked in poly time.
- Existence is NP-hard – just try all possible total orders on Σ.

LPO is not perfect

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

- Recall the theory CGE_2 mentioned earlier.
- Waldmeister fails to complete this system because no LPO is compatible with it.
- Why? Stuck on commutativity of *f* and *g*.

Automated Termination Checkers

- There are nifty tools to **automatically** prove termination using previous methods.
- Works for systems that are compatible with any one (or a combination of) a variety of reduction orders.
- E.g., **AProVE**: fast, effective and produces human-readable proofs.
- Could be useful later...?

Proving Confluence

Local confluence: $(\rightarrow)^{-1} \cdot \rightarrow \subseteq \rightarrow^* \cdot (\rightarrow^*)^{-1}$

Confluence: $(\rightarrow^*)^{-1} \cdot \rightarrow^* \subseteq \rightarrow^* \cdot (\rightarrow^*)^{-1}$

- Confluence is undecidable in general, but decidable for rewriting systems that are terminating.
- Shown for terminating systems in two steps:
  - joinability of special finite set of terms implies local confluence;
  - local confluence implies confluence.

Joining Critical Pairs

- 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**.
- Lem: joinability of all critical pairs implies confluence **for terminating systems**.

Critical Pair Example 1

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

- Common instances of rules' lhs rewrites two ways:

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

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 $(E_0,\varnothing)$ and finish with $(\varnothing,R_\infty)$.
- E_0 is the theory and R_∞ is an equivalent convergent system (a completion).

Completion as an Inference System

$$
\begin{array}{lcl}
\textsc{Orient:} & \dfrac{(E \cup \{s \mathrel{\dot{\approx}} t\}, R)}{(E, R \cup \{s \to t\})} & \text{if } s > t \\[2ex]
\textsc{Deduce:} & \dfrac{(E, R)}{(E \cup \{s \approx t\}, R)} & \text{if } s \leftarrow_R u \rightarrow_R t \\[2ex]
\textsc{Delete:} & \dfrac{(E \cup \{s \approx s\}, R)}{(E, R)} & \\[2ex]
\textsc{Simplify:} & \dfrac{(E \cup \{s \mathrel{\dot{\approx}} t\}, R)}{(E \cup \{u \mathrel{\dot{\approx}} t\}, R)} & \text{if } s \rightarrow_R u \\[2ex]
\textsc{Compose:} & \dfrac{(E, R \cup \{s \to t\})}{(E, R \cup \{s \to u\})} & \text{if } t \rightarrow_R u \\[2ex]
\textsc{Collapse:} & \dfrac{(E, R \cup \{s \to t\})}{(E \cup \{v \approx t\}, R)} & \text{if } s \overset{\sqsupset}{\rightarrow}_R v
\end{array}
$$

Correctness of Completion

- If executions eventually consider all critical pairs (are **fair**) and can orient every identity (is **non-failing**), completion succeeds.
- <u>Theorem</u>: 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

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

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

- 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

$$\begin{array}{lll}
\textsc{Orient:} & \dfrac{(E \cup \{s \mathrel{\dot{\approx}} t\}, R, C)}{(E, R \cup \{s \to t\}, C \cup \{s \to t\})} & \text{if } C \cup \{s \to t\} \text{ terminates} \\
\textsc{Deduce:} & \dfrac{(E, R, C)}{(E \cup \{s \approx t\}, R, C)} & \text{if } s \leftarrow_R u \to_R t \\
\textsc{Delete:} & \dfrac{(E \cup \{s \approx s\}, R, C)}{(E, R, C)} & \\
\textsc{Simplify:} & \dfrac{(E \cup \{s \mathrel{\dot{\approx}} t\}, R, C)}{(E \cup \{u \mathrel{\dot{\approx}} t\}, R, C)} & \text{if } s \to_R u \\
\textsc{Compose:} & \dfrac{(E, R \cup \{s \to t\}, C)}{(E, R \cup \{s \to u\}, C)} & \text{if } t \to_R u \\
\textsc{Collapse:} & \dfrac{(E, R \cup \{s \to t\}, C)}{(E \cup \{v \approx t\}, R, C)} & \text{if } s \overset{\sqsupset}{\to}_R v
\end{array}$$

- 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.
- ~3500-line Ocaml program
- Integrated with AProVE termination checker with help from that team.

Completion of CGE_2

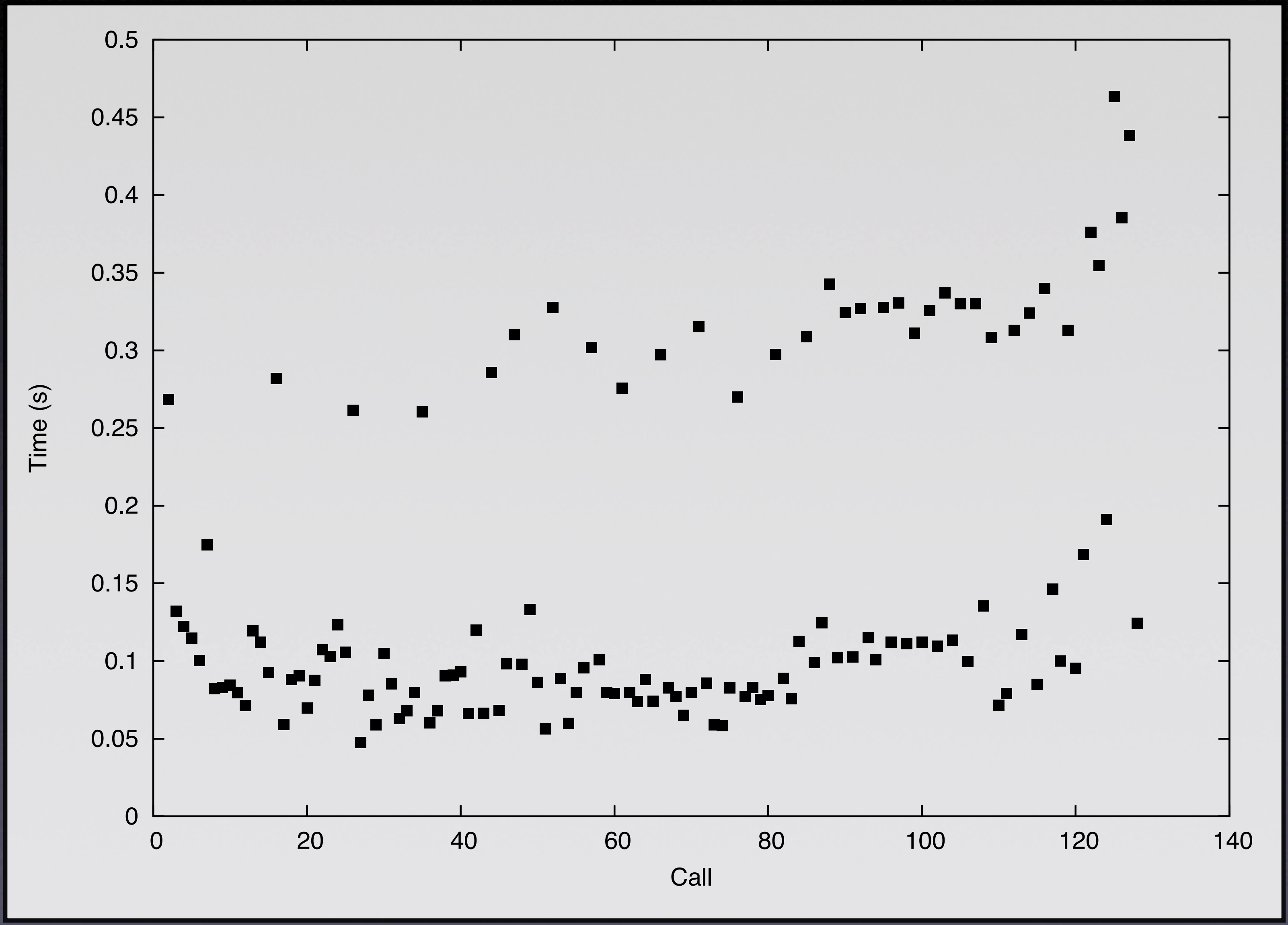
- Slothrop completes a variety of theories (e.g., groups and other algebraic structures).
- Completed CGE_2 – **first ever** automatic completion!

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

Performance

- Time: 6s to find G completion, 30s for GE_1, 15m for CGE_2.
- Calls to AProVE: 30 calls to complete G, 100 for GE_1, 2500 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.