

# 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:

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

# 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: equivalent 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 syntactic equality.

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

# Group Theory Proofs Made Easy

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

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

# 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!

$$1 * x \approx x$$

$$x^{-1} * x \approx 1$$

$$(x * y) * z \approx x * (y * z)$$

$$f(x * y) \approx f(x) * f(y) \quad g(x * y) \approx g(x) * g(y) \quad f(x) * g(y) \approx g(y) * f(x)$$

- Input: theory of groups + two commuting endomorphisms ( $\text{CGE}_2$ ).
- Output: ... **not a completion!**

# What Went Wrong?

- \* Beyond the theory, completion also needs an **order**, used to orient each identity.
- Waldmeister accepts an order as input, or makes a rough guess before starting.
- Works for simple theories; fails on  $\text{CGE}_2$ .
- Why? Waldmeister only looks for orders of a certain class, and no suitable order in that class exists for  $\text{CGE}_2$ .

# 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 preferred because they can lead to highly efficient decision procedures (e.g., for SMT).

# Our Mission

- **Revise** the algorithm used by Waldmeister.
- Use it to **find a completion** for  $\text{CGE}_2$ .
- Solve the word problem for  $\text{CGE}_2$  (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

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

- 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.

# Proof Construction 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.

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

# Rewriting to Normal Form

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

# Properties of Rewriting Systems

- These requirements correspond to two important properties of rewriting systems:
  1. **Termination**: no infinitely long reduction sequences.
  2. **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 I

- The non-confluent, terminating system

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

applied to term  $f(x, g(x))$  could yield either reduction sequence:

1.  $f(x, g(x)) \rightarrow x$
2.  $f(x, g(x)) \rightarrow f(x, x) \rightarrow h(x)$

# Rewriting Example 2

- The confluent, nonterminating system

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

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

$$\begin{aligned} f(x) &\rightarrow g(h(x)) \rightarrow \\ f(h(x)) &\rightarrow g(h(h(x))) \rightarrow \\ f(h(h(x))) &\rightarrow g(h(h(h(x)))) \rightarrow \\ f(h(h(h(x)))) &\rightarrow g(h(h(h(h(x)))))) \rightarrow \dots \end{aligned}$$

# Rewriting Example 3

- The convergent system

$$ack(0, n) \rightarrow n + 1$$

$$ack(m + 1, 0) \rightarrow ack(m, 1)$$

$$ack(m + 1, n + 1) \rightarrow ack(m, ack(m + 1, n))$$

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

$$ack(3, 3) \rightarrow ack(2, ack(3, 2)) \rightarrow ack(2, (ack(2, (ack(3, 1)))) \rightarrow \\ ack(2, (ack(2, (ack(2, ack(3, 0))))) \rightarrow ack(2, (ack(2, (ack(2, ack(2, 1))))) \rightarrow \\ ack(2, (ack(2, (ack(2, ack(1, ack(2, 0))))) \rightarrow \dots \rightarrow 61$$

# 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?

# Decidability of Termination

- If we can't prove termination, we're stuck.
- Unfortunately, termination isn't just undecidable, it's also hard.
  - Undecidable classes are vast: 3-rule monadic systems, single-rule systems.
  - Decidable classes are relatively modest: right-ground.

# Termination Warm-up

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

- Do the following TRS's terminate? Why?
- (Remember: a counter-example requires exhibiting a term that starts an infinite reduction sequence)

# Reduction Orders

- The foundation of most termination proofs is a reduction order.
- Def: A **rewrite order**  $>$  is stable under:
  1. contexts:  $s > t \rightarrow C[s] > C[t]$ ,
  2. substitutions:  $s > t \rightarrow s\sigma > t\sigma$ .
- Def: A **reduction order** is a well-founded rewrite order.

# Characterizing Termination

- Reduction orders characterize termination of term rewriting systems.
- Thm: a TRS is terminating iff it is compatible with some reduction order.
- An order  $>$  is **compatible** with a TRS if  $l > r$  for all rules  $l \rightarrow r$ .

# Proving Termination

- Three ways to show termination:
  1. Semantical – reduction order defined w.r.t. a well-founded algebra.
  2. Syntactical – reduction order defined using the subterm property.
  3. Transformational – apply a termination-preserving map  $\mu$ , then repeat with  $\mu(R)$ .
- Examples of 1 and 2 today.

# Semantical Methods

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

- Idea: interpret ground terms as natural numbers  $\mathbb{N}$ .
- Want interpretation s.t. rules' rhs are always less than lhs with the usual order  $<$ .
- Each rewrite yields smaller interpretation.
- Because  $(\mathbb{N}, <)$  is well-founded, no infinite reduction sequences are possible.

# Example Interpretation

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

- Interpret function symbols as functions in  $\mathbb{N}$ :
  - $f^{\mathbb{N}} : \mathbb{N} \rightarrow \mathbb{N}$ ,  $g^{\mathbb{N}} : \mathbb{N} \rightarrow \mathbb{N}$ ,  $h^{\mathbb{N}} : \mathbb{N} \rightarrow \mathbb{N}$ .
  - $f^{\mathbb{N}}(x) = x$ ,  $g^{\mathbb{N}}(x) = 1+x$ ,  $h^{\mathbb{N}}(x) = x$ .
- Show lhs is greater than rhs:
  1.  $[f(g(x))]^{\mathbb{N}} = 1+x > x = [f(h(f(x)))]^{\mathbb{N}}$
  2.  $[g(g(x))]^{\mathbb{N}} = 2+x > 1+x = [f(g(h(x)))]^{\mathbb{N}}$

# $\Sigma$ -Algebras

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

# Semantical Methods

- Def: Let  $\mathcal{A}$  be a well-founded monotone  $\Sigma$ -algebra. Then  $t <_{\mathcal{A}} t'$  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  $\Sigma$ -algebra.

# Polynomial Orders

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

# Weaknesses of Polynomial Orders

- Existence of suitable algebra is undecidable.
- Worse, proving compatibility is undecidable.
- Proof-theoretic strength is limited. Can't prove termination if TRS either:
  1. has reduction sequences of length  $> 2^{2^{c|t|}}$ ,
  2. computes a super-polynomial number-theoretic function.

# Syntactical Methods

- We can define reduction orders without relying on a well-founded algebra.
- But any order must be well-founded, and this is just what having a semantic interpretation helped with.

# Subterm Ordering

- Idea:  $t_1 < t_2$  iff  $t_1$  is a proper subterm of  $t_2$ .
- Unfortunately, weak – length of reduction sequences limited by depth of term.
- Can we find an **extension** of this order that remains well-founded?

# Simplification Orders

- Thm: **any** strict rewrite order with the subterm property is well-founded.
- Reduction orders with subterm property are called **simplification orders**.
- Important because they (typically):
  1. are relatively powerful,
  2. have simple recursive definitions,
  3. are easily tested for compatibility,
  4. are decidable.

# Recursive Path Orders

- Common orders: recursive path orders (RPO) and Knuth-Bendix order (KBO).
- Idea behind RPO's:
  - define partial **precedence order** on  $\Sigma$ ;
  - decide order by comparing root symbols then recursively comparing arguments.
- RPO's differ in how the arguments are compared: e.g., lexicographically (LPO), as multisets (MPO).

# Definition of LPO

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

**LPO1**  $t = \text{Var}(s)$  and  $s \neq t$ ; or

**LPO2**  $s = f(s_1, \dots, s_m)$ ,  $t = g(t_1, \dots, t_n)$ ; 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  $\exists 1 \leq i \leq m : s_1 = t_1, \dots, s_{i-1} = t_{i-1}$   
and  $s_i >_{\text{lpo}} t_i$ .

# LPO Example I

$$x + 0 \rightarrow x \quad x + s(y) \rightarrow s(x + y)$$

- Above TRS shown LPO-terminating with precedence  $+ > s > 0$ :
  1. (LPO1) because rhs is a variable in lhs
  2. (LPO2b)  $+ > s$  in precedence, so check that lhs  $>_{\text{lpo}}$  of rhs subterm. Yes: (LPO2c) then (LPO1) three times.

# LPO Example 2

$$x + 0 \rightarrow x$$

$$x * 0 \rightarrow 0$$

$$\text{exp}(x, 0) \rightarrow s(0)$$

$$x + s(y) \rightarrow s(x + y)$$

$$x * s(y) \rightarrow x + (x * y)$$

$$\text{exp}(x, s(y)) \rightarrow x * \text{exp}(x, y)$$

- Above TRS shown LPO-terminating with precedence  $\text{exp} > * > + > s > 0$ .
- Multiplication rules:
  1. (LPO2b)  $* > 0$ , and 0 has no subterms.
  2. (LPO2b)  $* > +$ , so check that  $\text{lhs} >_{\text{lpo}}$  of rhs subterms; (LPO1) and (LPO2c), and (LPO1) three times.

# LPO Example 3

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

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

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

- Above TRS is shown terminating with precedence  $ack > s > 0$ 
  1. (LPO2b)  $ack > s$ , then (LPO1).
  2. (LPO2c), (LPO1), (LPO2b), (LPO2a), (LPO1).
  3. (LPO2c), (LPO1), (LPO2c), (LPO1), ...

# 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  $\Sigma$  – but decidable.

# LPO is not perfect

$$1 * x \approx x$$

$$x^{-1} * x \approx 1$$

$$(x * y) * z \approx x * (y * z)$$

$$f(x * y) \approx f(x) * f(y)$$

$$g(x * y) \approx g(x) * g(y)$$

$$f(x) * g(y) \approx g(y) * f(x)$$

- Recall the theory  $\text{CGE}_2$  mentioned earlier.
- Waldmeister fails to complete this system because there is no compatible LPO.
- 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.
- Will be useful later...

# Proving Confluence

- Confluence is undecidable in general, but decidable for rewriting systems that are terminating.
- Proceed by showing joinability of a particular (finite) set of pairs of terms.
- Finite computation that implies confluence.

# Joining Critical Pairs

- Start with a common instance of two rules' lhs:  $s_1$ .
- Rewrite with both rules to two 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**.
- Thm: joinability of all critical pairs implies confluence **for terminating systems**.

# Critical Pair Example I

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

- 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

- But simply orienting the axioms of many theories yields a non-confluent TRS.
- If a 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.

# Equivalent Confluent Systems

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

# Critical Pair Example 2

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

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

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

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

# Completion

- Called **completion procedure**, invented by Knuth in '70.
- Completion can help **solve the word problem**.
- Use the equivalent, convergent 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 pairs  $(E,R)$  – set of identities and set of rewrite rules.
- Start with  $(E_0, \emptyset)$  and finish with  $(\emptyset, R_\infty)$ .
- $E_0$  is the theory and  $R_\infty$  is an equivalent convergent system (the completion).

# Completion as an Inference System

ORIENT:	$\frac{(E \cup \{s \approx t\}, R)}{(E, R \cup \{s \rightarrow t\})}$	if $s > t$
DEDUCE:	$\frac{(E, R)}{(E \cup \{s \approx t\}, R)}$	if $s \leftarrow_R u \rightarrow_R t$
DELETE:	$\frac{(E \cup \{s \approx s\}, R)}{(E, R)}$	
SIMPLIFY:	$\frac{(E \cup \{s \approx t\}, R)}{(E \cup \{u \approx t\}, R)}$	if $s \rightarrow_R u$
COMPOSE:	$\frac{(E, R \cup \{s \rightarrow t\})}{(E, R \cup \{s \rightarrow u\})}$	if $t \rightarrow_R u$
COLLAPSE:	$\frac{(E, R \cup \{s \rightarrow t\})}{(E \cup \{v \approx t\}, R)}$	if $s \xrightarrow{\exists}_R v$

# 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<sub>2</sub>

$$1 * x \approx x$$

$$x^{-1} * x \approx 1$$

$$(x * y) * z \approx x * (y * z)$$

$$f(x * y) \approx f(x) * f(y)$$

$$g(x * y) \approx g(x) * g(y)$$

$$f(x) * g(y) \approx g(y) * f(x)$$

- Recall: completion didn't work with the two commuting endomorphisms (CGE<sub>2</sub>) theory.
- Doesn't fail *per se*, because it never starts.
- How to orient identities? What reduction order to use?

# The Reduction Order Requirement

- Completion requires compatible reduction order, usually provided by the user.
- Can't find one in usual classes. 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

- We'll to solve the following problems:
  1. Compatible orders hard for the user to find.
  2. Implementations of completion only implement a few classes of orders. (Even if a compatible order exists, user can't specify it.)

# The Orient Rule

$$\text{ORIENT: } \frac{(E \cup \{s \approx t\}, R)}{(E, R \cup \{s \rightarrow t\})} \quad \text{if } s > t$$

- Trouble is with the user-supplied order.
- Manifested in the **orient** rule – only place the order is needed.
- Completion would work for **more theories** if the user didn't have to explicitly provide an order at the start.

# 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.

# New Precondition

- Tentative change to orient rule precondition:

$$\text{ORIENT: } \frac{(E \cup \{s \doteq t\}, R)}{(E, R \cup \{s \rightarrow t\})} \quad \text{if } s > t$$

← Original  
Modified

$$\text{ORIENT: } \frac{(E \cup \{s \doteq t\}, R)}{(E, R \cup \{s \rightarrow t\})} \quad \text{if } R \cup \{s \rightarrow t\} \text{ terminates}$$

# 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

- Completion with multiple orders was used for years (without correctness proof) in some systems.
- Useful – 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 by Sattler-Klein in '94 – not correct.
- Multiple orders with completion 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

COMPOSE:	$\frac{(E, R \cup \{s \rightarrow t\})}{(E, R \cup \{s \rightarrow u\})}$	if $t \rightarrow_R u$
COLLAPSE:	$\frac{(E, R \cup \{s \rightarrow t\})}{(E \cup \{v \approx t\}, R)}$	if $s \xrightarrow{\exists}_R v$

- 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 performance suffers.
- Instead: check termination of a **constraint** rewriting system not affected by compose and collapse.
- Lem: Termination of constraint system implies termination of rewriting system and existence of increasing chain of reduction orders.

# Revised Completion

ORIENT:	$\frac{(E \cup \{s \approx t\}, R, C)}{(E, R \cup \{s \rightarrow t\}, C \cup \{s \rightarrow t\})}$	if $C \cup \{s \rightarrow t\}$ terminates
DEDUCE:	$\frac{(E, R, C)}{(E \cup \{s \approx t\}, R, C)}$	if $s \leftarrow_R u \rightarrow_R t$
DELETE:	$\frac{(E \cup \{s \approx s\}, R, C)}{(E, R, C)}$	
SIMPLIFY:	$\frac{(E \cup \{s \approx t\}, R, C)}{(E \cup \{u \approx t\}, R, C)}$	if $s \rightarrow_R u$
COMPOSE:	$\frac{(E, R \cup \{s \rightarrow t\}, C)}{(E, R \cup \{s \rightarrow u\}, C)}$	if $t \rightarrow_R u$
COLLAPSE:	$\frac{(E, R \cup \{s \rightarrow t\}, C)}{(E \cup \{v \approx t\}, R, C)}$	if $s \xrightarrow{\exists}_R v$

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

# Completion Search

- What if a rule can be oriented two different ways?
- Just try both. **Search** for a correct completion.
- (Search avoids problems with 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.
- Integrates with AProVE termination checker, and recently with TPA.
- Applications of orient rule preceded with calls to checker to verify termination of constraint system.
- Best heuristic:  $\text{size}(C + E + \text{cp}(R))$

# Completion of CGE<sub>2</sub>

- Slothrop completes a variety of theories (e.g., groups and other algebraic structures).
- Completed CGE<sub>2</sub> – **first ever** automatic completion!

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

# 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  $CGE_2$ .
- Most of runtime spent in AProVE, but most calls return in  $< 0.5s$ .

# Evaluation of Slothrop

- Efficiency is a limitation, but with patience hard completions are found.
- Works well on small-to-medium theories, slow on large theories.
- Not yet able to find completion of CGE<sub>3</sub>.

# Future Work

- Improved termination checking:
  - Stronger – increase class of theories that can be completed.
  - Faster – find completions faster. Need **incremental** termination checking.
- Improved search heuristic:
  - **Hard question:** how can diverging completion instances be identified?

# Conclusion

- Read more: system description in RTA 2006, details in WUCSE-2006-42.
- Download Slothrop online: [cl.cse.wustl.edu](http://cl.cse.wustl.edu)
- Thanks for sitting through two long talks!

# Conclusion

- Fin.





# Confluence Details

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

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

- Critical pair lemma: joinability of critical pairs implies local confluence.
- Newman's lemma: local confluence + termination implies confluence.