AUTOMATED REASONING

SLIDES 14:

TERM REWRITING SYSTEMS
Term rewriting
Overview of Knuth Bendix completion
Properties of rewrite systems
Church-Rosser
Confluence
Termination
Relation between the properties
Using confluent rewrite systems

KB - AR - 2009

Term Rewriting Systems

14ai

- All sentences are unit equations (∀ is implicit).
- Problem is usually to show that two terms are equal modulo equations. Although this could be done using paramodulation ....
- To cut down the search space the equations are used in one direction only, called *orienting* the equations.
- A more general problem is to find bindings for variables such that two terms are equal modulo equations. (See later.)

EXAMPLES using oriented equations

1. x+0 => x 2. x+s(y) => s(x+y)

s(0)+s(s(0)) => s(s(0)+s(0)) (by 2) => s(s(s(0)+0)) (by 2) => s(s(s(0))) (by 1)

ie s(0)+s(s(0)) and s(s(s(0))) are equal modulo the equations 1 and 2.

Also:
s(z)+s(s(0)) => s(s(z)+s(0)) (by 2) => s(s(s(z)+0)) (by 2) => s(s(s(z))) (by 1)

Notice that bindings can be applied to the rules (1 and 2) but not the terms;

We can't rewrite s(u+v) using 1 or 2 (L=>R) since v is not known to be 0 or s(?)
We can't rewrite s(u+v) using 1 or 2 (R=>L) as arrow goes in other direction

Some Terminology of Rewrite Systems

14aii

- A *rewrite rule* is an oriented equation l => r, s.t. all variables in r occur in l.
- An expression e[s] *rewrites* to e[rθ] (e[s]=>e[rθ]) by l => r if s = lθ

Note: if an expression contains no variables, none are introduced by rewriting; i.e. ground terms remain ground.

- s =>*t denotes s rewrites to t using none or more steps
- A term is *irreducible* (canonical) *w.r.t. a rewrite system* if no rule applies to it.
- A term may rewrite forever: Given: 3. x+y => y+x
  a+b => b+a => a+b => b+a => ....
- A term may be rewritten in more than 1 way by a set of rules:

Example:
4. 0+x => x 5. -x+x =>0 6. (x+y)+z => x+(y+z)

0+((-1+-1)+1) => (-1+-1)+1 => -1+(-1+1) => -1+0
0+((-1+-1)+1) => 0+(-1+(-1+1)) => 0+(-1+0) => -1+0

But sometimes different orders may yield different results:
(--1+-1)+1 => 0+1 => 1
(--1+-1)+1 => --1+(-1+1) => --1+0

The aim of the Knuth Bendix Procedure is to eliminate this

If the data consists only of equations there are special techniques that can be applied to show a given goal. A set of equations can be used as a *term rewriting system*. This requires that (i) the equations are orientated and used in paramodulation steps in one direction only, (ii) they are not used to paramodulate into each other, and (iii) variables in the term being paramodulated into are not bound by the step. We will consider these things again later.

With the restrictions (i), (ii) and (iii), the proofs can be written down in a simpler way, when they are called *rewrite proofs* . There are two kinds of rewriting steps - simple *rewriting*, when the paramodulation step is restricted so that the term being paramodulated into is not instantiated by the step, and *narrowing*, when there is no such restriction. (See slide 14av.)

Some simple examples show that limiting the use of equations to a single direction and restricting their use can prevent some true goals from being derived. For example, consider a=>b and a=>c, which we know should entail b=c. However, if we are only allowed to substitute for a, then the negated goal ¬(b=c) cannot be refuted. We need the additional equation b=>c, from which we can derive the goal ¬(c=c) and hence [ ].

To avoid this limitation, the rewriting equations should satisfy the *Church-Rosser* property, or equivalently, *confluence*. The Church-Rosser property guarantees that if two terms can be shown to be equal (eg by refuting ¬(a=b) by paramodulation and reflexivity), then they can be rewritten into a common term by the orientated equations. In the above example, the rewriting equations do not have this property, as clearly b=c, yet b and c do not rewrite into a common term.

The *Knuth-Bendix Completion procedure* will attempt to find, from a given set of equations, a new set of (equivalent) rewrite rules that possess the Church-Rosser property.

14aiii

Rewriting and Paramodulation (1)

In general, given some equations, to show s=t by paramodulation, start from ¬(s=t) and try to use equations to turn both s and t into some z, deriving ¬(z=z) and then resolve with x=x. i.e. ¬(s=t) + equations ==>*[ ]

We'll write s =*t to denote that both s and t can be reduced to z by paramodulation

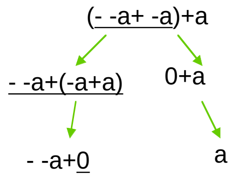
Example: (4) 0+x => x (5) -x + x => 0 (6) (x+y)+z => x+ (y+z)

If (4) - (6) are treated as equations, from ¬(--a+0=a) derive [ ] by paramodulation:
¬(--a+0=a) ==>(5) ¬(- -a+(-x1+x1) = a) ==>(6) ¬((- -a+ -x1)+x1 = a)
==>(5) (and x1/a) ¬(0+a = a) ==>(4) ¬(a=a) ==> [ ] (resolve using reflexivity)

Hence --a+0 =* a

If (4)-(6) are treated as rewrite rules can transform --a+0 into a *only* if rules can be used in both directions; i.e. it is *not* a rewrite proof. <= means rule is used in reverse

- -a+0 <= (5)
by instance -a+a==>0
- -a+(-a+a)<= (6)
(- -a+ -a)+a => (5)
0+a => (4)
a



If rewrite rules may be used in both directions and then s and t both rewrite to z we write s <=>* t. Hence --a+0 <=>* a

Paramodulation and Narrowing

- An expression e[s] is *narrowed* by l => r if sθ = lθ and (e[s])θ=> (e[r])θ.
- If no bindings are made to vars in the terms being rewritten, it is called *rewriting*.
- If bindings are made to vars in the terms being rewritten it is called *narrowing*.

Example: 1. x+0 => x 2. x+s(y) => s(x+y) 3. y=y
s(0)+v *narrows* to s(s(0)+y1) by 2, if v==s(y1) and *narrows* to s(s(0)) if y1==0

- Narrowing corresponds to using paramodulation with **oriented** equations
- Rewriting corresponds to using restricted paramodulation with **oriented** equations

These examples use **oriented** paramodulation – they use equations in direction of =>

¬(s(0)+s(s(0)) = x) ==>(p2)
¬(s(s(0)+s(0)) = x) ==>(p2)
¬(s(s(s(0)+0)) = x) ==>(p1)
¬ (s(s(s(0))) = x) ==>
[ ] (r3 if x==s(s(s(0)))

¬(0+s(0)=s(0)+0) ==>(p2)
¬ (s(0+0)=s(0)+0)==>(p1)
¬(s(0+0)=s(0)) ==>(p1)
¬(s(0) = s(0))==>
[ ] (r3)

¬(s(0)+x = s(s(0)))
==> (p2 if x==s(y1))
¬(s(s(0) + y1) = s(s(0)))
===> (p1 if y1 ==0)
¬(s(s(0))) = s(s(0)))
==> [ ] (r3)

ie: to show ∃x(t1=t2), derive [ ] from the negated goal
We'll return to Narrowing at the end of the course.

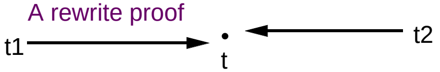
Summary of Rewriting So Far

Given a set of rewrite rules:

- To show s=*t:
either: rewrite s into t, (s =>*t), or rewrite t into s, (t =>* s)
or rewrite s into r and rewrite t into r (s =>* r and t =>* r)
– all steps in the direction of =>

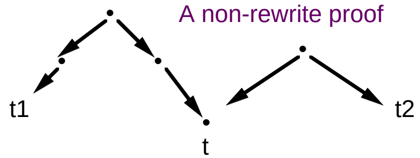
- This is essentially using paramodulation in direction of =>,
to derive [] from ¬(s=t) (needs additionally one resolution step using x=x)

A rewrite proof



Uses directions of =>

A non-rewrite proof



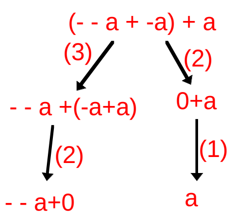
Does not always use directions of =>

Completion – Informal Overview (Specific case)

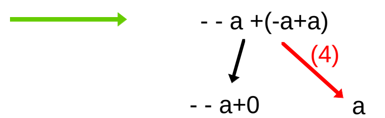
Example: Want to show: --a + 0 =* a but using all rules in => direction

Given (1) 0+x => x (2) -x + x => 0 (3) (x+y)+z => x+(y+z)

A non-rewrite proof

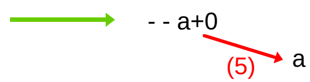


Suppose could derive (4) -x1+(x1+z) => z



Still not a rewrite proof

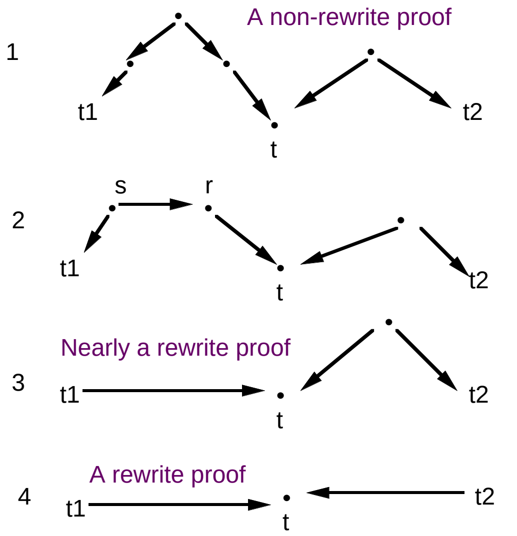
Suppose could derive (5) - -z+0 => z



A rewrite proof

The Knuth Bendix Procedure tells us how to derive (4) and (5) from (1) - (3)

Completion – Informal Overview (General case)



Want to derive some new rewrite rules to enable a 'tunnelling' effect.Then t1 and t2 can be rewritten to the same irreducible term t. (See 4)

The crucial terms are those at the top of the peaks which can be rewritten in more than one way, say into s and r.

s and r can be obtained from the rewrite rules and used to form another rewrite rule that reduces the size of the peaks.

Eventually all peaks will be removed and a simple rewrite proof can be found.

<u>Critical Terms (i)</u>

In general, a rewrite proof to show terms t1 and t2 are equal will rewrite t1 and t2 to a common term t. However, sometimes this can only be carried out if some of the steps are made in the *wrong* direction (i.e. using the rewriting equations from *right to left* instead of from *left to right*.) In this case the "proof" would have one or more *peaks*. The example on 14bi is like this. The term at the apex of the peak is (--a +-a)+a, which can either be rewritten into --a +(-a +a) by (6) and then into --a+0 by (5), or into 0 +a by (5) and then into a by (4).

If there is a peak in the proof, then at the apex there is a term *p* that can be rewritten in two different ways. Such terms as *p*, called *critical terms*, play a crucial role in the Knuth-Bendix procedure and can be rewritten (in 1 or more steps) into two <u>different</u> terms *s* and *r*. (If *s* and *r* could be rewritten to a common term, then there would be no need to go to the top of the peak and back.) The Knuth Bendix procedure finds cases of *most general* critical terms which rewrite to a *critical pair* of (different) terms *s* and *r* from which a rewrite rule can be derived, either *s => r* or *r => s*; this rule can be used to flatten out the peak. It allows a kind of tunnelling effect to avoid the apex.

In the example above the critical term (--a + -a) +a is an instance of the critical term (-x+x) +z. A new rule is found from the result of rewriting this in two ways, namely -x+(x+z) => z. This new rule will allow a shorter way to show --a +0=a:
--a+0 <=--a+(-a+a) => a. It might be quite useful for other rewrite proofs (in this domain) as well. The Knuth Bendix procedure gives a way of finding these new rules.

<u>Critical Terms (ii)</u>

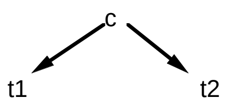
Finding critical terms is quite easy. Given two rules r1 and r2, if the LHS of r1 can be unified with the LHS of r2 or with a subterm of the LHS of r2, then the "common" instance can be rewritten by both r1 and r2. By applying rewrite rules to both results of this rewriting as far as possible, two terms will be derived that are either the same (no problem), or not. When they are not the same the two different terms yield a new rule. This overlapping and matching is called *superposition*. Actually, it is also paramodulation of one rule into another.

For example, suppose there are two rules r1: f(x,x)=>x and r2: f(a,u)=>b.
The common instance (and the critical term), found by superposition, is f(a,a) and it can be rewritten to both a (by r1) and to b (by r2). The new rule would be (say) b=>a.
This can be found by paramodulation too:paramodulate f(a,u)=b into f(x,x)=x to give b=a (bind u/a and x/a). This new rule is needed to show by rewriting that f(b,a) =>* b, which would not otherwise be possible by r1 and r2 alone, even though we can show f(b,a) <=>* b: f(b,a) <= f(f(a,a),a) => f(a,a) => b (using r1 and r2 only).
Using new rule and r1: f(b,a) => f(a,a) => a and b=> a.

The equivalent oriented paramodulation derivation would be: ¬(f(b,a)=b) ==>(2) ¬(f(f(a,u1),a)=b) ==>(1) (and u1/a) ¬(f(a,a)=b) ==>(2) ¬b=b ==>[] (by resolution with x=x). With the new rule we can go directly from ¬f(b,a)=b to ¬f(a,a)=b.

Paramodulation is therefore used in two ways in finding critical pairs: first in superposition and then in rewriting. In rewriting a restricted form is used.

Critical terms and Critical Pairs (1)

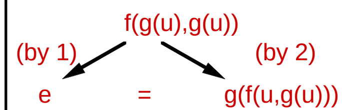


if c can be rewritten in two ways it is called a *critical term*
t1 and t2 are called a *critical pair*

(1) f(x,x) =>e
(2) f(g(u),v) =>g(f(u,v))

Most general critical terms are found by overlapping L.H.S. of rules in a process called *Superposition*.

ie an instance of a L.H.S. can be rewritten in more than one way. Find instance by unifying LHS of (1) and (2).



Obtain a new rule:

(3) g(f(x,g(x))) => e

Here it's f(g(u),g(u)) which will rewrite by (1) and (2):
f(g(u),g(u))=>e (1) and =>g(f(u,g(u)) (2)

Superpositions using (3): remember: either overlap LHSs of 2 rules, or LHS or one rule with a subformula in another rule

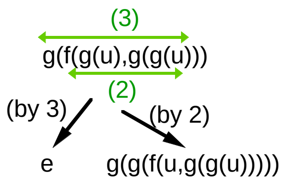
Unify f(x,g(x)) in (3) with f(g(u),v), or
Unify g(u) in (2) with g(f(x,g(x))), or
There's one more - can you find it?
Hnt: consider 2 copies of (3)

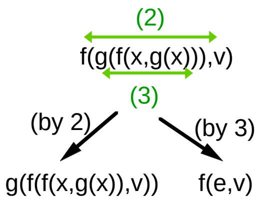
Critical Terms and Critical Pairs (2)

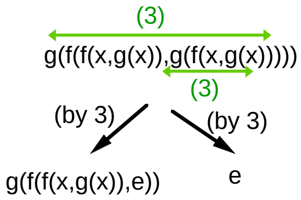
14bvi

(1) f(x,x) =>e
(2) f(g(u),v) =>g(f(u,v))
(3) g(f(x,g(x))) => e

(4) g(g(f(u,g(g(u))))) => e
(5) g(f(f(x,g(x)),v)) => f(e,v)







Superposition and Paramodulation

14bvii

We saw already that rewriting is a restricted form of paramodulation

Superposition and forming critical pairs is also paramodulation:

(1) f(x,x) = e   (2) f(g(u),v) = g(f(u,v))   (3) g(f(x,g(x))) = e

Use (1): unify f(x,x) with f(g(u),v)
giving f(g(u),g(u)) = e and f(g(u),g(u)) = g(f(u,g(u)))
to obtain e = g(f(u,g(u))) (or vice versa) by paramodulation.

Generally:

L1 = R1 and L2[L3] = R2 (meaning L3 occurs in context L2) and L1θ = L3θ
gives L2 [R1θ] θ = R2θ (ie replace L3θ = L1θ by R1θ)

In the example:

L1 is f(x,x) and L3 is f(g(u),v); the context L2 is empty; θ is {x/g(u), v/g(u)}
R1 is e and R2 is g(f(u,v)); R1θ =e and R2θ = g(f(u),g(u))
yielding:
L2[e] = e = g(f(u,g(u)))

Superposition:

14bviii

Example 1. On slide 14bv/14bvi rule (3) and rule (2) can be superposed in two different ways: the first way yields a critical term g(f(g(u), g(g(u)))), which rewrites by (2) into g(g(f(u,g(g(u))))) and by (3) into e giving new rule g(g(f(u,g(g(u))))) =>e. The second way yields a critical term f(g(f(x,g(x))),v), which can be rewritten by (2) into g(f(f(x,g(x)),v)) and by (3) into f(e,v). This gives another new rule g(f(f(x,g(x)),v))=>f(e,v). Rule (3) can be superposed onto a copy of itself:

g(f(x,g(x))) matches with g(x1) in the copy g(f(x1,g(x1)), rewriting to g(f(f(x,g(x)),e)) and also to e, giving the rule g(f(f(x,g(x)),e)) ==>e.

Note also that g(f(u,g(u)) on slide 14bvi cannot be further rewritten by (1) or (2); it can only be narrowed.

Example 2. Applying superposition to the example of 14bix, the first attempt at a new rule yields nothing. Although a term that matches (0+y)+z can be rewritten in two different ways, the result is the same eventually. But the second attempt, using rules (2) and (3), in which (x+y) in (3) is matched with -x1+x1 from (2), gives a new rule -x1+(x1+z)=>z. In the example, this allows - -a+(-a+a) to be rewritten into a, so the rewrite proof using this rule in addition to rules (1-3) is - -a+0 <= - -a+(-a+a)=>a (see slide 14bx). This has a smaller peak than before (and has a new critical term). The last step gives another new rule which allows - -a+0 to be rewritten directly into a.

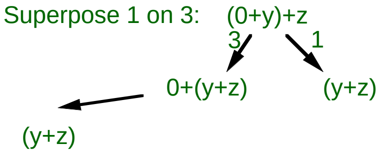
If the example on 14bix is continued, after some more superpositions it will eventually terminate, there being no new rules produced. But the example on 14bvi does not terminate - there are always new (and more and more complex) rules that can be derived.

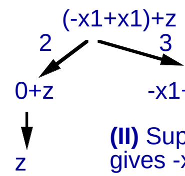
Example: **Want to show:** --a + 0 => a

14bix

Given (1) 0+x => x (2) -x + x => 0 (3) (x+y)+z => x+(y+z)

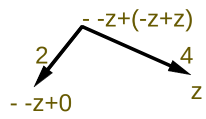
(I) Superpose 1 on 3:





(II) Superpose 2 on 3:
gives -x1+(x1+z) =>z (4))

(III) Superpose 2 on 4:
gives - - z+0 => z (5)



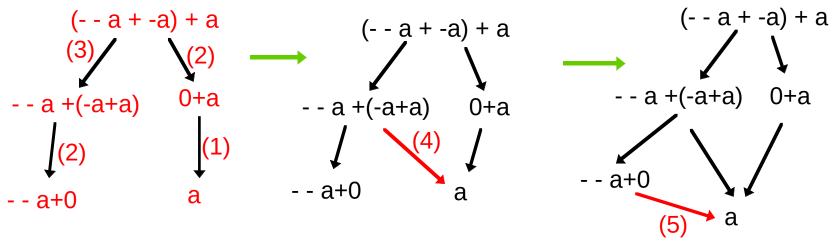
(IV) Now, using (5) can
rewrite - - a + 0 into a

Exercise:
Find some more rules that will
allow to rewrite --a into a.
Hint: Try (3) + (5) to give (6) and
then use (6) to derive z+0=>z

Example: **Want to show:** --a + 0 => a

14bx

Given (1) 0+x => x (2) -x + x => 0 (3) (x+y)+z => x+(y+z)
Derived (4) -x1+(x1+z) => z and (5) - -z+0 => z



Example:

1 0+y =>y
2 s(x) +y => s(x+y)

No possibilities here for overlapping LHSs except overlapping on a variable - only ever lead to equations of the form t1=t1, so no extra rules. eg overlap 0+y1 on x in (2). Effect is to bind x==0+y1

s(0+y1)+y =>(1) s(y1)+y and =>(2) s(y1+y), OR
s(0+y1)+y =>(2) s((0+y1)+y) and =>(1) s(y1+y)

PROPERTIES OF REWRITE SYSTEMS (2)

14ci

- Would like a rewrite system R to be *complete*

  If s =* t then ∃u[s=>*u and t =>*u]

i.e. when two terms are equal want to prove that they are by rewriting.
This is called the Church Rosser property.

- and *sound* If ∃u[s =>*u and t =>*u] then s =*t

i.e ¬(s=t) ==>* [ ] by paramodulation
i.e. two terms proved equal by rewriting are so.

To be useful, a rewrite system should also terminate -
else how could you use it to conclude ¬(s =* t)?

- A rewrite system is called *Noetherian* (terminating) if there is no infinite sequence of rewrites of the form s0 => s1 => … => sn =>...
(eg f(x,y) => f(y,x) is not terminating)

PROPERTIES OF REWRITE SYSTEMS (3)

14cii

Soundness: If ∃u[s =>*u and t =>*u] then s =*t

Proving **Soundness** is quite easy:
Recall that rewrite rules are also equations and rewriting is restricted paramodulation;

Hence
s =>* u and t =>* u implies s =* u and t =*u;
by paramodulation ¬(s=u) ==>* ¬(u=u) (1) and ¬(t=u)==>* ¬(u=u) (2)
(all by EQAX)
Now, given ¬(s=t) first apply steps of (1) to s to derive ¬(u=t),
then apply steps of (2) to to derive ¬(u=u),
and then use EQAX1 and resolution.

PROPERTIES OF REWRITE SYSTEMS (4)

14ciii

- *Church-Rosser* property:
  if s=*t then ∃u[s=>*u and t=>*u]
  i.e. equal terms rewrite to the same term.

- *Confluence*:
  if s =>*u and s =>* v then ∃t[u=>*t and v =>*t]
  i.e. if a term rewrites to 2 other terms then those terms rewrite to a common term.

- *Local confluence*:
  if s=>u and s=>v then ∃t[u=>*t and v=>*t].

Some Useful Facts (Proofs later)

(**Fact A**) R is Church-Rosser iff R is Confluent

(**Fact B**) If R is confluent and terminating then every term has a unique normal (irreducible) form. We say R is *canonical* .

(**Fact C**) If R is locally confluent and terminating then R is confluent.

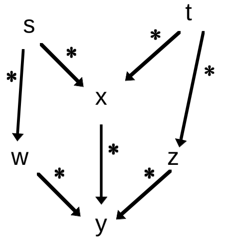
USING A REWRITE SYSTEM to SHOW s=t

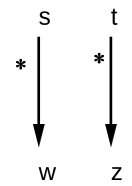
14civ

Given: R, a *confluent* and *terminating* rewrite system and two terms s and t.

(i) Since R is confluent it is *sound* and *complete*.
(ii) Apply R to s and t; since R is terminating the rewriting will stop.
(iii) Suppose s =>* w and t =>*z and w and z are identical.
(iv) Then s =* t (by soundness).
(v) Suppose s =>* w and t =>*z and w and z are not identical. Then s =*t is false:

<u>Proof of (v):</u> Suppose s =*t were true; by completeness ∃x[s =>*x and t =>*x] and by Fact B x rewrites to a unique irreducible term y (say). Hence s and t also rewrite to y uniquely, contradicting that w and z are not identical. (See left below)





If w and z are identical then s =*t

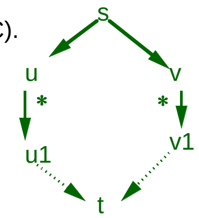
If w and z are different then ¬(s =*t)

Basis of the KNUTH-BENDIX procedure

14cv

- Using the facts A, B and C, and given a rewrite system R, to show R is complete you need to:

- show R has the Church-Rosser property;
- i.e. show R is confluent (by Fact A);
- i.e. show R is locally confluent and terminating (by Fact C).



If R is not locally confluent, then the dotted part in the diagram cannot be completed; so add rule u1 => v1 (or v1 => u1). The two terms u and v then rewrite to a common term, namely v1 (or u1).
This is the basis of the Knuth Bendix procedure.

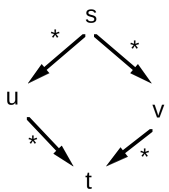
Knuth Bendix relies on rewriting sequences being terminating. Informally, "Termination" will occur if each term in a rewriting sequence is "smaller" than the previous one and no infinite descending chains of such sequences can exist.

If (i) for all rewrite rules the RHS is "smaller" than the LHS, and
(ii) reducing a subterm of a term also reduces the term,
then sequences of rewrites will lead to smaller and smaller terms.
As long as the ordering chosen is <u>well-founded</u>, termination will always occur.

Proof of FACT A

14di

Church-Rosser → confluence :

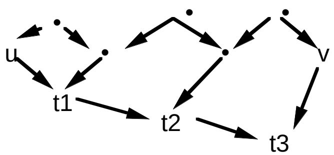


Suppose s =>* u and s =>*v:
then u =* v (turn around steps from s to u)
hence by assumption the rules have the Church-Rosser property and ∃t[u =>*t and v =>*t].

Confluence → Church-Rosser

Suppose u =* v

Let P(n) be "Confluence + a rewrite proof using n peaks => ∃t[u =>*t and v =>*t] ".



<u>Base– P(0)</u>: Either: u =>*v or v =>*u
or u =>* t' and v =>* t' (i.e. no peaks)
Clearly ∃t[u =>*t and v =>*t] is true in all cases.

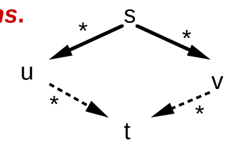
<u>Ind. Step</u> – let n>0 and assume as IH that P(n-1). We show P(n):
Suppose confluence and a rewrite proof using n peaks.

Then t1 exists by confluence and t1=* v ; there are n-1 peaks in the proof to show t1=* v; hence (by IH) ∃t3 [t1 =>*t3 and v =>*t3].
Since u =>*t1, ∃t3 [u =>*t3 and v =>*t3] and so P(n) holds.

PROOF OF FACT B:

14dii

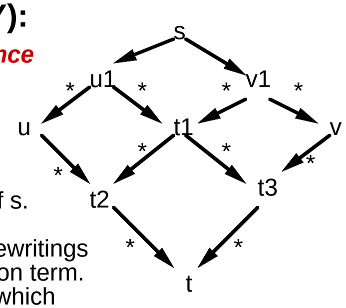
Confluent and terminating implies unique normal forms.



Suppose there were two different normal forms for s, namely u and v, u≠v.
By confluence u and v rewrite to a common term, which contradicts irreducibility. Termination ensures s does not rewrite for ever (so u,v exist).

PROOF OF FACT C (BUNDY):

Local Confluence + Termination → Confluence



- Assume Local confluence and termination.
Termination ensures that there are a finite number of terms obtained by rewriting s.
Let s be arbitrary and suppose s rewrites to two different terms u and v.
- Use structural induction over set of rewrites of s.
- *Ind Step* : Assume that all terms obtained by rewriting s satisfy confluence; i.e. u1,v1 and rewritings of these. Show that u and v rewrite to a common term.
- Consider the first steps from s to u and to v, which reach u1 and v1. t1 exists by local confluence. By hypothesis, since u1,t1,v1 are rewritings of s, t2 and t3 exist, hence t exists. Hence, u and v also rewrite to a common term.

Comments on Slides 14d:

In the proof of Fact A, the induction proof allows to conclude that P(n) holds for every n≥0. Since u =* v there must exist a rewrite proof, even if it uses some equations in the wrong direction. Remember that u and v are ground and the derivation by paramodulation to show ¬(u=v)==>[] can always be made into a ground derivation. This follows from the completeness of paramodulation. This rewrite proof must have n≥0 peaks and hence the property P(n) can be applied to derive the Church-Rosser property that ∃t3 [t1 =>* t3 and v=>* t3].

For Fact C: Let *s* be an arbitrary term. Structural induction over the set of all terms obtained by rewriting *s* is used to show that confluence holds for *s*. Note that there is a finite number of such terms as R is terminating.

The Induction Hypothesis states that, for all terms *t* obtained from *s* by rewriting, *t* satisfies confluence.
Let *s* rewrite to two different terms *u* and *v* and let *u1* and *v1* , respectively, be the results of the first rewriting steps from *s* to *u* and to *v* .

By local confluence *t1* exists and hence, by the induction hypothesis, *t2* and *t3* exist. (See diagram on 14dii.)
Again by the induction hypothesis applied to *t2* and *t3* , *t* exists. Hence confluence for *s* is shown.

The Base Case is when *s* doesn't rewrite at all. Clearly, *s* satisfies local confluence.

Summary of Slides 14

1. A rewrite rule is an ordered equation used in paramodulation in one direction only, from left to right. Variables on the rhs must also occur on the lhs.

2. A rewrite rule r=>s can be used to rewrite a term e[t], by matching t with rθ and then replacing it by sθ. Note no substitutions are applied to t. If r and t are unified, so a substitution is made also to t, then narrowing has occurred.

3. A term may often be rewritten in more than one way using rules in a rewrite system R. R is called *canonica*l if, whatever rewrites are applied to a term t, there is only one outcome (i.e the rewrite system is confluent and terminating).

4. A rewrite System is called *terminating* if there is no infinite sequence of rewrites for any term in the language.

5. A rewrite system is *confluent* if, whenever t rewrites to t1 and t2, then t1 and t2 rewrite to a common term s.

6. A rewrite system is *Church Rosser* if, whenever s=t (modulo rewrites taken as ordinary equations), then s and t rewrite to a common term.

7. At the heart of the Knuth Bendix procedure is the aim to make a rewrite system confluent.

8. The main operation in the Knuth Bendix procedure is the formation of critical pairs. All terms s that can be rewritten in 2 or more ways can be captured by superposition, in which the left hand sides of 2 rewrite rules (say rule 1 and rule 2) are matched, or overlapped. The resulting term is rewritten as far as possible starting in two different ways, first using rule 1 and then any of the other rules, and then using rule 2 and any of the other rules.

If the results are different, say s1 and s2, then s1 and s2 are called a critical pair.

9. The Knuth Bendix method relies on the fact that local confluence + termination imply confluence. A system is *locally confluent* if, whenever s rewrites to 2 different terms s1 and s2 in one step, then s1 and s2 rewrite to a common term.

Note the difference with confluence, where s is assumed to rewrite to s1 and s2 in an arbitrary number of steps. Thus local confluence is weaker, hence the extra condition on termination is required in the Knuth Bendix procedure.

10. A confluent and terminating system can be used to show s=*t modulo a rewrite system: if s and t (eventually) rewrite to the same term then s=*t, and if s and t (eventually) rewrite to different terms then ¬(s=*t).