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 - 2013

Term Rewriting Systems

14ai

- All sentences are unit equations (∀ is implicit).
- Problem is to show that ground terms t1 and t2 are equal given equations E. 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.

EXAMPLES of rewriting 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 given 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)

In these examples bindings are 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: ground terms rewrite into ground terms

- 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) =>(4) (-1+-1)+1 =>(6) -1+(-1+1) =>(5) -1+0
0+((-1+-1)+1) =>(6) 0+(-1+(-1+1)) =>(5) 0+(-1+0) =>(4) -1+0

But sometimes different orders may yield different results:

(--1+-1)+1 =>(5) 0+1 =>(4) 1
(--1+-1)+1 =>(6) --1+(-1+1) =>(5) --1+0

The aim of the Knuth Bendix Procedure is to eliminate this second effect

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.

With the restrictions (i), (ii) and (iii), the proofs can be written down in a simpler way, when they are called *rewrite proofs* and the steps are called *rewriting steps*. If requirement (iii) is relaxed, so that the term being paramodulated into may be instantiated by the step, then the process is called *narrowing*. (See slides 17.)

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 proved. For example, consider a=b and a=c, which we know should entail b=c. However, if we are allowed only to substitute for a (ie to use the equations as rules a=>b and a=>c), 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 problem, the rewriting equations should satisfy the *Church-Rosser* property, or, equivalently, *confluence*. The Church-Rosser property guarantees that if two terms *s* and *t* can be shown to be equal (eg by refuting ¬(s=t) 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. ¬(b=c) can be refuted by paramodulating with a=b and a=c to give ¬(a=a) and then resolving with x=x.

14aiii

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

Rewriting and Paramodulation

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 a common term r, deriving ¬(r=r) and then resolve with x=x. i.e. ¬(s=t) + equations ==>*[ ]

We'll write s =*t to denote that ¬(s=t) ==>* [ ] by paramodulation

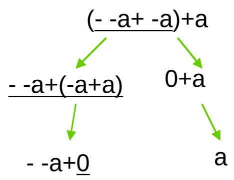
Example: (4) 0+x => x (5) -x + x => 0 (6) (x+y)+z => x+ (y+z)

If (4) - (6) are treated as equations, from ¬(--a+0=a) derive [ ] by paramodulation:
¬(--a+0=a) ==>(5) ¬(- -a+(-x1+x1) = a) (replace 0)
==>(6) ¬((- -a+ -x1)+x1 = a)
==>(5) ¬(0+a = a) (instantiate x1==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



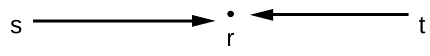
Summary of Rewriting So Far

Given a set of rewrite rules:

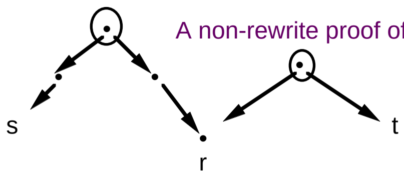
- To show s=*t by rewriting:
either: rewrite s into t, (s =>*t), or rewrite t into s, (t =>* s)
or rewrite s into r and rewrite t into r (s =>* r and t =>* r)
– all steps in the direction of => and no binding to variables in s or t

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

A rewrite proof of s=*t



Uses directions of =>



A non-rewrite proof of s=* t

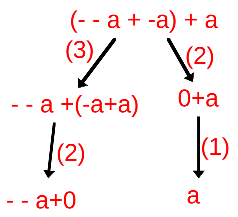
Does not always use directions of =>

Completion – Informal Overview (Specific case) (ppt)

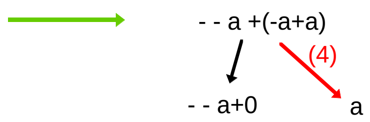
Example: Want to show: --a + 0 =* a but using all rules in => direction

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

A non-rewrite proof

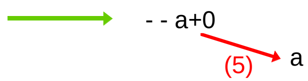


Suppose could derive (4) -x1+(x1+z) => z



Still not a rewrite proof

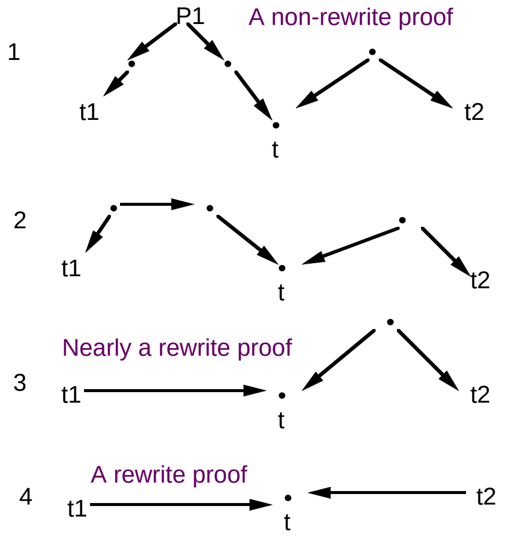
Suppose could derive (5) - -z+0 => z



A rewrite proof

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

Completion – Informal Overview (General case)
14bii



Want to derive some new rewrite rules to avoid peaks. 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, such as P1, which can be rewritten in more than one way, by (say) rules (a) and (b).

Then a new rewrite rule can be derived 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>
14biii

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" will 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 (3) and then into --a+0 by (2), or into 0 +a by (2) and then into a by (1).

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.

In the example on Slide 14bi 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>
14biv

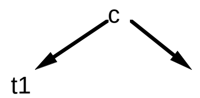
Finding critical terms is quite easy. Given two rules r1 and r2, if the LHS of r1 can be unified with the LHS of r2, or with a subterm of the LHS of r2, then the "common" instance can be rewritten by r1 and r2 into (say) r1' and r2'. By applying rewrite rules to r1' and r2', rewriting as far as possible, 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 both to a (by r1) and to b (by r2). The new rule would be (say) b=>a, which of course can be found by paramodulation too: paramodulate f(a,u)=b into f(x,x)=x to give b=a, maybe ordered as b=>a (bind u==a and x==a). This new rule b=>a is needed to show by rewriting that f(b,a) and b are equal (they both rewrite to a). This would not otherwise be possible by r1 and r2 alone, even though we can show f(b,a) =* b using r1 and r2 as equations and restricting paramodulation s.t. no bindings are made to the "into" term. The paramodulation derivation would be: ¬(f(b,a)=b) ==>(by r1 in the wrong direction) ¬(f(b,f(a,a))=b) ==>(by r2) ¬(f(b,b)=b) ==>(by r1) ¬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, and then to ¬b=b by r2, which resolves with x=x .

On the other hand, using the new rule and rewriting, f(b,a) => f(a,a) => a and b=> a, hence f(b,a) and b both rewrite to the same term "a".

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) (ppt)
14ci



If c can be rewritten in two ways it is called a *critical term*

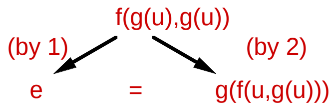
Results of rewriting t1 and t2 as far as possible are called a *critical pair*

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

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

ie a common instance of the LHSs of two rules can be rewritten in more than one way.

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)

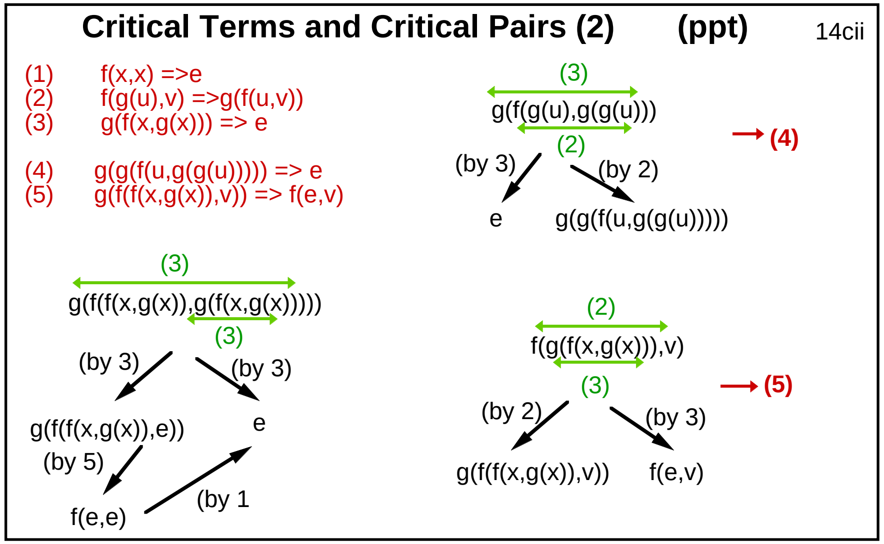


Obtain a new rule:

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

Exercise: find superpositions using (3):
can either overlap LHSs of 2 rules, or LHS of 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) *(Answer on 14cii)*



Superposition: Some Examples 14ciii

Example 1. (Slide 14ci/14cii) The 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 (4) 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 (5) g(f(f(x,g(x)),v))=>f(e,v). Rule (3) can also 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 then by (5) to f(e,e) and by (1) to e and also by (3) to e, giving no new rule.

Note also that g(f(u,g(u)) on slide 14ci cannot be further rewritten by (1) or (2) as to do so would require making a binding to u.

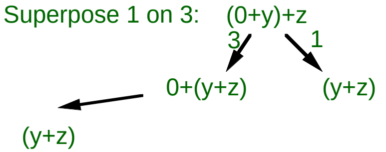
Example 2. (Slides 14civ-cvi) Applying superposition to the rules on 14civ, the first attempt at a new rule yields nothing. Although a term that matches (0+y)+z can be rewritten in two different ways, the result is the same eventually. But the second attempt, using rules (2) and (3), in which (x+y) in (3) is matched with -x1+x1 from (2), gives the 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 14cv). This has a smaller peak than before (and has a new critical term). The last step superposes (2) onto (4) giving new rule (5), which allows - -a+0 to be rewritten directly into "a".

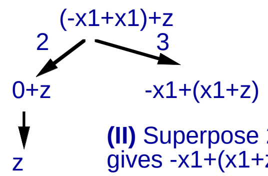
If the example on 14civ is continued, after some more superpositions it will eventually terminate, there being no new rules produced. But the example on 14cii does not terminate - there are always new (and more and more complex) rules that can be derived.

Additional Example: **Want to show:** --a + 0 => a 14civ

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

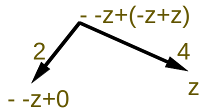
(I) Superpose 1 on 3:





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

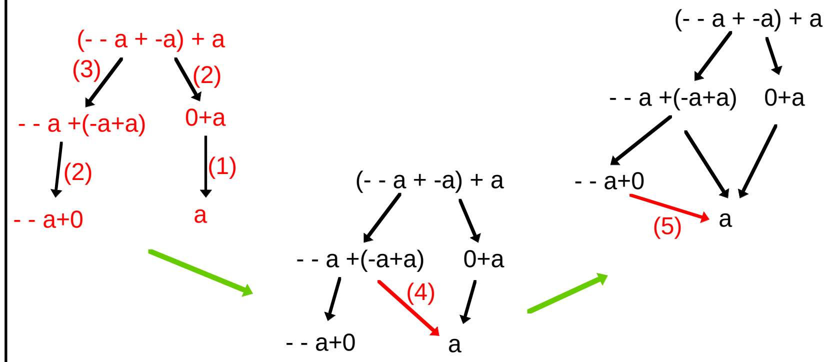
(III) Superpose 2 on 4:
gives - - z+0 => z (5)



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

Example: **Want to show:** --a + 0 => a 14cv

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



You can see how rule (4) allows to remove the peak at (--a+-a)+a,
and rule (5) allows to remove the peak at --a+(-a+a)

<ins>**Exercise:**</ins> 14cvi

Using the rules (1) to (5) from 14civ (repeated here), find some more rules that will allow to rewrite --a into a.

(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

<ins>**Hints:**</ins>

Try (3) + (5) to give a further new rule (6) --z+w =>z+w
and then use (5) and (6) to derive z1+0=>z1
and then possibly (2) and (6) to obtain z+-z=>0
or (2) + (4) ....

Example (see ppt): (1) 0+y =>y (2) s(x) +y => s(x+y) 14cvii

No possibilities here for overlapping LHSs except overlapping on a variable, which only ever leads to equations of the form t1=t1, so no extra rules.

eg, try overlapping 0+y1 on x in (2). Effect is to bind x==0+y1

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

Can show that the only terms rewritable in two ways which need to be considered are critical terms formed by
overlapping LHS(rule1) onto (non-variable subterm of) LHS(rule2)

(see optional part of slides 16 for a justification)

That is, there is no need to overlap LHS(rule1) onto a variable in LHS(rule2)

Superposition and Paramodulation (Non-examinable)

We saw already that rewriting is a restricted form of paramodulation

Superposition and forming critical pairs is also paramodulation:
(but now the "to" and "from" terms are the LHS of equations only)

(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)))
leading to e = g(f(u,g(u))) by paramodulation.

Generally: if given
L1 = R1 and L2[L3] = R2 (meaning L3 occurs in context L2) and L1θ = L3θ
then superposition 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)))

PROPERTIES OF REWRITE SYSTEMS (1)

- 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 (2)

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 implies (1): s =* u and and t =>* u implies (2): t =*u;
Therefore, by one or more paramodulation steps
¬(s=z) ==>* ¬(u=z) (for any z) by (1), and
¬(v=t)==>* ¬(v=u) (for any v) by (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 derive ¬(u=u),
and then use EQAX1 and resolution.

PROPERTIES OF REWRITE SYSTEMS (3)

- *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 in Optional Material)

(**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* .

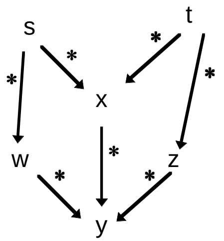
(**Fact C**) If R is locally confluent and terminating then R is confluent.

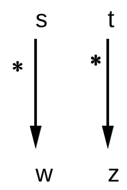
USING A REWRITE SYSTEM to SHOW s=t
14div

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)



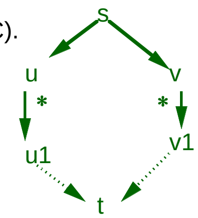


If w and z are identical then s =*t

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

Basis of the KNUTH-BENDIX procedure
14dv

- 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.

Summary of Slides 14
14ei

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.

3. A term may often be rewritten in more than one way using rules in a rewrite system R. R is called *canonical* 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).
14eii

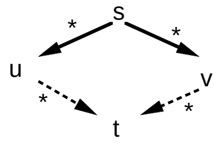
START of OPTIONAL MATERIAL (SLIDES 14)

Proofs of Facts A, B and C on Slide14diii

PROOF OF FACT B:

Confluent and terminating implies unique normal forms.

Suppose there were two different normal forms for s,   namely u and v, u≠v.



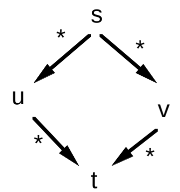
By confluence u and v rewrite to a common term, which contradicts irreducibility. Termination ensures s does not rewrite for ever (so u,v exist).

Proof of FACT A

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].

Proof of FACT A (continued)

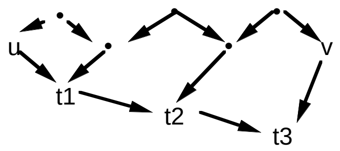
Confluence → Church-Rosser

Suppose u =* v.

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

Base– P(0): 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.



Ind. Step – let n>0 and assume as IH that P(n-1).

We show P(n):   Suppose confluence and a rewrite proof using n peaks.

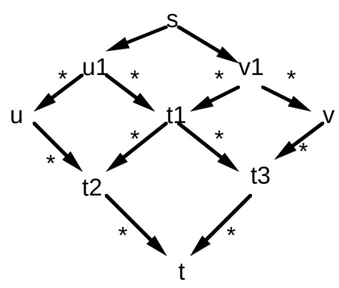
Then t1 exists by confluence and t1=* v ; there are n-1 peaks in the proof to show t1=* v; hence (by IH) ∃t3 [t1 =>*t3 and v =>*t3].

Since u =>*t1, ∃t3 [u =>*t3 and v =>*t3] and so P(n) holds.

PROOF OF FACT C (BUNDY):

Local Confluence + Termination → Confluence

- Assume Local confluence and termination.
- Termination ensures 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 14f:

In the proof of Fact A, the induction proof allows to conclude that P(n) holds for every $n \geq 0$. 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 $n \geq 0$ 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 14eiii.)
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.