AUTOMATED REASONING

SLIDES 16:

KNUTH BENDIX COMPLETION
Basic steps of Knuth Bendix completion
Aspects of Critical Pair Formation
Knuth Bendix Procedure
Outline of Correctness (Non-examinable)

KB - AR - 2012

Knuth-Bendix Completion Procedure (Rules 1)

The KB procedure essentially consists of 3 steps:

- orient equations to form directed rewrite rules
- form critical pairs and hence new equations
- use the rewrite rules to rewrite terms (and so make them smaller)

These steps can be taken in various combinations.
eg we used all three in our earlier examples of finding new rules in Slides 14.
There are also other steps useful to keep the final rule set streamlined.

In what follows, R are the rewrite rules and A are equations not yet orientated.

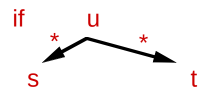
orient equation

$$\frac{A \cup \{s=t\}\ ;\ R}{A\ ;\ R \cup \{s \Rightarrow t\}} \text{ (Oeq)}$$

(or $A\ ;\ R \cup \{t \Rightarrow s\}$)

find critical pairs

$$\frac{A\ ;\ R}{A \cup \{s=t\}\ ;\ R} \text{ (CP)}$$



Knuth-Bendix Completion Procedure (Rules 2)

Use of various kinds of Normalisation is implicit in finding critical pairs
(recall that in (CP) u is rewritten as far as possible into terms s and t
effectively using normalise rule (Neq))

normalise equation

$$\frac{A \cup \{s = t\}\ ;\ R}{A \cup \{s = u\};\ R} \quad \text{if } \{t \Rightarrow {}^{*}u\} \text{ (Neq)}$$

normalise rule

$$\frac{A\ ;\ R \cup \{s \Rightarrow t\}}{A\ ;\ R \cup \{s \Rightarrow u\}} \quad \text{if } \{t \Rightarrow^{*} u\} \text{ (Nru)}$$

Example of using (Nru) (Nru) is similar to transitivity.
given f(x)=>g(x,x) and g(x,y)=>x
then by (Nru) obtain f(x)=>x

remove useless equation

$$\frac{A \cup \{s=s\}\ ;\ R}{A;R} \text{ (Req)}$$

Knuth-Bendix Completion Procedure (Rules 3)

collapse rule

A ; R ∪ {s => t}
——————————
A∪{u = t} ; R

if {s =>* u} and first rule used is v =>w, where s **|>** v (Coll)

where s **|>** v if s, or some subterm of s, is an instance of v but not vice versa (i.e. s and v are not identical upto renaming, and s can be *rewritten* by v)

Eg1: given (i) f(x) => g(x,x), (ii) f(b)=>c, (iii) b=>a

Take s as f(b), t as c, v as b, and w as a

f(b) =>f(a)=>g(a,a).

Then by (Coll) obtain (iv) g(a,a)=c and remove (ii)

If (iv) orders as g(a,a)=> c, left with (i), (iii), (iv)

Informally, f(b)=>c is redundant, as can obtain same result with (i) (iii) and (iv)

(Coll) is very useful and applies if the critical term is identical to s in rule s=>t (see next slide for a comparison of (Coll) and (CP))

Knuth-Bendix Completion Procedure (Rules 3)

collapse rule

A ; R ∪ {s => t}
——————————
A∪{u = t} ; R

if {s =>* u} and first rule used is v =>w, where s **|>** v (Coll)

where s **|>** v if s, or some subterm of s, is an instance of v but not vice versa

Eg2: Given (i) f(y)=>g(y,y), (ii) f(f(x))=>h(x)

Take s as f(f(x)) and v as f(y); then f(x) in f(f(x)) is an instance of f(y)

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

Then (Coll) derives g(g(x,x),g(x,x)) =h(x) (iii) and can remove (ii)

But also can take s as f(f(x)) and v as f(y) such that f(f(x)) is an instance of f(y)

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

Then (Coll) (again) derives g(g(x,x),g(x,x)) =h(x) and removes (ii)

Eg3: Given (1) -0=>0 (2) 0+z=>z (3) -0+z=>z

Apply (Coll) to (3) using (1): -0+z => 0+z => z giving z=z which (Req) removes

Also, Coll removes (3) leaving just (1) and (2)

<u>Rules (Coll) and (CP) compared</u>

collapse rule

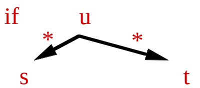
if {s =>* u} and first rule used is v =>w, where s |> v (Coll)

A ; R ∪ {s => t}
——————————
A∪{u = t} ; R

where s |> v if s, or some subterm of s, is an instance of v but not vice versa

find critical pairs

A ; R
——————
A ∪ {s=t} ; R



Consider the case when (Coll) and (CP) both apply. The condition of (Coll) means that variables in s are not bound by the superposition step. Given s=>t, s=L[vθ] for some context L, v=>w, (CP) will give L[wθ]=>t as the first step of CP, where θ applies only to variables in v and w. If (Coll) applies too, then L[wθ]=>*u. The result u=t is the same, but under (Coll) the original rule s=>t is removed, whereas under (CP) it is not. Thus in this circumstance (Coll) is better.

In case s and v are renamings of each other (or the same), applying (CP) will achieve the effect of removing either s=>t or v=>w. For example, f(x)=>h(x) and f(x)=>x, will give by (CP) the critical term f(x), which will rewrite in two ways to give h(x) and x. If the equation is ordered h(x)=>x, it can be used to normalise (Nru) f(x)=>h(x) into f(x)=>x, which is already present. The net effect is to remove f(x)=>h(x). If (Coll) is used (to collapse f(x)=>h(x)), the effect is exactly the same. If (Coll) is used to collapse f(x)=>x, the effect is first to remove this rule and add h(x)=>x and then to normalise f(x)=>h(x) to f(x)=>x. A roundabout way to get the same effect as with (CP).

Knuth-Bendix Completion Procedure (Rules 4)

remove subsumed equations

A∪ {s=t, u[sσ] = u[tσ]} ; R
——————————————
A ∪{s=t} ; R

(Sub)

Example of subsumption: a= b and h(g(a),x) = h(g(b),x)

Of course, equations or rules θ-subsumed by rules can be removed too.

Q: Can equations be used to θ-subsume <u>**rules**</u>?
Hint: consider f(x,y)=f(y,x) and f(b,a)=>f(a,b)

Knuth Bendix Procedure:

The Knuth Bendix procedure can be presented in several different ways:

(1) As a collection of inference rules that can be applied in any order to a set of equations and rewrite rules;
(2) As an imperative program;
(3) As a corresponding declarative (eg Prolog) program.

In all cases, the input to the procedure is a set of unorientated equations and, when successful, the output is a confluent set of rewrite rules. The various steps may be applied in any order, although a fixed sequence of applying the various steps can be made, as shown on the slides.

There are two unsuccessful outcomes:
(i) the procedure doesn't terminate - always another step can be applied, or
(ii) an equation is derived that cannot be orientated sensibly.
An example of such an equation is x+y = y+x - it is bound to lead to non-termination of a rewriting sequence.

In fact, both undesirable outcomes can still be put to some good use.
In the case of (i), called *divergence*, the rules obtained at a given stage may be adequate to show that the answer to the current problem (is s=*t?) is TRUE; however, an incomplete set of rules cannot be used to show the answer is FALSE.

If an equation E: l=r can't be orientated, then it can be left as an equation and used for rewriting in both directions. The only restriction is this: if an instance lσ=rσ of E is used for rewriting lσ into rσ then lσ > rσ and if used for rewriting rσ into lσ then rσ > lσ.

Knuth Bendix Algorithm (Imperative)


```
WHILE equations remain in A {
    remove equations that rewrite to x=x
       or are subsumed
    select an equation E
        and remove from A {
            normalise E;              //(Neq)
            orient E;                 //(Oeq)
            normalise RHS. of rules in R
                using and including E;   //(Nru)
            find all critical pairs C of E with R; //(CP)
            add E to R;   add C to A;
            apply (Coll) using E;
        }
    }
```

Often, (Neq), (Nru), (Coll) and (Oeq) are performed on all current equations before finding critical pairs (CP). New equations cause a new sequence of (Neq), (Oeq), (Nru) and (CP). But one at a time may be easier for a person to do.

Outcomes of Knuth Bendix Procedure:

Terminates converges to a confluent set of terminating rewrite rules.

Diverges (and never stops): the confluent set would be infinite.

Fails (and stops): cannot find a termination ordering to orient the rules.

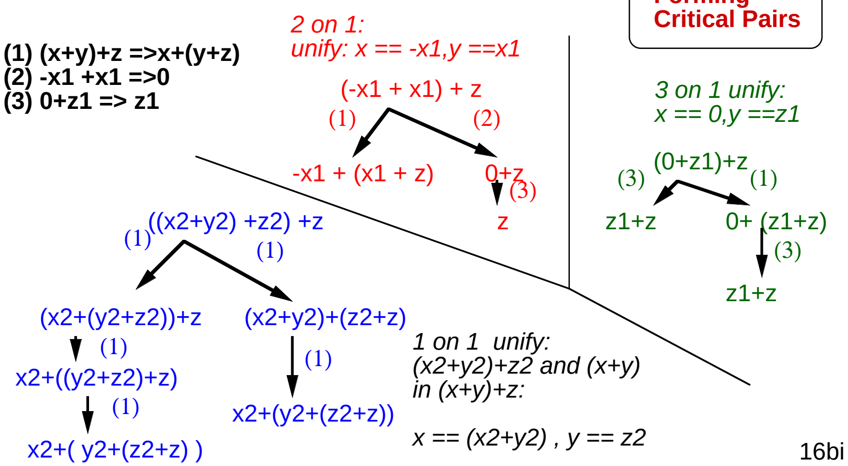
(eg x+y = y+ x causes difficulties.)

Apply the Knuth Bendix Procedure (Revision)

Finding critical pairs enables new rewrite rules to be formed, which will contribute towards confluence.

Examples of Forming Critical Pairs

(1) (x+y)+z =>x+(y+z)
(2) -x1 +x1 =>0
(3) 0+z1 => z1



Applying the Knuth Bendix Procedure (1)

16bii

(1) (x+y)+z => x+(y+z)
(2) -x1 +x1 => 0
(3) 0+z1 => z1
(4) -x1 + (x1+z) => z

There are various options next, but the useful ones are (2) on (4) and (3) on (4) giving (5) and (6).

I leave the rest as an exercise, for you to work out their derivations

(5) --x1+0 => x1
(6) -0+z => z

NOTE: There are 2 different orderings that can be used, which are shown on 16biii and 16biv

(7) --0+z => z
(8) -0 => 0

(6) and (7) can be removed using (8) by (Coll)

(9) --x1 + z => x1 + z
(10) x + 0 => x
(11) --x => x

(9) and (5) can be removed using (11) by (Coll)

In fact (1), (2), (3), (4), (8), (10), (11) is not the final confluent set.
Can also derive
(12) x+ -x => 0 (use 2 and 11)
(13) x + (-x + z) => z (use 1 and 12)
(14) - (x + y) => -y + -x (use 4 and (i), where (i) is y+ -(x+y)=>-x, from 1 and (ii), where (ii) is x + (y+ - (x+y))=>0, from 1 and 12, and (i) subsumes (ii) and (14) subsumes (i))

Ordering using lpo

16biii

(1) (x+y)+z => x+(y+z)
(2) -x1 +x1 => 0
(3) 0+z1 => z1
(4) -x1 + (x1+z) => z

Ordering is lpo: ranking of operators is "-" $>_1$ "+" $>_1$ "0"

2-7, and 10,11,12, 13 are fairly clearly ordered left to right
8 is ordered left to right as -0 > 0 by ranking
1 is ordered left to right:
{(x+y),z} $\geq^*_{lpo}$ {x,(y+z)} since x+y $>_{lpo}$ x, and
(x+y)+z $\geq_{lpo}$ (y+z), since (x+y) $>_{lpo}$ y and (x+y)+z $>_{lpo}$ z

(5) --x1+0 => x1
(6) -0+z => z

9 is ordered left to right
{--x1,z} $\geq^*_{lpo}$ {x1,z} since --x1 $>_{lpo}$ x1, and
--x1+z $\geq_{lpo}$ z (case 1 of lpo)

(7) --0+z => z
(8) -0 => 0

14 is ordered left to right by ranking, and
-(x+y) $>_{lpo}$ -y and -(x+y) $>_{lpo}$ -x (case 3 of lpo)
(i) is ordered by case 1 of lpo
(ii) is ordered by ranking

(9) --x1 + z => x1 + z
(10) x + 0 => x
(11) --x => x

(12) x+ -x => 0 (use 2 and 11)
(13) x + (-x + z) => z (use 1 and 12)
(14) - (x + y) => -y + -x (use 4 and (i), where (i) is y+ -(x+y)=>-x from 1 and (ii), where (ii) is x + (y+ - (x+y))=>0 from 1 and 12
(i) subsumes (ii) and (14) subsumes (i)

Ordering using kbo

16biv

(1) (x+y)+z => x+(y+z)
(2) -x1 +x1 => 0
(3) 0+z1 => z1
(4) -x1 + (x1+z) => z

Ordering is kbo: basic order on ground terms is to sum the weights of terms, where wt(-)=0, wt(+)=wt(0)=1
ranking of operators is "-" $>_1$ "+" $>_1$ 0

1-7,10,12,13, (i) and (ii) are clearly ordered left to right
8 is ordered left to right as -0 > 0 by ranking
9 is ordered left to right:
sum of wt(left) = sum of wt(right) (for any x1 and z)
left $\geq^*_{kbo}$ right since --x > x for every x (if x is u+v or 0, this is easy; if x is -u, show --u>u; use induction: since the term structure is decreasing, will reduce to previous cases of 0 or +.

(5) --x1+0 => x1
(6) -0+z => z

(7) --0+z => z
(8) -0 => 0

(9) --x1 + z => x1 + z
(10) x + 0 => x
(11) --x => x

For 11, use similar argument as for 9.
14 is ordered left to right:
sum wts(left)=sum wts(right) for any x and y
and - $>_1$ +

(12) x+ -x => 0 (use 2 and 11)
(13) x + (-x + z) => z (use 1 and 12)
(14) - (x + y) => -y + -x (use 4 and (i), where (i) is y+ -(x+y)=>-x from 1 and (ii), where (ii) is x + (y+ - (x+y))=>0 from 1 and 12
(i) subsumes (ii) and (14) subsumes (i)

About forming Critical Pairs (apart from slides 16ciii, 16cv, slides 16c are optional):

A critical pair may occur when a term (the *critical term*) rewrites in two different ways. If the two resulting terms are different and cannot be further rewritten to the same term, the eventually resulting different terms are called the *critical pair*. On Slide 16bi there are 3 examples. The first yields the critical pair (z , -x1+(x1+z)) and the second and third examples do not yield a critical pair. Critical terms arise because the LHSs of two rewrite rules apply to a term ***s*** in two different ways. (It may be just one rule involved in different places.) This can happen in essentially three ways.

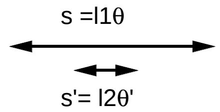
(a) One way is when the parts of ***s*** being rewritten do not overlap. This way will not yield a critical pair (see 16civ, case 1): if a term ***s*** can be rewritten in two ways, but by rewriting two non-overlapping terms, then this will not be because the LHSs of the rules overlap. The two steps can be applied separately. If θ is the substitution applied to rule 1 and σ the substitution applied to rule 2, then ***s*** can be written as s[LHS1θ, LHS2σ], which rewrites into s[RHS1θ, LHS2σ] or s[LHS1θ, RHS2σ] and then into s[RHS1θ, RHS2σ].

(b) Otherwise, the LHSs themselves must "overlap" or can be superposed. That is, either LHS1 and LHS2 unify, or LHS1 unifies with a subterm of LHS2 (or vice versa). There are two different ways in which this can occur, only one of which is useful. If the LHSs of the two rules overlap on a variable subterm ***x*** – ie LHS1 unifies with a variable x in LHS2 with substitution θ, then the critical term is the instance LHS2θ of LHS2; although LHS2θ rewrites to 2 different terms, these can always be rewritten to a common term: LHS2θ, rewrites into RHS2θ (by rule 2) and also into LHS2θ' by rule 1, where θ' is the substitution x==RHS1. Both of these rewrite into RHS2θ', the first by rule 1 and the second by rule 2. You should draw a diagram to convince yourself that this is so. Case 2 on 16civ illustrates this. Note that if ***x*** does not occur in RHS2 then RHS2θ is the same as RHS2θ'.

(Continued on Slide 16cii.)

(c) ***s*** rewrites in 2 different ways by rules that overlap on a non-variable sub-term of ***s***. In this case the LHS of the two rules also overlap on a non-variable sub-term (see below). (This is illustrated in Case 3 on 16bviii).

s matches l1 (l1 => r1)
s' in s[s'] matches l2 (l2 => r2)



s=l1θ, s' = l2θ' and s' is not in a variable position in l1θ

i.e. l2 and a subterm of l1 have a common instance.
Hence ∃σ: l2σ is a subterm of l1σ.

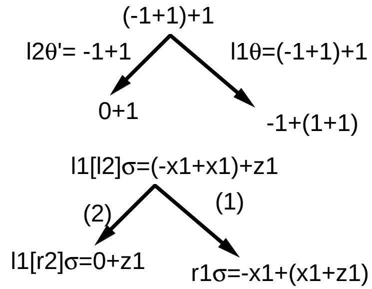
So l1[l2σ]σ rewrites (by 1) to r1σ and (by 2) to l1[r2σ]σ

Exercise: identify θ,θ',σ in the following:

Use (x+y)+z => x+(y+z) (1)
and -x1+x1 =>0 (2)

s=(-1+1)+1 matches with l1 and s' = (-1+1)
s rewrites to 0+1 (by 2)
and to -1+(1+1) (by 1)

θ' is {x1==1}, θ is{x==-1, y==1, z==1}
and σ is {x==-x1, y==x1}



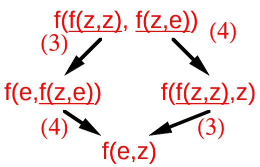
Are all critical pairs found by (CP)?

When might a term be rewritten in more than one way?

Case 1: non-overlapping occurrences of LHSs of two instances of a rule:

can rewrite occurrences in turn and will write to a common term.

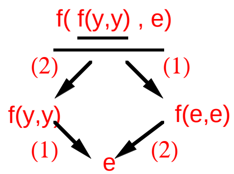
3. f(y,y) => e
4. f(x,e) => x



Case 2: Rules apply such that they *overlap on a variable subterm*:

will also rewrite to a common term.

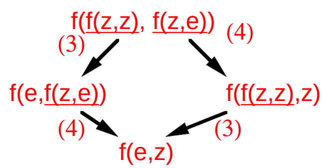
1. f(y,y) => e
2. f(x,e) => x



Formation of critical pairs - possibilities for non-confluence

Case 1: non-overlapping occurrences: can rewrite occurrences in turn and can write to a common term.

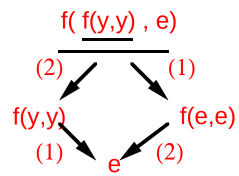
3. f(y,y) => e 4. f(x,e) => x



s = f(f(z,z), f(z,e))
can be rewritten by 3 and 4:
θ (for 3) = {y==z}; σ (for 4) = {x==z}
s = f(LHS3θ, LHS4σ)
=> (by 3) f(RHS3θ, LHS4σ)
or (by 4) f(LHS3θ, RHS4σ)
=> f(RHS3θ, RHS4σ) (by 4, or by 3)

Case 2: Rules apply such that they *overlap on a variable subterm* - can also rewrite to a common term.

1. f(y,y) => e
2. f(x,e) => x



s = f(f(y,y), e)
can be rewritten by 1 and 2
(they overlap on variable x in f(x,e))
θ (for 2) = {x==f(y,y)}
s = LHS2θ => (by 2) RHS2θ
or (by 1) LHS2θ' (θ' = {x==e})
=> RHS2θ' (by 1, or by 2)

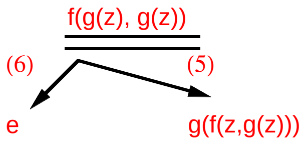
Formation of critical pairs - possibilities for non-confluence continued

Case 3: rules apply such that they *overlap* on a **non-variable** subterm

• Only need to check occurrences of Case 3 for possible non-confluence.

• All necssary critical pairs can be found by unifying the LHS of rules with *non-variable* subterms of other LHS and rewriting <u>as far as possible</u> (first using overlapping rules, then maybe other rules)

5. f(g(x),y1) =>g(f(x,y1))
6. f(y2,y2) => e



Optional part of slide Using notation of 16cii,
l1[l2σ]σ = f(g(z) , g(z)) and l2σ = f(g(z) , g(z))
σ = {x==z, y1==g(z), y2==g(z)}

l1[l2σ]σ rewrites (by 5) to r1σ = g(f(z,g(z))) and (by 6) to l1[r2σ]σ = l1[e] = e

Correctness of Knuth Bendix Rules (Bachmair) (For interest only)

- The inference rule approach allows logic and control to be separated
- Invariant properties can be found that imply confluence on termination.

- A derivation using the inference rules has the form: (A0,R0) , (A1,R1) , ….
- Because of subsumption and Collapse some rules may not remain forever.
- A *persistent rule* is one that occurs in Ri and remains in Rj, ∀j≥i.
- R_∞ = {persistent rules} [formally = $\cup(i\geq 0)$ $(\cap(j\geq i) Rj)$]
- Aim is for R_∞ to be canonical -
  any equation valid in (A0,R0) = (A0,{}) has a rewrite proof in R_∞.

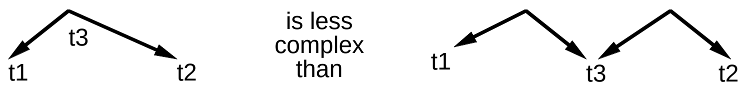
- We define the relation ⇔A∪R by (u,v) ∈ ⇔A∪R iff (A,R) |= u=v
  ⇔A∪R is obtained by using A and R together and treating R as equations.
  ⇔A∪R is an equivalence relation on terms;
  (Exercise: Show ⇔A∪R <u>is</u> an equivalence relation)

- Invariant of procedure: For each i, (Ai, Ri) and (Ai+1, Ri+1) are related:
  $\Leftrightarrow Ai\cup Ri = \Leftrightarrow A_{i+1} \cup R_{i+1}$

- Ensures that (A0,{}) |= u=v iff ({}, R_∞) |= u=v
  i.e. no proofs (possibly with peaks) have been lost or gained.

• <u>**Idea of the proof**</u>:
to show R_∞ is confluent must show that rewrite proofs using derivation (Aj ,Rj) are "less complex" than those using (Ai,Ri), i<j

eg



- A *non-rewrite-only* proof uses equations as well as rewrite rules
- Making an equation a rewrite rule may mean it is used 'backwards' in a proof
- Generating critical pairs allows for new rewrite rules to be added which will smooth out a proof (ie remove a peak or two)
- *Fairness* is required so that equations cannot be ignored for ever
- All critical pairs will eventually be formed
- Any proof eventually becomes a rewrite proof as proofs decrease in complexity as rewrite rules are formed from equations.

Problems:

- The KB algorithm can fail because a selected equation cannot be oriented or because R_∞ is not finite.
- It may be possible to try a different ordering,
or may still be able to use rewrites generated so far to show s=t .

Summary of Slides 16

1. The Knuth Bendix procedure can be described using an imperative or declarative program, or by a set of inference rules. The main operations are orient, find critical pairs and normalise.

2. It is only necessary to search for overlapping of the LHS of rules in order to find all possible terms that could lead to a critical pair. Overlapping onto a variable is not necessary.

3. Normalising is the operation that applies rewrite rules to other rules or equations. It can be applied to rewrite rules (the RHS), or to equations (either side).

4. The operation of removing useless equations (they rewrite to s=s), or subsumed equations (they are implied by other equations or rules) is helpful.

5. The Knuth Bendix procudeure can terminate, diverge (non-terminating), or fail (an equation can't be oriented – eg x+y=y+x).

6. The Knuth Bendix procedure is correct - when it terminates the final set of rules is confluent and terminating. The proof method shows that the procedure does not remove any proofs, but each proof becomes more like a rewrite proof as each new rewrite rule is generated.