AUTOMATED REASONING

SLIDES 15:

TERMINATION OF REWRITE SYSTEMS
Properties for termination
Stable orderings
Ordering multi-sets
Useful partial orders:
kbo, lpo, rpo
The ERUDIO tool (for interest only)

KB - AR - 2012

Termination of Rewrite Systems (I)

15ai

(based on Dershowitz, JSC, 3, 87)

Some background on partial orderings on terms relevant for termination

- A partial order relation ">" is a transitive and irreflexive relation
  i.e. $\forall x,y,z[x>y$ and $y>z \rightarrow x>z]$, $\forall x.\neg(x>x)$
- It is also non-symmetric - i.e. $\forall x,y[x>y \rightarrow \neg(y>x)]$ (derivable)
  A partial order > is usually written in infix notation - x>y rather than >(x,y)
- Most relations we consider are total on ground terms (ie x>y or y>x)
- A partial order > is well founded on a set of terms S
  if there is no infinite descending chain t1 > t1 > ... > ti > ...

eg1 S is the set of integers >-10 and > is the ordinary "greater-than" relation
eg2 Any relation on a finite set S

Note: *s<t is equivalent to t>s;* *s ≥t means s>t or s=t;*
if > is a partial order
then ≥ is reflexive ($\forall x.x\geq x$) *and antisymmetric* ($\forall x,y[x\geq y$ and $y\geq x \rightarrow x=y]$)

Exercise: s > t if #(s) > #(t) – for ground term s #(s) is the number of symbols (constants or functions) in s

Is > a partial order (is it transitive? irreflexive? non-symmetric?
Is > a total order? Is it well-founded?

Termination of Rewrite Systems (2)

15aii

Some basic properties relevant for termination

- *Monotonicity:* if t>u then f(…t…) > f(…u…).

i.e. reducing a subterm reduces any superterm of it.
e.g. would like to be sure that if a<b then g(h(a))<g(h(b)).

- *Simplification*: A monotonic ordering > is called a *simplification* ordering if for all ground terms t, f(… t …) > t.

Most standard orderings used to prove termination are simplification orderings.

- *Stability*: if t>u then tσ > uσ for all ground substitutions σ.

i.e. enables > to be applied between non-ground terms.

Example: s > t if #(s) > #(t) – for ground term s #(s) is the number of symbols (constants or functions) in s

Monotonic? – yes: if s has more symbols than t, then
f(...,s,...) has more symbols than f(...,t,...)

Simplification? – yes: f(...,s,...) has more symbols than s

Stable? – depends:
eg we can say f(x,x)>g(x) – whatever ground term x is #f(x,x)>#g(x)
but not f(x,y) > h(x,x) – if x is bound to a longer term than y, #f(x,y)<#h(x,x)

Useful Facts about a rewrite ruleset R

15aiii

(**Fact D**) If < is well-founded on the set of ground terms, then R will be terminating if for ground terms s and t, if s =>*t then s>t.
(Not very useful actually, as it is hard to consider all pairs of ground terms)

(**Fact E**) If there exists a monotonic and well-founded ordering > such that lσ > rσ for each rule and each ground substitution σ, then R will be terminating

Example: (1) f(e,x) => x (2) f(i(x),x) => e s > t if #(s) ># (t)

We'll use Fact E) and count terms:

For each ground substitution for x, clearly LHS(2) > RHS(2)
Also LHS(1)>RHS(1): #f(e,x) =2+#x > #x (#x means number symbols in x)

The order is monotonic: if s<t then f(s,z)<f(t,z), f(z,s)<f(z,t), i(s)<i(t) (for any z)
The order is well-founded as #s is ≥0.

For you to try:
g(g(f(x))) => f(g(x)) Again can count terms.

More Examples (solutions on 15av)

(**Fact F**) Even if > is not monotonic, Fact E can be relaxed:

R will be terminating if > is well-founded, $l\sigma > r\sigma$ for each rule and each ground substitution σ, and s =>*t and s>t implies f(...s..)>f(...t...) – i.e. R is monotonic at least on terms that rewrite to each other.

*2. (**for you to try**)*: f(f(x)) => f(g(f(x))) ?
(Try ffgfggfa => ? - does it terminate?)

Use Fact F here.

(i) Find a well-founded order (not necessarily monotonic) and show LHS>RHS for all x

(ii) Show the order is monotonic on terms that rewrite to each other.

Hint: consider counting adjacent occurrences of f

3. (***for you as well but more esoteric***): f(g(x)) => g(g(f(x)))

(Try fgfggfa =>? does it terminate?)

Again use Fact F.

Hint: What happens to the occurrences of f to the left of an occurrence of g?

<u>Solutions to Examples on 15aiv</u>

f(f(x)) => f(g(f(x))).

Count #pairs of adjacent fs. It is clear that for any x the number of adjacent pairs of f is reduced by 1 after applying the rule. As counts are ≥0 the ordering is well-founded.

The ordering is not monotonic though: g(f(f(a))) has 1 pair of adjacent f and f(a) has none. So g(f(f(a)))>f(a). But f(g(f(f(a)))) is not > f(f(a)) as both have 1 pair of adjacent f. However, g(f(f(a))) does not rewrite to f(a). If this is a general property, then we can apply FACT F:

Comparing f(s) and f(t) it's clear this property **is** maintained (and also for g(s) and g(t)). Assume s=>t, then s has the form (<...>ff<....>) and t has the form (<..>fgf<...>). If the number of pairs of f in s > number of pairs in t, then it's easy to argue, by considering cases and counting, that the same holds for f(s) (i.e. f (<...>ff<....>)) and f(t) (i.e. f(<...>fgf<....>)).

---

f(g(x)) => g(g(f(x))).

Count #gs to right of each f. Note #fs remains fixed for rewriting a given term – let #fs = n.
Let (ai) be the number of gs to right of i'th f from the left.

Define (a1, a2, …, an) > (b1, b2, … bn) iff ai > bi and ∀j : i+1 ≤ j ≤n → (aj = bj)

ie i is first position from right at which ai ≠ bi

e.g. fgfggfa => fgggfgfa and the counts are (3,2,0) and (4,1,0); Notice (3,2,0)>(4,1,0)

Check > is well-founded and LHS>RHS for all substitutions of x

<u>Well founded</u>: minimal counts =(0...0) (n × 0 for n fs). eg count(g...gfffa)=(0,0,0)

Suppose f(g(x)) has k occurrences of f with count =(c_k,...,c1).

Then g(g(f(x))) has count = (c_k-1,...,c2,c1) which is <(c_k,...,c1).

Also, check that if s=>*t and s>t then f(s)>f(t) and g(s)>g(t) (do this in a similar way to above – notice g(s) has the same count value as s). Then use Fact F.

<u>Well-founded Ordering:</u>

An order < is a well-founded ordering on a set of terms if there is no infinite descending sequence of terms s0>s1>s2>... . eg < is well-founded on {integers>k} for any particular choice of k, but not on the set of integers. For our purposes we assume the s_i are derived by rewriting: s.t. s0 =>s1, ..., s_i=>s_{i+1}

<u>Proof of Fact D</u>:
Let < be well-founded and s=>t imply s>t for all terms s and t. (*). Suppose first that s0 => s1 => … sn ... is a non-terminating, ground rewrite sequence using R, then, by (*), s0>s1>…>sn> ... . But as > is well-founded the sequence cannot continue forever. So the original rewrites cannot do so either. This is a contradiction, so the original assumption is false.

For the general case, notice that no variables other than those in s0 may appear in any si. Suppose s0 is not ground and there is a non-terminating sequence s0 => s1 => … sn ... . Consider some ground instance s0θ of s0 and hence of {si} ({siθ}). It is still the case that s0θ =>s1θ => … => and hence s0θ > s1θ > … > and this sequence must terminate at some skθ as < is well-founded. Hence skθ does not rewrite to sk+1θ. But then sk could not rewrite to sk+1 either, a contradiction. So the original rewrite sequence must terminate.

<u>Proof of Fact E</u>:
Let < be a well-founded monotonic order and $l\sigma > r\sigma$ for each rule and each ground substitution σ. Suppose that s0 => s1 => … sn ... is a non-terminating ground rewrite sequence using R, then, by assumption s0>s1>…>sn> ... (since each rewrite uses a rule and < is monotonic). But as > is well-founded the sequence cannot continue forever, so the original rewrites cannot do so either. This is a contradiction, so the original assumption is false.

In case the rewrite sequence includes variables, instantiate to obtain a non-terminating ground rewrite sequence and use the above.

A little bit more notation (Dershowitz):

15bi

Recall:
A *standard partial order* < is irreflexive and transitive (and s<t implies ¬(t<s) WHY?)

The relation ≤ defined by s ≤ t iff s < t or s = t, where < is a standard partial order, is reflexive, transitive and anti-symmetric (i.e. s≤t and t≤s implies s=t)

Sometimes we want to relax anti-symmetry so that R(s,t) and R(t,s) but s≠t.

A *quasi-partial order* <≈ is reflexive and transitive but need not be anti-symmetric. i.e. if s <≈ t and t <≈ s, then we say s ≈ t

For a quasi-order <≈, we define s < t iff s <≈ t and not (t <≈ s).

There are some famous quasi-orders: ≤kbo and ≤rpo etc. defined on slides 15ci - 15civ, which are also simplification orderings.

To show termination of R using a simplification quasi-order <≈ such as ≤kbo or ≤rpo show each rule in R satisfies lσ > rσ for all substitutions σ.

Quasi-orderings can be simplification orderings, monotonic, stable, etc.
The definitions of those things are adjusted by using the quasi-order <≈ in place of the standard partial order <.

Knuth - Bendix ordering (kbo) (ppt)

15ci

s= (f(s1 … sm)) $\geq_{kbo}$ t (= g(t1 … tn))
- if s> t (where >≈ is a simplification quasi-ordering on ground terms) (definitely >)
- or s ≈ t and f $>_1$ g ($>_1$ applies to functors here) (definitely >)
- or s ≈ t, f = g and (s1 … sm) ≥* (t1 … tn)

≥* is the lexicographic ordering induced by $\geq_{kbo}$

To use kbo to show termination of R:
show each rule in R satisfies l > r for all substitutions σ.

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

s <≈ t
if # occurrences of +/- in s ≤
occurrences of +/- in t;

1. is ok since #+/- in LHS > #+/- in RHS.
2. is ok since #+/- in LHS≥2 > #+/- in RHS = 0.
3. is ok since #+/- in LHS = #+/- in RHS, both terms have outer +, and ((x+y),z) $\geq^*_{kbo}$ (x,(y+z)) as #+/- in x+y > #+/- in x;
(i.e. lexicographic order based on kbo)

Also, <≈ is a simplification ordering: eg x<y==>-x < -y and -x > x.

Recursive Path Ordering (rpo)

15cii

s (=f(s1 … sm)) $\geq_{rpo}$ t (=g(t1 … tn))
- if si $\geq_{rpo}$ t for some i = 1… m (definitely >)
- or f $>_1$ g and s $>_{rpo}$ tj for all j =1…n
  ($>_1$ orders functors) (definitely >)
- or f = g and {s1 … sm} >> {t1 … tn} (>> is a multi-set ordering)

Multi-set Ordering (ppt)

- A *multi-set* over a set of terms E with order > is a mapping *m* from E to N.

e.g. S={3,3,4,0} = {3 -> 2, 4 -> 1, 0 -> 1} or S(3) = 2, S(4) = 1, S(0) = 1.
- If S is a multi-set then d(S) = {elements in S (as a set)}
- If e in E and not(e in d(S)) then S(e)=0
- S>>T iff ∀e:e in d(T) [S(e) ≥ T(e) ∨ ∃g [g in d(S) ∧ g > e ∧ S(g)>T(g)]]

{3,3,4,0}>>{3,3,2,2,1,1} if {4,0}>>{2,2,1,1) (remove occurrences of = elements)
{4,0}>>{2,2,1,1} if each element in {2,2,1,1} is dominated by an element in {4,0}
(Here 4 dominates all of 2,2,1,1.)
{3,3,4,0}>> {3,3,3} if {4,0} >> {3}. (4 dominates 3 so OK)
{4,3,3} >> {4,3} if {3} >> ∅. (OK)
{4,1,1} >> {4, 2} if {1,1} >> {2}. (Not OK as 1 doesn't dominate 2)

Example of using recursive path ordering (ppt)

1 $\neg\neg x \Rightarrow x$
2 $\neg(x \wedge y) \Rightarrow \neg x \vee \neg y$
3 $\neg(x \vee y) \Rightarrow \neg x \wedge \neg y$
4 $x \wedge (y \vee z) \Rightarrow (x \wedge y) \vee (x \wedge z)$
5 $(y \vee z) \wedge x \Rightarrow (y \wedge x) \vee (z \wedge x)$

- 1 $\neg\neg x \geq_{rpo} x$ {x is a subterm}
  (use n(x) for $\neg x$ if you prefer)
- 2 $\neg(x \wedge y) \geq_{rpo} \neg x \vee \neg y$ if $\neg(x \wedge y) >_{rpo} \neg x$ and $>_{rpo} \neg y$
  (choose $\neg >_1 \vee$)
  i.e. if $\{x \wedge y\} \gg \{x\}$ (and $\gg \{y\}$) which it is as x/y are subterms of $x \wedge y$
  (use a(x,y) for $x \wedge y$, and o(x,y) for $x \vee y$ if you prefer)
- 3 similar
- 4 $x \wedge (y \vee z) \geq_{rpo} (x \wedge y) \vee (x \wedge z)$ if $x \wedge (y \vee z) >_{rpo} x \wedge y$ and $>_{rpo} x \wedge z$
  (choose $\wedge >_1 \vee$)
  i.e. if $\{x, (y \vee z)\} \gg \{x,y\}$ and $\gg \{x,z\}$ which they are.
- 5 similar

Exercise: Suppose that s and t are terms, and t is a subterm of s, or of a subterm of s. When comparing s and t by rpo, explain why the first case will always hold (though it may have to be applied more than once).

Lexicographic path ordering (lpo)

s (= f(s1 … sm) $\geq_{lpo}$ t (= g(t1 … tn)) if
- si $\geq_{lpo}$ t for some i = 1 … m, or
- f $>_1$ g and s $>_{lpo}$ tj for all j = 1 … n, or
- f = g and (s1…sm) $\geq^*_{lpo}$ (t1 … tn) and s $>_{lpo}$ tj for all j = 2 … n,
  where $\geq^*_{lpo}$ is the lexicographic ordering induced by $\geq_{lpo}$

Example of using lexicographic path ordering

$(x+y)+z \geq_{lpo} x+(y+z)$: main functor = + in both cases so case 3

- $((x+y, z) \geq^*_{lpo} (x,(y+z))$ since $(x+y) \geq_{lpo} x$
  (because x is a subterm of x+y), and
- $(x+y)+z \geq_{lpo} (y+z)$: main functor = + in both, so case 3
  - $((x+y, z) \geq^*_{lpo} (y,z)$ since $(x+y) \geq_{lpo} y$, and
  - $(x+y)+z \geq_{lpo} z$ (because z is a subterm)

The three orderings kbo, rpo and lpo

The **Knuth Bendix Ordering (*kbo*)** (on 15ci) is the easiest to use. To apply it you need an order on functors that you choose and a quasi-simplification order $<\approx$ on ground terms. A standard choice for $<\approx$ is the number of symbols, but others are possible. If you use another order, you just need to show it is a simplification order (ie if x $<\approx$y then f(...x..)$<\approx$f(...y...) for all functors f, and that f(...,x,...)>x.) In Case 3 the lex ordering is a dictionary ordering based on the underlying kbo. That is, to compare two lists of terms, compare the lists as you would compare words in a dictionary.

The **Recursive Path Ordering (*rpo*)** (on 15cii) is next easiest. There are 3 cases. To show s$\geq_{rpo}$t, Case 1 checks if an argument of s$\geq_{rpo}$t. This will be true, for example, if t occurs as a subterm of s, for you can recursively apply this case until you have extracted t as a subterm of s. If Case 1 doesn't hold, then look at the outer functors of s and t. Note that the second condition requires s to be definitely greater than t. In Case 3 a multi-set ordering is used. Despite the complicated definition it is easy to check. First strike out identical terms from the two lists. Next, take each element e left in t and check there is an element left in s that is larger than e. (**Exercise:** Show this procedure satisfies the given definition of >>.)

The **Lexicographic Path Ordering (*lpo*)** (on 15civ) is similar to rpo, except for Case 3. In Case 3 you must first check the arguments of s and t are pairwise lexicographically ordered (as in kbo) and then recursively check s is definitely $>_{lpo}$ than each argument of t (other than the first). You can show that if the first lexicographic condition finds (say) argument 2 of s $>_{lpo}$ argument 2 of t, then the second condition can start at argument 3, since Case 1 would hold for s$>_{lpo}$ argument 2. For example, to compare f(x,b,y) and f(x,a,y), where b>a, Case 3 says to compare [x,b,y] and [x,a,y] lexicographically, which holds as b>a. The second condition of Case 3 says to check f(x,b,y)$>_{lpo}$ a and f(x,b,y)$>_{lpo}$ y. Both of these are true by Case 1.

Summary of Slides 15

1. Rewrite systems are most useful when they are terminating. There are several ad hoc methods to show termination, as stated in Facts D, E and F.

2. Important properties of term orderings are *stability*: if s<t then also sθ <tθ; *monotonicity*: if s<t then f(...s...) < f(...t...); and *simplification*: t<f(..t...).

3. s≤t is defined as s<t or s=t (for a partial order ≤).

4. The more useful orders are based on *quasi-orderings*, which are (partial) orders that are not anti-symmetric. That is, it is possible for two terms s and t to satisfy s <≈ t and t <≈ s and yet for s≠t. s < t iff s <≈ t and not (t <≈ s).

5. The three quasi-orderings considered here are *knuth bendix ordering* (kbo), *recursive path ordering* (rpo) and *lexicographic ordering* (lpo). All three orders depend also on an ordering of function symbols, which can be chosen by the user. The last two are very similar and only differ when the two terms to be compared have the same top level functor. The knuth bendix ordering depends also on a total order on ground terms that is also a simplification ordering. The recursive path ordering uses the concept of multi-set ordering.

ERUDIO

In 2010, MSc student Andrei-Dvornik implemented ERUDIO, a tool for carrying out Knuth Bendix completion. It operates in two modes - automatic, or step-by-step. The latter is very useful for learning about the various orderings, finding critical pairs and so on. (I will arrange a lab session to show how it works.)

It has several built-in ways to order equations (namely kbo, rpo and lpo).

You can run the tool by typing erudio at the linux prompt. The first window allows you to add equations and save them, or to load previously saved equations. The next window allows you to perform ordering of the equations. Only after equations have been ordered consistently can you move on to the superposition window, which allows to find critical pairs and other useful simplification steps. (See slides 16).

A further feature is that the tool deals with equations that cannot be ordered.

Warning: There are one or two bugs in the tool, but I assure you that there are not very many (we only found one or two last year).

There are many improvements and extensions possible if anyone would like an interesting project.