AUTOMATED REASONING

SLIDES 7:

SOME SYNTACTIC STRATEGY REFINEMENTS
- **Predicate Ordering**
- **Locking**
- **Atom Ordering**

KB - AR - 09

The Predicate Ordering Syntactic Refinement

7ai

Predicate Ordering Strategy:
- Give each predicate a value from some (partial) order
- Resolve only on a predicate in a clause that is *minimal in the order*
- Within the strategy perform a saturation search.

Example: S= (1) ¬Ha, (2) Fx ∨Hx, (3) ¬Gz ∨¬Fb, (4) ¬Fx ∨¬Hb, (5) Gx ∨¬Fx

Stage 0

Order predicates as G <H <F. i.e. resolve on an F literal only if no H or G literals and resolve on an H literal only if no G literals.

---

6 (1,2) Fa 7 (2,4) Fb ∨¬Fx

8 (3,5) ¬Fb ∨¬Fx (factors to ¬Fb - safe factoring as ¬Fb subsumes ¬Fb ∨¬Fx which can be removed. Also subsumes clause 3.) Stage 1

---

9 (6,7) Fb (subsumes clause 7) 10 (8,9) [] Stage 2

Exercise:
(1) Identify forwards and backwards subsumption steps in an alternative Stage 2
(2) Try a different ordering.

When is Predicate Ordering useful?

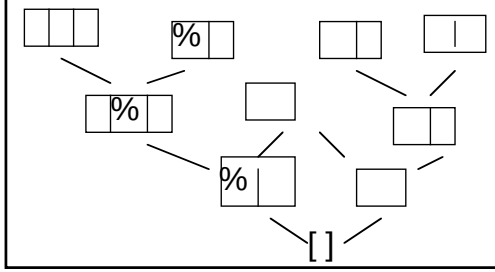
7aii

Answer: If some predicates can naturally be regarded as being more important than others and the problem involves several different predicates.

e.g. in reasoning about plans, some goals may be more difficult to achieve than others and early detection of their failure leads to smaller search spaces.

Make higher priority predicates come early in the order (i.e. high priority predicates are "<" low priority predicates); think of this as the strategy "preferentially choose high priority predicates".

eg In a resolution derivation every literal from the instances of given clauses must eventually be resolved upon and eliminated. In the schematic example below assume factoring occurs within a resolution step (when required).



If the literal "%" is one which it is considered might not easily be resolved away, it may be better to know this early in a derivation attempt; so give it a low position in the order. If it is one which can very likely be resolved away then can give it a low priority) ie it should come late in the order).

Properties of the Predicate Ordering Strategy

7aiii

Is the strategy *SOUND* ? (When it yields a refutation, is that "correct".)
YES - WHY?

The reason is typical of refinements that restrict the resolvents that may be made in a resolution derivation, but nevertheless perform correct steps.

Is the strategy *COMPLETE* ?
(Does it yield a refutation whenever one is possible?) YES

To show completeness can modify the semantic tree argument. Consider atoms in the tree according to their order: those highest in the order go first (i.e. at the top) and those lowest go at the end, with the rest in order in between.
eg if G<F then F atoms are put above G atoms.
Question: Why does this show completeness?

(NOTE for resolution applications: It is most usual to require just one refutation of [] . In the case of a logic programming system all refutations of the goal clause may be required, which is only applicable if the "goal clause" is identified. Therefore, unless otherwise stated, backwards subsumption will be applied, even though some “proofs” may be lost – see slide 6biii.)

Proving Completeness of Resolution Refinements

7aiv

Most completeness proofs for resolution strategies have two parts:

(i) show that a ground refutation of the right kind exists for some finite subset of the ground instances of the given clauses (eg see 4cii), and then

(ii) transform such a ground refutation to a general refutation (called *Lifting*).

To show (i) there are 3 possible methods:

(a) show that some refutation exists and transform it to one of the right kind; (useful for restrictions of resolution to control the search space), or

(b) show a refutation of the right structure exists by induction on a suitable metric (e.g. number of atoms, size of clauses), or

(c) give a prescription for directly constructing a resolution refutation of the right structure (eg using a semantic tree)

For Predicate ordering we used the third approach and adapted the method using semantic trees.

Strategies for Controlling resolution:

7av

The control strategies in Slides 7 all make use of a syntactic criterion on atoms to order literals in clauses. Literals are selected in order (<), lowest in order first. The strategies are quite simple, but can have a dramatic effect on the search space in some circumstances.

Exercise: In which situations would *predicate ordering* be useless? When might it be useful? When might *locking* not be very effective? When could it prove useful?

In all the strategies we assume that subsumption occurs, although for locking, subsumption may be a problem - see the examples on 7biii. Factoring can probably be restricted to safe factoring, and at any time, but I am not aware that this has been proved formally. Here it is assumed that when factoring (whatever kind) occurs, the factored literals that are retained are the ones least in the order. eg in locking, if some factoring substitution unifies 3 literals with indices 1,5,7, and 2 other literals with indices 4,10, the indices on the retained literals are 1 and 4.

The *atom ordering strategy* is the most fine-grained (and complex) of those considered. As an example, *ground atoms* might be given a weight according to the symbols (predicate, function and constant) that occur in it. Each symbol is given a weight value ≥0 and the weight of a literal is the sum of the weights of the symbols occurring in it. Such an ordering is called a *weight ordering* (a particular kind of atom ordering). As long as there are at most a finite number of symbols assigned any particular weight, a set of ground literals may be partially-ordered by their weight. If the signature is finite (as it usually is for a given problem) this restriction is obviously satisfied.

Atom Orderings:

7avi

e.g. let Sig = <{H,G, F}, {}, {a, b}>. If symbols are assigned a weight according to the order they appear in the signature (H=1, b=5), the weights of the ground literals occurring in the problem on slide 7ai are: Ha : 5, Fb : 8, Hb : 6. *The weights of literals with variables will depend on the bindings to the variables.* In each clause the literal with lowest weight is (1): ¬ Ha, (2): Hx, (3): ¬Gz, (4): ¬Hb, (5): Gx. In clause (2), whatever value is given to x, Hx always has lower weight than Fx. In clause (3) Gz is always smaller than Fb, whatever z. A derivation using the above ordering is: (6=1+2): Fa, (7=2+4): Fb ∨¬ Fx, (8=6+7): Fb, (9=3+5): ¬Fx ∨ ¬Fb, which safe factors to ¬Fb, which subsumes (3), (10=8+9):[ ]

If the weights had been differently assigned, e.g. a=1, b=2, F=3, G=4, H=5, then Fb has weight 5 and Gz has weight either 5 or 6, depending on the binding to z. In clause (3) therefore, Fb and Gz are not comparable, since in one case ¬Fb must be selected and in the other case either Fb or ¬Gz could be selected. Thus both are deemed minimal in clause (3) and either can be selected. A refutation using this weight ordering is:
(6=2+3): Hb ∨¬Gz, (7=2+4): Hx ∨ ¬Hb, (8=2+5): Hx ∨Gx, (9=6+8): Hb∨ Hx, which safe factors to Hb and then subsumes (6), (10=1+7): Hx, (11=1+10): []

Other kinds of orderings have been investigated; e.g. we might ignore variables, so Fx is assigned a weight depending only on F, etc. In this case, literals that are not ground are preferred over ground literals with a similar structure (e.g. Fx <Fa). The derivation according to this ordering is: (6=1+2): Fa, (7=3+5): ¬ Fb ∨¬ Fx, which safely factors to ¬Fb, (8=4+6): ¬ Hb, (9=2+8): Fb, (10=7+9): [ ]

The atom ordering strategy can simulate predicate ordering but not locking.
Locking can simulate predicate ordering and sometimes atom ordering.

The Locking Refinement

7bi

A more general kind of literal ordering is *Locking*, invented by BOYER.
Each literal in a given clause is assigned an index (not necessarily unique) and literals in a clause are ordered by the index, lowest indexed literals being the ones selected for resolution.

Locking Strategy:

Assign arbitrary indices (called locks) to literals.
(It is usual to use numbers, but any partially ordered set could be used.)
Only resolve on a literal with lowest lock in a clause.
Note:There is no need for indices on unit clauses.

Locking can be used in applications to restrict the use of clauses. eg.

- some clauses are more suited to 'top-down' use; (Horn clauses ?)

eg a clause A ∧B → C might only be useful in the context of the goal C

- some clauses are better suited to 'bottom-up' use (security access ?)

eg a clause A∧B → C may have originated from A →(B → C)
and the B → C part should only be made available when A is known to be true; ie perhaps A might be a more important condition than B.

- Locking can simulate predicate ordering - HOW?

Example: (We'll omit the "∨" symbol in the clauses in what follows)
(locks are indicated in brackets after each literal and the lowest is underlined)

(1) ¬Ha (2) { Fx (3), Hx(8)}
(3) {¬Gz(5), ¬Fb (2)} (4) {¬ Fx(6),¬ Hb (4)} (5) {Gx(1), ¬ Fx(7)}

6. (2,3) {Hb (8), ¬Gz(5)}

7. (5,6) {¬Fz(7), Hb(8)}
8. (2,7) {Hz(8), Hb(8)} factors to Hb which subsumes 6 and 7
9. (8,4) ¬Fx subsumes 3,4 and 5
10. (9,2) Hx subsumes 2 and 8
11. (10,1) []

In all ordering refinements (ordinary) factoring should involve one literal lowest in the ordering. Of the factored literals the lowest is retained. Any other factoring can be delayed.

On the other hand safe factoring can occur at any time **(*)**
In Locking, subsumption can be problematic, so (*) is not immediately obvious.

Exercise: Consider whether the statement (*) is true. 7bii

Properties of the Locking Refinement 7biii

Locking is *COMPLETE*.
(The inductive proof of this is given on Slide 7bv)

Some Unexpected Problems:

(i) *Problem with Tautologies*:
Sometimes it is necessary to generate tautologies in the locking refinement.

Example
{P(1),Q(2)},{P(13),¬Q(5)},{¬P(9), Q(6)},{¬P(3),¬Q(8)}

The only resolvents are tautologies; e.g. {P(13), ¬P(9)}.

(ii) *Problem with Subsumption*:
Does {A(8), B(6)} subsume {A(3), B(5), C(7)}?
Although it does so without locks, could it be that removing the longer clause will prevent a locking refutation, since the first clause allows only to resolve on B, whereas the second allows only to resolve on A?

7biv gives an example of forward subsumption causing a problem of this sort.

Example 7biv
1. {P(1),Q(2)}, 2. {P(13),¬Q(5)}, 3. {¬P(9), Q(6)}, 4. {¬P(3),¬Q(8)}

5. (1+4) {Q(2), ¬Q(8)} a tautology (can safely be deleted)
6. (2+3) {¬P(9), P(13)} another tautology (cannot be deleted)
7. (5+2) {¬Q(8),P(13)} ``subsumed'' by 2
8. (1+6) {Q(2), P(13)} ``subsumed'' by {P(1), Q(2)} **MUST KEEP**

If the subsumed (8) is removed there is no way to derive the empty clause.

If {Q(2),P(13)} (i.e. clause (8)) is retained, the refutation continues:

9. (8+2) P(13), 10. (4+9) ¬Q(8), 11. (3+10) ¬P(9), 12. (9+11) [ ]

(Indices on unit clauses are retained here to show the origin of the literals.)

In this example a backward subsumption step between {Q(2),P(13)} and {P(1),Q(2)} allows a solution to be found, whereas a forward subsumption step that removes clause (8) does not. Also, if the subumed clause (ie clause (1) here) has not been used in all ways, deleting it too early may also be problematic.

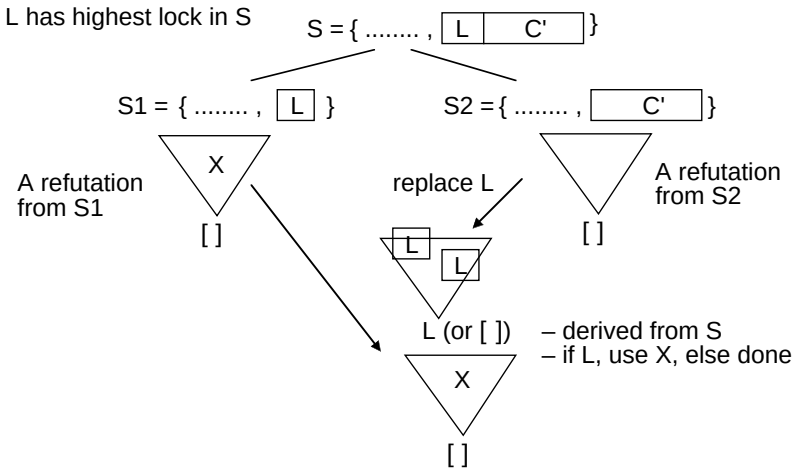
Proof that Locking for ground clauses is a Complete Refinement 7bv

The proof uses induction on the number of *excess literals*:
(excess literals = number of literals - number of clauses)
And also shows that whatever locks are chosen there is always an available step

The IDEA:
Assume S is unsatisfiable set of ground clauses (See diagram on next slide)
Base Case: (excess literal count = 0).
All clauses are unit clauses, so if unsatisfiable must be A and ¬A.
Induction Step: (excess literal count =k>0).
- Select the literal L with highest lock, which will be in a non-unit clause C = L∨C', and temporarily remove it from C.
- Divide the clauses S into two sets S1=S-{C}+{L} and S2=S-{C}+{C'}.
- Then show S1 and S2 are unsatisfiable and have excess literal count <k and use the induction hypothesis to find refutations.
- Add L back into S2 and combine the resulting refutation of L with that of S1.
- (The details are on Slide 7bvii. This proof is an example of approach (b) on Slide 7aiv. It still has to be lifted to the general case, but that is not difficult.)

Structure of Locking derivation used in completeness

7bvi



Proof that Locking for ground clauses is complete:

7bvii

Let set of ground clauses S be unsatisfiable. We show by induction on the excess literal count (k) that there is a locking refutation from S.

Base Case (k=0). Every clause in S must be a unit clause and as S is unsatisfiable there must be A and ¬A for some A which can be resolved to give [ ].

Induction Step (k>0) Assume as IH that for any unsatisfiable clause set S' with excess literal count <k there is a lock refutation.

Select the literal L in S with highest lock (it must be in a non-unit clause C=L∨C') and form S1=S-{C}+{L} and S2=S-{C}+{C'}. Both are still unsatisfiable: for if either had a model that model would satisfy L or C' and hence also S. Both have an excess literal count <k as C' and L are both smaller than C. Therefore, the induction hypothesis can be used to find a lock refutation of both. In the case of S2, adding back L gives either a derivation of L from S or of [ ] from S, in case L merges with a literal of lower lock index.

In the second of these we are done. In the first case, use the derived clause L wherever L was used in the refutation from S1 to derive [ ].

It is instructive to follow the construction for an example, say for the one on Slide 7biii. If you do so, you should find it constructs exactly the proof given on Slide 7biv.

Atom ordering

7ci

An extension of predicate ordering is to order atoms according to all of their symbols, not just the predicate.

Ground atoms can be ordered according to any (ordering) criteria.

Example 1: P(a) < P(b) < P(f(a)) < … < Q(a,a)<Q(a,b)< ...< R(a) etc.
(i.e. A<B if A is lexicographically before B when A and B are written as lists)
This example is continued on slide 7ciii.

Example 2: (Sig = ({P,Q,R,S}, {f}, {a,b});
give each symbol a weight (P:1, a: 1, Q:2, b: 2, R:3, f: 3, S:4) and
define wt(atom (or term)) = sum of wts of symbols in atom or term;
For atoms or terms A and B, we say A < B
if wt(A) < wt(B), or wt(A) = wt(B) & $wt(arg_j(A)) < wt(arg_j(B))$
& $wt(arg_i(A)) = wt(arg_i(B))$, for all i<j
(Note: count the predicate/function symbol as arg0.)

i.e. Pa<Pb<P(f(a))<P(f(b))<P(f(f(a)))…; P(x) < Q(x,a) < R(x) < S(x) for any x;
Qaa<Qab<Qba<Qbb<Qaf(a) <Qf(a)a<Qbf(a)<…; P(f(a))<Q(a,b)<...

If no two predicate symbols and no two functors (including constants) have the same weight, then the order is total. (**Easy exercise**: Show this.)

7cii

Atom Ordering Strategy:
Literal L is *minimal* in clause C if not(∃K in C s.t. K<L) .
Select for resolution only literals that are minimal in a clause.
Notice that a clause always has at least one such literal.

But clauses are not usually ground:
An ordering on ground atoms can be partially extended to non-ground atoms as long as it ***respects instantiation:***
i.e. $A \le B$ iff $A\theta \le B\theta$ for any θ.
This is called a ***stable ordering***.

Examples: Use the lexicographic ordering above.
In the clause {P(a), P(b)}, P(a) < P(b)

In {P(a), P(x), R(x)}, $P(a) \le P(x)\theta$ for all substitutions θ hence $P(a) \le P(x)$.
In {P(b), P(x), R(x)}, P(b) and P(x) are incomparable;
(it depends whether x≥b or not), so can select either.

In {P(x), P(y), Q(x,y)} if x==a, y==b, then P(x) is the minimal literal.
If x==b and y==a then P(y) is the minimal literal.

In the clause {Q(x, y), Q(y, x)}, is Q(x,y) < Q(y,x) ?
It depends on the bindings to x and y, so Q(x,y) and Q(y,x) are not comparable in the clause and *both* are minimal.

Example 7ciii

(1) {Px,Ry,¬Qxy} (2) {¬Sz, ¬Rz } (3) {Pu, Qf(v) v}
(4) Sa (5) Sb (6){¬Pf(a), ¬P(f(b)}

(7) (1+6) {Ry, ¬Qf(a)y, ¬ Pf(b)}
(8) (3+6) { Qf(v)v, ¬Pf(b)}

(9) {1+7} {Ry, ¬Qf(a)y, ¬Qf(b)y1, Ry1}
(10) (3+7) {Ry, ¬Qf(a)y, Qf(v)v} } Note 2 minimal literals
(11) (1+8) {Ry, ¬Qf(b)y, Qf(v)v}
(12) (3+8) Qf(v)v (after factoring)
(12) subsumes (3), (8), (10), (11)

(13) (9+12) {Ra, ¬Qf(b)y1, Ry1}

(14) (12+13) {Ra, Rb}

(15) (14+2) {¬Sa, Rb}

(16) (15+2) {¬Sa, ¬Sb}

Use ordering in Example 1 on 7ci. (Lexicographic)

(17) (16+4) ¬Sb
(18) (17+5) []

Features of Atom Ordering

7civ

A possible improvement?
Do *not* resolve on an atom if the mgu would make it non-minimal in its clause.

e.g. In {Px, Pb}, don't resolve on Px if x becomes bound by the mgu to c>b.

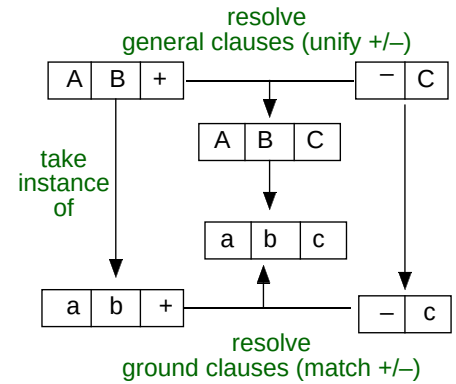
Completeness of Atom Ordering?
It is not difficult to show by the semantic tree method that an atom ordered refutation will always exist for a set of *ground* unsatisfiable clauses - can use the method suggested for predicate ordering. But in the general case, can we be sure all the necessary steps are allowed at the first order level?
(See 7cv for an illustration)

If extension to first order is stable, then:
Given a ground refutation, when lifting it to the general level, either:
- all instances Lθ of the general literal L used for resolving respect the ordering in the clause - i.e. L is minimal so no problem, or
- some instances do not; then L would be incomparable with others in the clause and so still be available by the strategy; thus no proofs are lost.

(Note that no other literal in the clause would be <L, for then its instances would be < Lθ and the ground literal of L used in the given ground refutation could not have been minimal.

Completeness of Atom Ordering (illustrated)

7cv



Assume:
a>+ b>+ c>–

If A>+ and B>+ there is no problem as we will be able to find the required step corresponding to the step taken at ground level.

Could +>A?
No - because of stability; it would force +>a, contradicting assumption.

It could be that both A and + were minimal, so the step resolving between + and – would be allowed.

Analagously for B,C and –.

Summary of Slides 7

7di

1. Resolution derivations need to be controlled to avoid the search space exploding, even when tautology deletion, subsumption deletion and factoring are applied.

2. Syntactic methods that restrict particular literals in a clause can be used. Three methods were considered: Predicate Ordering, Locking and Atom Ordering.

3. Locking orders literal occurrences in a given set of clauses. Literals in resolvents inherit their locks from the corresponding literals in the parent clauses. (For recursive clauses this might be a problem - can you see why? - and an extension is to use dynamic locking. See Chapter Notes on Controlling Resolution if interested.)

4. An order is *stable* iff,
A<B in a clause C implies Aθ < Bθ in Cθ, for any instance Cθ of C.

5. In Locking, the order of a literal in a clause depends solely on its lock relative to other literals and does not vary with instance. Thus the locking order is stable: if A<B in a clause C then the lock on A is < the lock on B and similarly Aθ < Bθ in Cθ, for any instance Cθ of C.

6. Atom Ordering (and the simpler Predicate Ordering) orders atoms according to some syntactic criteria, such as weight ordering, or lexicographic weight ordering.

7. Atom Orderings are not necessarily stable. (**Exercise**: make up an example of a non-stable atom ordering). If C is a clause in which literals A and B may be ordered A<B or B<A, depending on the particular instance of C, then A and B are not comparable. If there is no literal definitely < A, whatever the instance, then A is minimal in the clause and may be selected for resolution.

8. Locking can simulate Predicate Ordering, as can Atom Ordering. That is, suitable locks can be found, or a suitable atom ordering can be found, such that applying the strategy of Locking (or Atom ordering) generates the same search space as using the strategy of predicte ordering. (**Exercise**: Suggest suitable locks/atom ordering for the simulations.)

9. Atom Ordering cannot simulate Locking, nor can Locking simulate Atom ordering.

10. There are problems with subsumption and tautolgy deletion in the Locking strategy - completeness may be lost if unrestricted tautology deletion and subsumption is allowed.