AUTOMATED REASONING

SLIDES 7:

SOME SYNTACTIC STRATEGY REFINEMENTS

Predicate Ordering

Locking

Predicate Ordering Syntactic Refinement (ppt)

Predicate Ordering Strategy:
- Give each predicate a value (called an index) from some (partial) order (usually use positive integers as indices)
- 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

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.

Stage 0

| 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.

Predicate Ordering a good strategy? (ppt)

Is Predicate Ordering a good strategy? (ppt)

If some predicates are regarded as being "more important" than others and the problem involves several different predicates, predicate ordering can be useful.

Example:

In a resolution derivation every literal from the used 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 an index near minimal in the order. If it is one which can very likely be resolved away then can give it a low priority (i.e. its index should come near maximal in the order).

Generally, 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".
Properties of the Predicate Ordering Strategy

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 always 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 (or least preferred) go first (i.e. at the top) and those lowest (or most
preferred) go at the end, with the rest in order in between.
i.e. if G<F then F atoms are put above G atoms.

Question: Why will this show completeness?
(Hint: think about how a semantic tree is reduced to produce a refutation)

Proving Completeness of Resolution Refinements

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 5ai), and then
(ii) transform such a ground refutation to a general refutation (also of the right
kind) (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 (c) adapting the method of semantic trees.
For Locking (next) we'll use (b).

Strategies for Controlling resolution:
The control strategies in Slides 7 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 in some circumstances can have a dramatic effect on the search space.
In all the strategies we assume that subsumption occurs;
however, as the examples on 7biiv show, for the locking refinement subsumption may be a
problem; furthermore, although factoring can probably be restricted to safe factoring, and be
applied at any time, I am not aware that this has been proved formally. Here, it is assumed
that when factoring (whatever kind) occurs in locking, the factored literals that are retained
are the ones least in the order. e.g. if some factoring substitution unifies three literals with
indices 1, 5, 7 and two other literals with indices 4 and 10, then the indices on the retained
literals are 1 and 4.

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?
The atom ordering strategy ( in the optional material) is the most fine-grained (and complex)
This strategy orders atoms by taking into account their arguments as well as their predicate.

NOTE for resolution applications: It is usually required to find just one refutation of []
(cf logic programming where all refutations of the goal clause may be required – this is
usually applicable only if the "goal clause" is identified. Unless otherwise stated, backwards
subsumption will be applied, even though some "proofs" may be lost – see slide 6biv.)
The Locking Refinement (ppt)

A different kind of literal ordering is Locking, invented by BOYER (1973).
Each literal in a clause is assigned a numeric index (not necessarily unique).
Literals in a clause are ordered by index, lowest first.

Locking Strategy:
1. Assign arbitrary indices (called locks) to literals in given clauses.
2. Only resolve on a literal with lowest lock in its clause.
Note: There is no need for indices on unit clauses.
3. Resolvent literals inherit indices from the parents.

Example: (We’ll omit the “∨” symbol in clauses in what follows)
(llocks 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) {¬Fx(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) []

Use of Locking

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 betrue; ie perhaps A might be a more important condition than B.

• Locking can simulate predicate ordering - HOW?

Locking is COMPLETE.
(An inductive proof of this is given on Slide 7bvi)

Properties of the Locking Refinement

Some Unexpected Problems:

(i) Problem with Tautologies:
In most resolution refinements a tautology is redundant, so is never generated
In the locking refinement, it is sometimes necessary to generate tautologies

Example
{P(1), Q(2)}, {P(13), ¬Q(5)}, {¬P(9), Q(6)}, {¬P(3), ¬Q(8)}
The only resolvents are tautologies; eg. {P(13), ¬P(9)} (match ¬Q(5) and Q(6))

(ii) Problem with Subsumption:
Question: Does {A(8), B(6)} subsume {A(3), B(5), C(7)}?
Although it does so without the locks, could it be that removing the longerclause will prevent a locking refutation, since the first clause allows only toresolve 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 of the locking subsumption problem

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)\} is a tautology (can safely be deleted)\}  
6. \{(2+3) \{¬P(9), P(13)\} another tautology (cannot safely 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, even if all of 1-7 are kept.

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.)

Continued on 7bv ......
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 $S$.

Assume as induction hypothesis (IH) that for any unsatisfiable clause set $S'$ with excess literal count < $k$ there is a lock refutation.

Case 1 ($k=0$): Every clause in $S$ must be a unit clause and as $S$ is unsatisfiable there must be $A$ and $\neg A$ for some $A$ which can be resolved to give $\bot$.

Case 2 ($k>0$): Select the literal $L$ in $S$ with highest lock (it must be in a non-unit clause $C=L\lor C'$) and form $S_1=S-\{C\}+\{L\}$ and $S_2=S-\{C\}+\{C'\}$. Both are still unsatisfiable: for if either had a model that model would satisfy $L$ or $C'$, as well as $S-\{C\}$, and hence also $S$. Both have an excess literal count < $k$ as $C'$ and $L$ are both smaller than $C$. Therefore, IH can be used to find a lock refutation of both. In the case of $S_2$, adding back $L$ gives either a derivation of $L$ from $S$ or of $\bot$ 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 $S_1$ to derive $\bot$.

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

Notice also that since locking is complete, for any ground clauses $S$ and any set of locks there must always exist a first step.

More details of Proof that Locking for ground clauses is complete:

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. Two methods were considered: Predicate Ordering and Locking.

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, whereby locks are globally reset during a proof.)

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

5. Locking can simulate Predicate Ordering. That is, suitable locks can be found, or a suitable atom ordering can be found, such that applying the strategy of Locking generates the same search space as using the strategy of predicate ordering.

(Exercise: Suggest suitable locks/atom ordering for the simulations.)

Summary of Slides 7:

START of OPTIONAL MATERIAL (SLIDES 7)

Atom Ordering
The Atom Ordering Strategy (continued on 7dii):

In the atom ordering strategy, shown in Slides 7e as an optional Case Study, atoms are ordered by taking into account their arguments as well as their predicate.

For example, a ground atom might be given a weight according to the symbols (predicate, function and constant) that occur in it. Each symbol is given a weight \( \geq 0 \) and the atomic weight of atom \( A \) is the sum of the weights of the symbols occurring in \( A \). Such an ordering is called a weight ordering (a particular kind of atom ordering).

Assuming the signature is finite then a set of ground atoms may be partially-ordered by their weight. (See slide 7dvi for details.) If no two symbols have the same weight, then the ordering on 7dvi will be total. (Find an example of a non-total ordering if this condition is false.)

Example:

Assign symbols a weight according to their order in the signature (\( H=1, b=5 \)), and the atomic weight of atom \( A \) is the sum of the weights of the symbols occurring in \( A \).

The weights of literals with variables will depend on the bindings to the variables. In each clause the literal with lowest weight is (1): \( \neg Ha \), (2): \( Hx \), (3): \( \neg Gz \), (4): \( \neg Hb \), (5): \( Gx \). In clause (2), whatever the binding to \( x \), \( Hx \) always has lower weight than \( Fx \). In clause (3), \( Gz \) is always smaller than \( Fb \), whatever the binding to \( z \).

Atom Ordering (1) (A case study)

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

Example of atom order 1:

\[
\begin{align*}
P(a) < P(b) < P((a)) < \ldots < Q(a,a) < Q(a,b) < \ldots < R(a)
\end{align*}
\]

(i.e. \( A \leq B \) if \( A \) is lexicographically before \( B \) when \( A \) and \( B \) are written as lists )

This ordering is used in the example on slide 7cvi.

Atom Ordering Strategy:

Literal L is minimal in clause C if not(\( \exists K \) in C s.t. \( K \leq L \)).

Select for resolution only literals that are minimal in a clause.

Notice that a non-empty clause always has at least one such literal.

Can you think of a problem when all symbols are considered?

An ordering on ground atoms can be partially extended to non-ground atoms as long as it respects instantiation:

i.e. \( A \leq B \) iff \( A[\theta] \leq B[\theta] \) for every \( \theta \).

\( A \leq B \) if \( A \leq B \) or \( A=B \)

This is called a stable ordering.

Example of atom order 2:

\[
\begin{align*}
\neg Hx < \neg Fx < \neg Hb < \neg Gx
\end{align*}
\]

Atom Orderings continued:

A derivation using the ordering on 7ei is:

\[
(6=1+2): Fa, \quad (7=2+4): Fb \vee \neg Fx, \quad (8=6+7): Fb, \quad (\text{note literals in 7 are not comparable})
\]

\[
(9=3+5): \neg Fx \vee \neg Fb, \quad \text{which safe factors to } \neg Fb, \quad \text{which subsumes (3),}
\]

\[
(10=8+9): [ ]
\]

The weights could have been differently assigned; e.g. \( a=1, b=2, F=3, G=4, H=5 \).

In this case \( Fb \) has weight 5 and \( Gz \) has weight either 5 or 6 depending on the binding to \( z \).

Therefore, in clause (3) \( Fb \) and \( Gz \) are not comparable, since in one case \( \neg Fb \) must be selected and in the other case either \( \neg Fb \) or \( \neg Gz \) could be selected. Thus both are deemed (potentially) minimal in clause (3) and either can be selected.

A refutation using this weight ordering is:

\[
(6=2+3): Hb \vee \neg Gz, (7=2+4): Hx \vee \neg Hb, (8=2+5): Hx \vee Gx, (9=6+8): Hb \vee Hx, (10=1+7): Hx, (11=1+10): [ ]
\]

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:

\[
(6=1+2): Fa, \quad (7=4+6): \neg Hb, (8=3+4): \neg Fb \vee \neg Fx, \quad \text{which safely factors to } \neg Fb, \quad (9=2+7): Fb, \quad (10=8+9): [ ]
\]

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

Atom ordering (2)

Unfortunately for general clauses an atom order is not a total order (given two atoms \( X \) and \( Y \), it may not hold that \( X \leq Y \) or that \( Y \leq X \)).

Examples: These all use the lexicographic ordering

In the clause \( P(a) \vee P(b) \), \( P(a) \leq P(b) \)

In \( P(a) \vee P(x) \vee R(x), P(a) \leq P(x) \theta \) for all substitutions \( \theta \) hence \( P(a) \leq P(x) < R(x) \)

In \( P(b) \vee P(x) \vee R(x), P(b) \) and \( P(x) \) are incomparable

(it depends whether \( x \equiv b \) or not), so can select either

In \( P(x) \vee P(y) \vee Q(x,y) \)

if \( x \equiv a, y \equiv b \), then \( P(x) \) is the minimal literal

if \( x \equiv b \) and \( y \equiv a \), then \( P(y) \) is the minimal literal

In \( Q(x,y) \vee \neg Q(y,x) \), is \( Q(x,y) \leq Q(y,x) \)?

It depends on the bindings to \( x \) and \( y \), so \( Q(x,y) \) and \( \neg Q(y,x) \) are not comparable in the clause and \( both \) are minimal.
Example of a refutation using Atom ordering

1. \( P_x \vee R_y \vee \neg Q_{xy} \)
2. \( \neg S_z \vee \neg R_z \)
3. \( P_u \vee Q_f(v) \vee v \)
4. \( S_a \)
5. \( S_b \)
6. \( \neg P_f(a) \vee \neg P_f(b) \)

(1) \((1+6)\) \( R_y \vee \neg Q_f(a) \vee \neg P_f(b) \)
(2) \((3+6)\) \( Q_f(v) \vee \neg P_f(b) \)
(3) \((1+7)\) \( R_y \vee \neg Q_f(a) \vee \neg P_f(b) \)

Note 2 minimal literals

The Weight Atom Ordering

\( \text{Sig} = (\{P,Q,R,S\}, \{f\}, \{a,b\}) \);
give each symbol a weight (e.g. \( P:1, a:1, Q:2, b:2, R:3, f:3, S:4 \));
define \( \text{wt}(\text{atom} / \text{term}) = \text{sum of wts of symbols in atom / term} \);
\( \text{wt}(\neg A) = \text{wt}(A) \)

For ground atoms or terms \( A \) and \( B \), we say \( A < B \) if
either \( \text{wt}(A) < \text{wt}(B) \),
or \( \text{wt}(A) = \text{wt}(B) \) & \( \text{wt}(\arg_j(A)) < \text{wt}(\arg_j(B)) \)
& \( \text{wt}(\arg_0(A)) = \text{wt}(\arg_0(B)) \), for all \( i<j \)

\( \text{Note: we count the predicate symbol of an atom, or functor of a term, as} \)
arg0.)

For the above weights:

\( P_a < P_b < P(f(a)) < P(f(b)) < P(f(f(a))) \ldots \);
\( P(x) < Q(x,a) < R(x) < S(x) \) for any \( x \);
\( Q_a < Q_b < Q_f(a) < Q_f(b) < \ldots \);
\( P(f(a)) < Q(f(a),a) < Q(f(b),b) < \ldots \);
\( P(f(a)) < Q(a,b) < \ldots \)

If no two predicate symbols or functors (including constants) have the same
weight, then the order on ground terms is total. i.e. given 2 ground atoms \( A \)
and \( B \) then either \( A < B \) or \( B < A \). (Easy exercise: Show this.)

Is Atom Ordering Complete?
Can employ the semantic tree method as for predicate ordering to show that an
atom ordered refutation exists for any set of ground unsatisfiable clauses.

Lifting to general case (Proof, see 7eix)

A possible improvement?
Do not resolve on an atom if the mgu would make it non-minimal in its clause.
e.g. In \( P_x \vee P_b \), don’t resolve on \( P_x \) if \( x \) becomes bound by the mgu to \( c>b \).

Do you see any problems with this suggestion?

Questions:
Suggest a simple ordering of atoms that enables atom ordering to simulate
duplicate predicate ordering.

Explain why locking can simulate atom ordering for ground clauses.

Explain why locking cannot usually simulate atom ordering for general
duplicate clauses? (Hint: consider the example on slide 7dv)

Can atom ordering simulate locking for ground clauses?
(Hint: locks can be arbitrary)
Lifting an Atom Ordering Derivation - proof of completeness

Assume the atom ordering is stable and that a ground refutation exists
Consider lifting a step (at ground level) involving a minimal literal $L$ and its complement to the general level; then either:
- all instances $L\theta$ of the general literal $L$ used for resolving respect the ordering in the clause - i.e. $L$ is minimal in the clause so no problem, or
- some instances do not respect the ordering; then $L$ would be incomparable with other literals in the clause and so still be available by the strategy; thus no proofs are lost.

Notice that no other literal $M$ in the clause would satisfy $M < L$;
Proof: Assume for contradiction there is such a literal $M$, $M < L$.
Then by stability $M\theta < L\theta$ for all ground substitutions $\theta$;
this includes the ground instance of $L$ used in the given ground refutation contradicting its minimality.
Hence $L$ is minimal (possibly by incomparability).

Summary of Optional Material in Slides 7
1. Atom Ordering orders atoms according to some syntactic criteria, such as weight ordering, or lexicographic weight ordering.

2. An order is stable iff,
   $A < B$ in a clause $C$ implies $A\theta \leq B\theta$ in $C\theta$, for any instance $C\theta$ of $C$.

3. 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 $< \theta$ the lock on $B$ and similarly $A\theta < B\theta$ in $C\theta$, for any instance $C\theta$ of $C$.

4. 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 $< A$, then $A$ is minimal in the clause and may be selected for resolution.

5. Atom Ordering cannot simulate Locking, nor can Locking simulate Atom ordering (in general).