AUTOMATED REASONING

SLIDES 8:

HYPER-RESOLUTION
Hyper-resolution Refinement
The Otter Theorem Prover
Generalised Hyper-resolution

KB - AR - 09

Hyper-Resolution

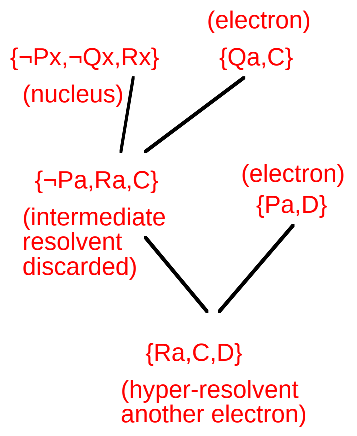
Hyper-resolution is the strategy employed in the widely used *Otter* family of provers.

Hyper-resolution generalises ``bottom-up'' reasoning and combines several resolution steps into one big step.

Hyper-resolution Strategy:

Clauses are divided into
nucleii (those with ≥1 negative literals),
electrons (those with no negative literals).

Resolution occurs between 1 or more electrons and 1 nucleus. There is 1 electron clause used for each negative literal in the nucleus.



A hyper-resolution step

Example (M(x,y,z) reads as z=x∗y)

(N)1. {M(x,y,z), ¬M(y,x,z)} Commutativity of times (x∗y=z if y+x=z)
(E)2. M(x,x,s(x)) x_squared=x∗x
(N)3. {D(x,y), ¬M(x,z,y)} y=x∗z → x divides y
(N)4. {¬P(x),¬D(x,u),¬M(y,z,u),D(x,,y),D(x,z)}
(x is prime ∧x divides u ∧ u=y*z→ x divides y ∨ x divides z)
(E)5. M(a,s(c),s(b)) (E)6. P(a) (N)7. ¬D(a,b)

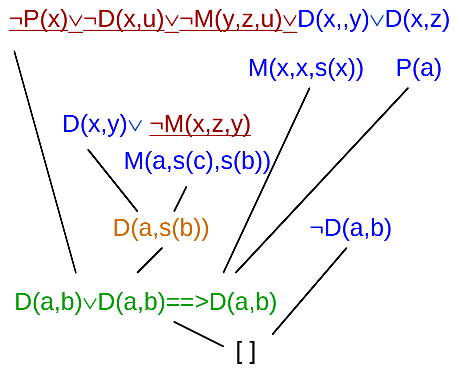
Goal is to show:
∀x[x is prime ∧ b_squared=x∗c_squared → x divides b] ≡
(when negated) ¬∀x[P(x) ∧M(x,s(c),s(b))→D(x,b)]
(which Skolemises to) P(a), M(a,s(c),s(b)), ¬D(a,b) (a is Skolem constant)

Nucleii: 1,3,4,7;
Electrons: 2,5,6

8. (1,5) M(s(c),a,s(b)) 9. (2,3) D(x,s(x)) 10. (3,5) D(a,s(b))
((1+2) gives M(u,u,s(u)) which is subsumed by 2

11. (8,3) D(s(c),s(b)) 12. (4,6,8,10) {D(a,s(c)),D(a,a)}
13. (4,6,9,2) {D(a,a),D(a,a)} factors to D(a,a) and subsumes 12
14. (4,6,10,2) {D(a,b),D(a,b)} factors to D(a,b)
15. (14,7) []

The derivation as a tree ....



Notice that only electrons are formed as final resolvents. In this refutation they happen to be facts, but need not be. (See clause 12 on Slide 8aii.)

Although nucleii are never derived, they may be deleted by subsumption using derived electrons.

Tautologies can initially be deleted, but they are never derived as they are nucleii.

Nucleii can be safely-factored at the start. Electrons can be factored.

Exercise: Show that, if a tautology is derived part way into forming a new electron, the final electron will be subsumed.

Outline PROLOG program for Hyper-resolution

8aiv

```
hyper(Ns, OldN,Es,Num):- member(empty,Es).
hyper([],OldN,Es,0):- writenl(['fail']), fail.
hyper([],OldN,Es,Num):- Num>0, not member(empty,Es),
                  hyper(OldN,[],Es,0).
hyper([N|RestN],OldN,Es,Num):-N_is_subsumed_by_Es(N,Es),
                  hyper(RestN,OldN,Es,Num).
hyper([N|RestN],OldN,Es,Num):-
                hyperresolve(N,Es,NewEs,Es1),
                append(NewEs,Es1,E2,Count),Num1 is Num +Count,
                hyper(RestN,[N|OldN],E2, Num1).
Initial call: hyper(N,[],E,0).
```

`hyper(A,B,C,D)` holds if Nucleii A and B and electrons `C` yield the empty clause. `D` is a flag to indicate no new resolvents can be formed. `append(A,B,C,D)` appends `A` and `B` to give `C` and D=length(A).

`hyperresolve(N,E,NewE,RestE)` holds if nucleus `N` yields non subsumed hyper-reolvents `NewE` , no clauses in `RestE`, a subset of `E`, are subsumed, clauses in `E-RestE` are subsumed (by clauses in `NewE`).

The 2nd and 4th args of `hyper` are used to maintain fairness. The nucleii are processed again and again, unless there are no new electrons, in which case there is failure.

Properties of Hyper-Resolution

8av

Hyper-resolution is Complete:
An inductive proof is considered in the problem sheets so is not given here.

Hyper-resolution can be combined with both predicate/atom ordering and locking:

Predicate ordering:
Only use ordering in electrons (Why?)
Only resolve on minimal atoms from an electron.
eg if R<Q<P then from P(a,x)∨Q(x,y) can only use Q(x,y)

Locking:
Only lock positive literals in electrons or nucleii;
Implicitly, negative literals are locked lowest in nucleii to force their use
eg Given $\neg P(a)\vee\neg Q(x)\vee R(x)_5$, $P(a)$, $Q(b)_4\vee Q(c)_7$ can derive $R(b)_5\vee Q(c)_7$

Notice that intermediate nucleii can only be used to derive electrons and not resolved with each other as the lowest locked negative literals in two nucleii cannot be resolved together.

If all positive literals are locked at 1, and all negative literals are locked at 0, then locking effectively simulates hyper-resolution. Can therefore adapt the completeness proof for locking to obtain a completeness proof for hyper-resolution, though that's not what's usually done. (See exercise sheet and answers.)

Hyper-Resolution:

8avi

Hyper-resolution forms the basis of a family of theorem provers from Argonne. OTTER (which you'll get a chance to use in the lab) was the first, and is best for beginners; its most recent descendant is called Prover9. In hyper-resolution each "step" is actually one or more resolution steps, made according to a simple syntactic principle: only (final) resolvents with positive literals (called *electrons*) are allowed to be derived. To form them, a clause with one or more negative literals (called a *nucleus*) is sequentially resolved with electrons, each time removing 1 negative literal, until an electron is produced. If the nucleus has 5 negative literals there would be 5 intermediate steps. Note that a particular electron may be used in more than 1 intermediate step, but for each use a fresh copy is taken. As each overall hyper-resolution step is generally more than one resolution step there are fewer possible steps overall, so the search space is reduced (compared with that for binary resolution).

The outline Prolog program on Slide 8aiv performs a saturation search to find the empty clause by hyper-resolution. Initially, the clauses are divided into nucleii and electrons. Assuming there are some of each (**why** must this be so if S are unsatisfiable?) each nucleus is used to find all hyper-resolvents from the current electrons. At the end of all the processing, and after applying subsumption, the number of new electrons produced is recorded. If there is at least one, then the process is repeated on the list of (non-subsumed) nucleii, otherwise there is failure. If the empty clause (an electron) is produced there will be success. All the hard work is performed by `Hyper-resolve`.

[By the way, the answer to the question above is that if all clauses are electrons, then they have a model - just assign T to all atoms. If no clause is an electron, then again the given clauses have a model - assign F to all atoms. For an unsatisfiable set of clauses S neither can happen, so S must contain at least one each of an electron and nucleus.]

Set of Support Strategy

8bi

The Set of Support strategy is related to Hyper-resolution and is used by Otter.

• For a given problem the clauses are divided into two sets, the SOS (set-of-support) and the rest. For example, at least some clauses in the clausal form of the negated conclusion are often put into the SOS.

• Resolvents may only be formed if at least one clause contributing to the resolution step is from the SOS, or is derived from such a clause (i.e. it has an SOS clause as an ancestor). As a consequence, note that **clauses not initially in SOS will never be resolved with each other.** (Therefore, if C is to be resolved with D at least one of them must be in SOS.) Such clauses behave a little like the nucleii in Hyper-resolution.

• Why is this a good strategy?
The clauses not in SOS are often satisfiable. e.g. the clauses in a Horn clause program could be the complement of the SOS. Resolving between the satisfiable non-SOS clauses may give interesting results, but not results that are useful for the problem in hand. Using at least one clause derived from the conclusion or clauses in SOS will more likely give useful results.

Exercise: (1) Prolog uses the SOS strategy. What is its SOS?
(2) What happens in this strategy if SOS is initially empty?

The OTTER Theorem Prover

8bii

Otter is a very versatile theorem prover with a simple interface. The successor is Prover9, but Otter is used in class as it gives single step and user interaction.

Otter forms resolvents using the SOS restriction strategy, and processes them according to user defined settings until the empty clause is deduced.

Otter makes use of 2 main lists: *sos* and *usable* (non-sos clauses).
The main loop is:

```
While sos is not empty and empty clause is not derived
     select given from sos (various criteria for selection)
     generate resolvents using usable clauses and given
             (various inference rules are allowed and restrictions can be imposed)
    move given to usable
    process new clauses and put kept  ones into sos (various criteria for retention)
End
```

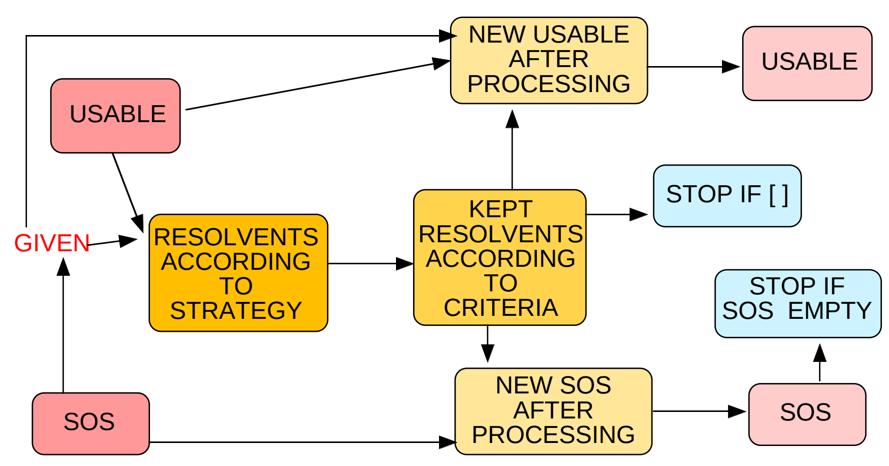
Selection criteria include number of literals, number of variables, weight of terms, most recent addition, oldest clause, etc.

Inference rules include resolution, hyper-resolution.

Retention criteria include measures as above, plus factoring, subsumption.

The OTTER Theorem Prover Main Loop

8biii


NEW USABLE AFTER PROCESSING
USABLE
USABLE
GIVEN
RESOLVENTS ACCORDING TO STRATEGY
KEPT RESOLVENTS ACCORDING TO CRITERIA
STOP IF [ ]
STOP IF SOS EMPTY
SOS
NEW SOS AFTER PROCESSING
SOS

Example1 usable: 1: ¬Ha 2: ¬Gz ∨¬Fb 3: ¬Fx ∨¬Hb 4: Gx ∨¬Fx
sos: 5: Fx ∨Hx

In all examples assume hyper-resolution, for/back subsumption, no factoring.

Select 5 as *given*. Compute resolvents: (5+1= 6): Fa, (5+4=8): Gx ∨Hx
(5+5+3=7): Hx ∨Fb (Remember, use Hyper-resolution,
and have only 1 electron (5), so cannot use (2))
sos is now 6, 7, 8 and usable is 1- 5.

Select 6. Compute resolvents: (6+5+3=9): Fb (subsumes 7), (6+4=10): Ga.
(Remember, use Hyper-resolution)
sos = 8, 9,10 and usable = 1- 6.

Select 10. Compute resolvents: (10+2+5=11): Hb.
sos = 8, 9,11 and usable = 1 - 6, 10.

Select 11. Compute resolvents: (11+3+6=12):[], (11+3+5=13): Hx.
Stop , as empty clause is deduced.

Example 2 usable = 1 - 5; sos = empty. Stops immediately!

Example 3 usable = 1: H ∨¬G 2: G sos = 3: ¬H
Select 3. No hyper-resolvents possible using only 3 and usable.
sos = empty, usable = 1 - 3. Stop! even though refutation possible.

8biv

The Otter Theorem Prover

8bv

Otter is a resolution based theorem prover that was developed at Argonne. It is based on the set of support principle. The user can control its deduction cycle in many ways by setting numerous flags and parameters.

Otter has a single main loop that forms resolvents and processes them until either the empty clause is deduced or the resources are used up. Otter keeps the clausal data in two main lists called the *usable list* and *sos list*. Resolvents are produced using one clause selected from the sos list and one or more clauses taken from the usable list. Its main operating loop is shown on Slide 8bii.

Criteria for selecting the given clause are measures such as number of literals, number of variables, weight of terms in the clause, etc. These can be controlled by the user. The inference rules can be resolution or hyper-resolution, as well as several others (see the on-line manual). Derived clauses are subject to various processing, such as factor forming, subsumption tests, etc. Again, the user can control which tests are carried out.

For example, the user can set a flag so that resolvents that have more than a set number of variables or literals are not kept. Although this will make the search space incomplete, it may keep it within reasonable bounds.

Otter allows user interaction which is helpful for beginners. However, a replacement called Prover9 (See http://www.cs.unm.edu/~mccune/mace4/) is recommended for serious work.

More about Otter 8bvi

There is a third list used by Otter called the *passive list*. Clauses in this list are not used to form resolvents unless they can be used to derive the empty clause immediately from two facts. For instance, perhaps we would like to know if an intermediate literal L is ever derived. In that case, we can put the negation of L into the passive list. If L is ever derived, the empty clause is immediately generated and Otter stops.

The clauses in the passive list can also be used to remove subsumed clauses. For example, suppose we know that clauses with a literal of the form P(f(f(f(x)))), for any x, will be useless. Otter uses the passive list to detect additional subsumed clauses: if P(f(f(f(x)))) is put into the passive list and the subsumption flag is set, then all clauses of this form will be subsumed and removed.

Initially, the user puts each initial clause either into the sos-list or into the usable-list. The sos-list acts as a set of support facility. All resolvents will be formed using a clause in the initial sos-list or from a resolvent having at least one ancestor from the initial sos-list. Note therefore, that if the sos-list is initially empty *no processing can occur* as there is no clause to be the first given-clause. It is possible for the user to interact with Otter to tell it which should be the next given-clause. Otter will find all proofs of the empty clause within the parameters set by the user, who can also constrain Otter to find just one proof. As an experiment, run Otter on the three clauses

-p(u,v) | -p(v,u), p(x,f(x)), p(f(y),y)

with various combinations of initial assignments of clauses to the sos-list and usable-list.

You will have a chance to try Otter on some simple problems. For instance, you could try the three problems from Slides 0 (Otter includes reasoning with equality). **You are expected to have used Otter and be familiar with the basic flags as part of the course.**

Hyper-resolution has an interesting feature 8ci

- Let I be an H-interpretation in which all atoms are **False**, so all negative literals are **True**. A nucleus will be True in I and an electron False in I.
- A hyper-resolvent (an electron) is False in I.
- Resolution occurs between one clause which is True in I (the nucleus) and other clauses (electrons) that are False in I.
- Hyper-resolution can be generalised for any H-interpretation I that splits a given set of clauses S into *non-empty* S1 and S2 s.t.

  - S1 = clauses which are True in I (still called nucleii)
  - S2 = clauses which are False in I (still called electrons).

- The strategy is still to resolve between 1 nucleus and ≥1 electrons.
- In the 'standard' hyper-resolution method it is easy to distinguish between nucleii and electrons:

- every instance of an electron clause is an electron
- every instance of a nucleus clause is a nucleus
- for each nucleus literal, either all instances are true, or all instances are false

- So whatever unifiers are involved, a clause remains either a nucleus or an electron. Any H- interpretation I that satisfies these criteria is called a **uniform** H-interpretation and can be used to form nucleii and electrons.

Example of a Uniform Interpretation in Hyper-resolution 8cii

Suppose I makes all atoms TRUE.
That is, atoms L are true and literals ¬L are false.
An all-negative clause becomes a standard electron, and
a clause with ≥1 positive literals is a nucleus.
The positive literals are resolved upon.
Notice that a Horn clause with 1 +ve atom is a nucleus needing 1 electron

EG Given the Horn clauses:

(1) P∨¬Q ∨¬R (N) (2) Q (N) (3) R ∨¬Q (N) (4) ¬P (E)

A hyper-resolution refuation using I can resolve (N) on underlined literals:

(1+4=5) ¬Q ∨¬R (5+2=6) ¬R (5+3=7) ¬Q (2+7=8) **[ ]**

(Do you notice any pattern? HINT: Think of a logic programming trace)

Rename each atom A as ¬A' (so I makes A' false) – gives

(1) ¬P'∨ Q' ∨R' (N) (2) ¬Q' (N) (3) ¬R' ∨ Q' (N) (4) P' (E)

Now use standard hyper-resolution

We can use renaming to show completeness using any uniform interpretation

In fact, for Horn clauses standard Hyper-resolution (without renaming) is just forward reasoning with Modus Ponens, whereas hyper-resolution w.r.t. I (I makes all atoms true) can simulate "top down" refutations (as in Prolog).

Using Uniform Interpretations in Hyper-resolution (1) 8ciii

Remember: electons are false in an interpretation I and nucleii are true.

Suppose I is a uniform interpretation that makes R, P True and Q false, and given: (1) P∨Q (*N*) (2) ¬P∨R (*N*) (3) ¬Q∨R (*N*) (4) ¬R (*E*)

A refutation using this uniform interpretation (true literals underlined) is

(4+2 =5): ¬P (5+1=6): Q (6+3+4): [ ]

Notice all electrons and resolvents are false in I.

We now ***rename*** literals in the original clauses so that under the standard interpretation (when all atoms false) electrons are false and nucleii are true.

Rename P to ¬P' and R to ¬R'. Since I makes P and R true it makes P' and R' false, and it still makes Q false. (¬P becomes P' and ¬R becomes R')

Then: (1) ¬P'∨Q (*N*) (2) P'∨¬R' (*N*) (3) ¬Q∨¬R' (*N*) (4) R' (*E*)

A standard HR-refutation is (4+2 =5): P' (5+1=6): Q (6+3+4): [ ]

which is exactly the same as the original refutation after renaming.

SUMMARY: For each atom that I makes false - do nothing;
For each atom A that I makes true rename A as ¬A' (so I makes A' false)
Then can use standard hyper-resolution

How might a suitable uniform interpretation I be found?

- If clauses T are a *theory about something or other*, they will be satisfiable and have a model M. With respect to M, the clauses in T are nucleii.
- Suppose a conclusion C, when negated, yields clauses G.

If T $\models$ C then T+ ¬C is inconsistent, and the clauses T+G are not satisfiable. That is, any model M of T must make at least one clause in G false and that clause would be an electron with respect to M.

- So we look for a *uniform* I such that (if possible)
  - nucleii = clauses in T   (I makes clauses in T True)
  - electrons = clauses in G   (I makes clauses in G False)

then hyper-resolution allows resolution between a clause in T and clauses in G or derived from G, but **never** between two clauses in T.

It is unlikely this can be achieved exactly; for any selected I, usually at least some clauses in G are nucleii or some clauses in T are electrons, but the idea is good - divide clauses into 2 sets, called the *major* and *minor* sets, such that resolvents must be formed using at least one clause derived from the major set and clauses in the minor set are never resolved together.

- The *set-of-support* strategy considered earlier is the result.

Example

Show $\forall xy[(x<y \vee x=y) \wedge y<c \rightarrow x<c]$ (Use E for = and L for <)

1.	Exx	$x=x$	2.	{¬Exy,¬Eyz,Exz}	$x=y \wedge y=z \rightarrow x=z$
3.	{¬Exy,Eyx}	$x=y \rightarrow y=x$	4.	{¬Lxy,¬Lyz,Lxz}	$x<y \wedge y<z \rightarrow x<z$
5.	{¬Lxy,¬Lyx}	$x<y \rightarrow \neg y<x$	6.	{Lxy,Exy,Lyx}	$x<y \vee y<x \vee x=y$
7.	{¬Exy,¬Lxy}	$x=y \rightarrow \neg x<y$			
8.	{Lab,Eab}	9. Lbc		10. ¬Lac	

negated goal is $\equiv \exists xy[(x<y \vee x=y) \wedge y<c \wedge \neg x<c]$
which becomes after Skolemising $(a < b \vee a=b) \wedge b < c \wedge \neg a < c$ (i.e. 8,9,10)

Let I = {Exy all True, Lxy all false.}; (a uniform interpretation)
Then all except 9 are nucleii with True literals positive E or negative L.
9 is the only electron. Electrons have negative E and positive L literals

The True literals in nucleii for resolving are underlined - all the resolvents are electrons so no true literals.

11. (7+9).	¬Ebc	16. (7+15).	¬Eba
12. (2+11).	{¬Eby, ¬Eyc}	17. (3+16).	¬Eab
13. (12 +6).	{Lyc, Lcy, ¬Eby}	18. (8+17).	Lab
14. (13+10)	{Lca, ¬Eba}	19. (18+4+9).	Lac
15. (14+4+9)	{Lba, ¬Eba}	20. (19+10)	[]

Generalising Hyper-resolution.

As Slide 8ci argues, hyper-resolution can be generalised. One interesting and simple generalisation is to exchange the roles of positive and negative literals. We'll call a clause with only negative literals an N-electron and a clause with at least one positive literal an N-nucleus. Then all positive literals in an N-nucleus are resolved with N-electrons in an N-hyper-resolution step. A special case of this occurs for Horn clauses – all N-nucleii will have just one positive literal, so each step is a single resolution step and the result is, in effect, a simulation of a logic programming system. **Exercise**: Show that this is the case. Slide 8cii shows the relation between hyper-resolution and N-hyper-resolution.

Slide 8cv shows an example of a *uniform* interpretation. A *non-uniform* interpretation for the given clauses on 8cv is the following one:
    Lxx = false, Lab=Lbc=Lca=true and Lba=Lcb=Lac=False.
    Exx = true and Exy = false if x≠y.
For this interpretation consider clause 3 on Slide 8cv. It is always a nucleus and ¬Exy is always true, except if x=y. In that case the clause becomes a tautology, which is perhaps not too bad. Now consider clause 5. It also is always a nucleus, but when x/a, y/b, then ¬Lyx is true, whereas when x/b and y/a , then ¬Lxy is true. So sometimes the nucleus should be used by resolving on ¬Lxy and sometimes on ¬Lyx. But if resolving with Luv (say) how can you tell which to use? Perhaps this too is a special case as it is a symmetric clause. So consider clause 4. The instance ¬Lab∨¬Lbc∨Lac is an electron as it is false, but the instance ¬Lab∨¬Lba∨Laa is a nucleus, with ¬Lba (from ¬Lyz) the true literal.

Because of these kinds of problems, non-uniform interpretations are not often considered.

Relation between Standard Hyper-resolution and using any Uniform Interpretation.

If you were to attempt hyper-resolution for the clauses on Slide 8cv using the non-uniform interpretation suggested on slide 8cvi, it would be difficult. You might resolve on a literal in an electron only to find later that the implicit instance used was a nucleus! It is for this reason that uniform interpretations are the best. Even then, they may not be so efficient for humans or machines to process as the standard interpretation.

A uniform interpretation may appear to be a useful generalisation, but in fact, any proof found using it is isomorphic to an ordinary hyper-resolution proof, and so the generalised version using uniform interpretations is complete too. This is shown next.

In a uniform interpretation, for any literal occurring in a clause either all its instances will be true or all its instances will be false. Thus each literal in a clause can be given a label t or f and it can be determined whether a clause is an electron or a nucleus, and, in the latter case, which literals are true. (In the standard interpretation, where all atoms are assigned false, each negative literal is true and each positive literal is false.) Let I be a uniform interpretation. We can rename each atom L, such that L is true in I, as ¬L' for some new L', which will be false in I, and hence all instances of ¬L' will be true. That is, L' is equivalent to ¬L and for the renamed atoms, each occurrence of the literal ¬L will be replaced by L' and each occurrence of L will be replaced by ¬L'. Atoms that are already false in I are not affected. I now makes all atoms false (the unaffected ones were already false and the renamed L' type of atoms were constructed to be false). The clauses that were nucleii are still nucleii (but now in the usual sense) and the same literals are true in them. For example, if C was a nucleus and included atom S, which was true in I, then S will be replaced by ¬S', where S' is false in I. Similarly, the clauses that were electrons are still electrons (now in the usual sense). See Slides 8cii and 8ciii for an example. (Continued on Slide 8cviii.)

(**Continued from Slide 8cvii**).
Each step in the original derivation (using the original uniform interpretation) corresponds exactly to a step in the renamed version (made using the standard rules).

On the other hand, after the transformation the standard hyper-resolution strategy can be used to find a refutation, always possible since the standard strategy is complete. Since only syntactic changes have been made, a refutation from the original clauses respecting the original interpretation, if desired, can be recovered from the standard one that is found using the transformed clauses. (See Slide 8ciii again.)

As another example, in the particular case when the interpretation makes every atom true, all literals must be renamed. Any Horn clause with one positive literal now becomes a nucleus with one negative literal, and any *negative clause* with no positive literals becomes an electron. Since only one negative clause is necessary for a Horn clause refutation, the resulting electron may be the only one.

There is at least one class of problems when a non-uniform interpretation can be useful. It is when the clauses represent a problem with a *sorted* Herbrand universe. That is, the terms are divided into disjoint subsets (called *sorts*). For example, a predicate A(x,y) may occur in two kinds of places - those in which x and y would both be bound to elements of one sort, or those in which x and y would both be bound to elements of another sort. It may also be the case that when x and y were of different sorts A(x,y) would never participate in a successful refutation. Then, we might be tempted to divide the clauses containing A(x,y) or its instances into those in which x and y belong to the first sort and those in which x and y belong to the second sort and to ignore the others. Then, if required, A(x,y) could be made true for the first sort and false for the second, or vice versa. An example of this idea is described on 8cix.

For Interest Only:
Example of a non-uniform interpretation making use of a sorted universe.

Let the signature = <{A,B,C}, {g}, {a,b,c,k,m}> and A and B be of arity 1, and C be of arity 2. Also, let g, of arity 1, return sort 1, whatever its argument, a and b be of sort 1, and c, k, m be of sort 2. Assume arguments of A, B and C can be of either sort.

Consider an interpretation in which A(x) is assigned true iff x is of sort 1 and B(x) is assigned true iff x is of sort 2 and suppose an assumption is made that the only useful instances of the clause $\neg A(x) \vee \neg B(y) \vee C(x,y)$ are when x is of sort1 and y is of sort 2. In this clause C(x,y) instances can be assumed to be of the kind C(sort1, sort2) and all these can be given the same truth value. On the other hand, in $\neg A(x) \vee \neg A(y) \vee C(x,y)$ perhaps the useful instances can be assumed to be of kind C(sort1 ,sort1), which can also all be given the same truth value. but not necessarily the same one as was given to the C(sort1,sort2) instances. Thus in transforming from the non-standard interpretation to the standard interpretation by renaming, as illustrated on Slide 8cviii, C(x,y) might be renamed in one clause (say the first one), but not in another.

If the predicates A and B were not explicitly in the sorted signature the singleton clause C(x,y) could still be split into two clauses as above, by implicitly introducing the predicates A and B in order to control the sorts of the arguments in the clause. Applying this approach to the so-called “**Steam-roller**” problem, makes it quite easy, whereas otherwise it appears hard for theorem provers. See more about this in Problem Sheet.

Summary of Slides 8

1. Hyper-resolution is a refinement in which each step consists of one or more linked resolution steps made according to a specific restriction.

2. Each clause is classified either as an Electron (no negative literal) or as a Nucleus (at least one negative literal). Each step is made by resolving one nucleus clause with one or more electron clauses in turn, until the resolvent is an electron. Only electrons are generated by hyper-resolution.

3. Otter is one of a famous family of theorem provers which uses the Hyper-resolution strategy, together with the set-of-support strategy. In searching for Hyper-resolvents, at least one clause in the linked resolutions making up a hyper-resolution step must be derived from the so-called SOS.

4. The SOS strategy allows a user to prevent the theorem prover making potentially non-useful steps by resolving clauses from some satisfiable theory.

5.The set-of-support strategy can be ``explained’’, or "derived" by considering a generalisation of hyper-resolution to use uniform interpretations. In fact, the use of arbitrary, but uniform, interpretations is equivalent to standard hyper-resolution.

6. Standard hyper-resolution is based on the H-Interpretation which assigns False to every ground atom. Nucleii are true in this interpretation and electrons are false. Other uniform interpretations can assign either true or false to atoms, restricted so that, for each literal occurring in the given clauses, all instances are assigned the same value. Nucleii are clauses that are True under the uniform interpretation and electrons are clauses which are false. Uniform interpretations with at least 1 each of a nucleus and electron can always be found for refutable sets of clauses, but not always for satisfiable sets of clauses. (Why?)

7. Standard hyper-resolution is **complete**. One proof uses an amended semantic tree argument (see Chapter notes), another uses an induction proof (see solutions to Problems).

8. Predicate Ordering can be combined easily with Hyper-resolution. Each electron is restricted in its use to resolving on the lowest atom in the order.

9. Locking also can be combined with hyper-resolution. Both extensions, of Predicate ordering and locking, can be shown to be complete by choosing a suitable set of indices for literals so that a locking refutation produces a simulated hyper-resolution refutation. There is no need to index negative literals as they muct be resolved in each step anyway. (See Problems for examples and discussion.)