AUTOMATED REASONING

SLIDES 11:

ASPECTS OF TABLEAU THEOREM PROVING

Controlling Backtracking
Universal Literals in Model Elimination
Model Generation and Tableaux

KB-AR - 12

Variants and extensions to Model Elimination 11ai

In these slides we consider some extensions and alternatives to Model Elimination;

1) *Variation in the search mechanism*: The method of removing potentially redundant backtracking (called non-essential back-tracking by the author) has been proposed by Jens Otten "Restricting Backtracking in Connection Calculii" (2010). Although the method is not complete, it has proved very effective in practice. A large proportion of problems can be solved with the restriction, and the average saving in search time allows for more complex proofs to be found that would not be found by standard model elimination in a reasonable time. Shown next.

2) *Universal Literals*: When discussing Re-Use we saw that in first order ME it may be possible to derive universal lemmas of the form ∀z.R(z), which can be used elsewhere in the tableau. Such universal literals can arise in other ways and we discuss how to exploit this as shown in Slides 11b.

3) The relation between Clausal Tableau and Model Generation (MG) of slides 2 is revisited. See Slides 11c.

4) In the slides 9-11 Appendix 2 there are two Optional Case Studies:
Case Study 1 - KE Tableaux: This variation of tableau uses a single splitting rule;

Case Study 2 - Intermediate Lemma refinement (ILE): This is a variant of model elimination

---

Backtracking in ME (also see ppt) 11aii

Searching for a closed tableau in ME employs a limit on the size of the tableau (called depth-bound search) – e.g. maximum branch length.

Normally, on failure of some step, backtracking tries the next available step:
Either:
- i) if branch closure led to failure, try a different way to close branch
- ii) if no different ways, try branch extension
- iii) if extension led to failure try a different way to extend
- iv) if no different extensions backtrack to branch on the left and look for a different derivation leading to a closed tableau
- v) if no branches on the left try to backtrack to parent node
- vi) if no parent node try a different top clause

Else FAIL

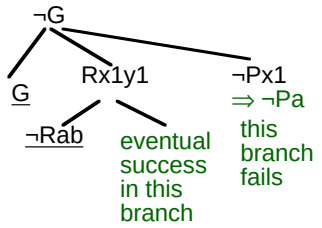
Otten (2010) saw that in trials with the problems in the TPTP database (Thousands of Problems for Theorem Provers), many problems could be solved even if case iv) is prohibited.

Although completeness is lost, a dramatic decrease in time to find proofs is gained. He coined the phrase "essential backtracking".

Essential and non-essential backtracking (1) 11aiii

Example of "non-essential backtracking" (as named by Otten):
¬G, G ∨ Rxy ∨¬Px, Pc ∨..., Pd ∨..., ¬Rab ∨ ..., ¬Rbd ∨ ..., G ∨ ...
top clause ¬G

Assume eventual closure below Rx1y1 using ¬Rab ∨ ... with x1==a,y1==b
¬Pa will fail and normally would backtrack to use ¬Rbd ∨ ... **(Case iv)**
Instead, *non-essential backtracking prohibits this and backtracks to try a different clause to use in extension step from ¬G* ***(Case vi)***



In fact, nothing is lost in this example as using ¬Rbd ∨ ... instead leads to ¬Pb which also fails.

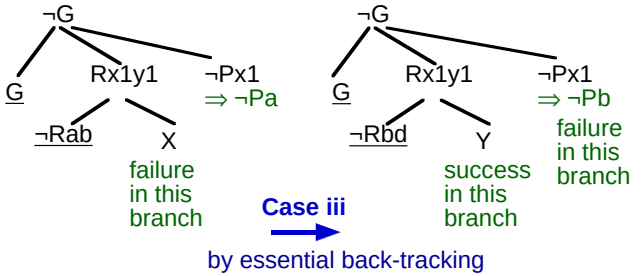
But if a clause such as Pb ∨ ... were available, and if the tableau below ¬Pb happened to close, then the particular fully closed tableau so found would be lost by non-essential back-tracking

Essential and non-essential backtracking (2)

11aiv

Example of "essential backtracking" (as named by Otten):
¬G, G ∨ Rxy ∨¬Px, Pc ∨..., ¬Rab ∨ X, ¬Rbd ∨ Y, ¬Rcd ,
top clause ¬G

Assume no eventual closure below Rx1y1 using ¬Rab ∨ X then backtrack
(essential backtracking) to use ¬Rbd ∨ Y **(Case iii)**
Suppose closure obtained with binding x1==b,y1==d **(Case iv)**
Suppose failure beneath resulting ¬Pb (backtracking to Rx1y1 is non-essential)
Hence no back-tracking to use ¬Rcd, even though ¬Pc might succeed



Do not backtrack to try ¬Rcd because it's non-essential back-tracking

Try to match ¬G with another clause; no clause so fail

Lose solution using ¬Rcd

11av

Formalising Essential Backtracking and Non-Essential Backtracking:

Consider a Model Elimination derivation that is part completed, such that the next leaf node to be extended is L in branch B.
Suppose also that L could close using any of the literals at N_1, N_2, ..., N_k in B (above L) and can be extended using any of the matching clauses R_1, R_2, ..., R_m.

Then Essential Back-tracking allows every one of the N_i and the R_i to be tried to close the tree. This is the normal kind of systematic back-tracking.

Non-essential Back-tracking truncates the list of choices as soon as one of them succeeds in closing the sub-tableau beneath L.

As a result, if other branches in the tableau to the right of B fail to close, then there is no back-tracking to try a different choice at L.

In the Example on 11aiv, beneath Rx1y1 the matching clauses are ¬Rab ∨ X, ¬Rbd ∨ Y and ¬Rcd. ¬Rab ∨ X fails and so ¬Rbd ∨ Y is tried. This succeeds, but although the last branch below Px1 (now Pb) fails, non-essential backtracking has truncated the choices so ¬Rcd is no longer available to try as an alternative beneath Rx1y1.

Essential and non-essential backtracking (3)

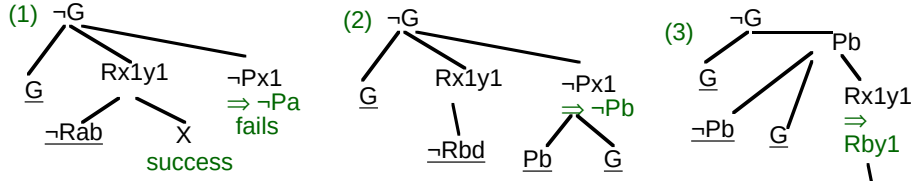
11avi

Consequences of ignoring non-essential backtracking:
- Loss of completeness (See 11aiv)
- Irrelevant back-tracking is by-passed in a simple way (See 11 aiii)
- Proof search is therefore much faster, allowing harder proofs to be found

Why does this back-tracking restriction not lose many proofs?

There are usually several different ways to make a ME derivation from a set of clauses, each derivation differing in the order in which the clauses are used, and/or the instances used. Disregarding non-essential back-tracking simply searches for one of the other orders, or one of the other derivations.

Example. Given: ¬G, G ∨ Rxy ∨¬Px, Pa ∨..., G ∨ Pb, ¬Rab ∨ X, ¬Rbd
Assume Closure below X is ok and also Closure below ¬Pb is ok



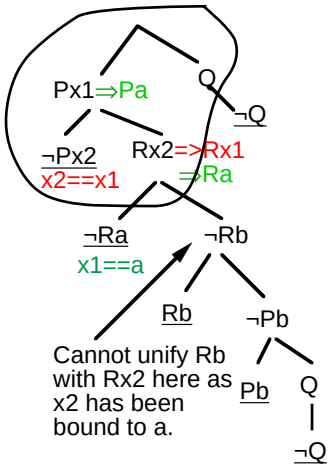
(1) fails and does not find (2) as Case iv is prohibited;
after changing match for ¬G (3) succeeds by Case vi

An Extension to Basic ME Tableaux (1)

11bi

Example Given: 1. Rx ∨¬Px, 2. Px ∨Q, 3. ¬Ra ∨¬Rb, 4. ¬Q

Standard ME Tableau **vs.** Resolution Refutation



(5 = 2+1): Rx ∨Q
(6 = 5+3): ¬Rb ∨Q
(7 = 6+5): Q ∨Q ⇒ Q
(9 = 8+4): [ ]

Notice that clauses 5 and 6 correspond to the leaf literals of the open branches after each of the first 2 steps of ME.

Compare step 3 (giving (7)) with ME:

Instead of standard ME tableau below ¬Rb (as shown) try repeating enclosed part of the tableau below ¬Rb but with new free variables x3 and x4, which will bind to b instead of a.

(See next slide)

An Extension to Basic ME Tableaux (2)

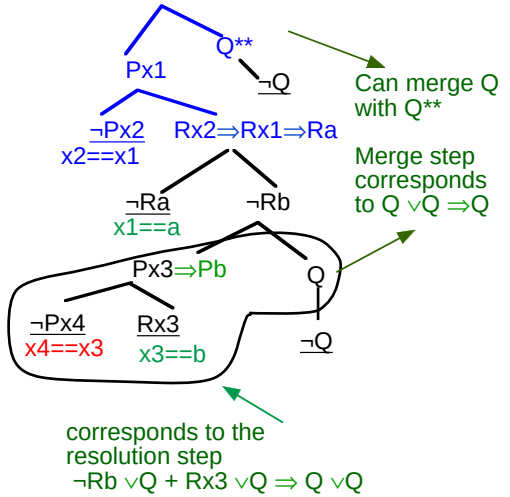
11bii

Example Given: 1. Rx ∨¬Px, 2. Px ∨Q, 3. ¬Ra ∨¬Rb, 4. ¬Q

Instead of standard ME tableau below ¬Rb try repeating 1st 2 steps below ¬Rb (blue part of the tableau) but with *new free variables x3 and x4*, which will become bound to b instead of a.

In some cases can avoid actual duplication. e.g. here would close at the leaf ¬Rb (implicitly matching a second copy of Rx1). Results in the generalised closure rule.

Notice that x1 does not occur in any open branch to the right of the branch ending at ¬Rb. In the fresh instance of the enclosed tableau the branch below Q can be closed by a merge with Q**, so simulating the resolution proof.



An Extension to Basic ME Tableaux (3)

11biii

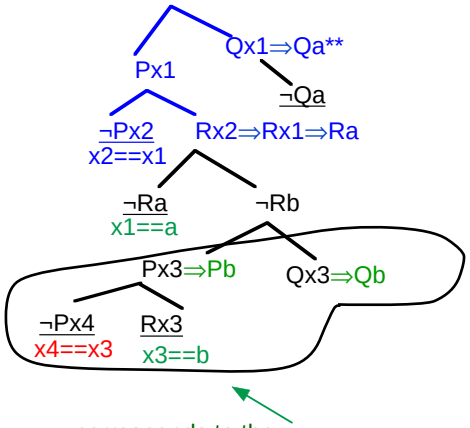
Given: 1. Rx ∨ ¬Px, 2. Px ∨ Qx, 3. ¬Ra ∨ ¬Rb, 4. ¬Qa

In some cases cannot avoid actual duplication.

Notice that x1 now occurs in two branches, including the open branch to the right of the branch ending at Rx1, in Qx1.

In the fresh instance of the enclosed tableau the 2nd branch below ¬Rb becomes Qb, which doesn't merge any more with Qa.

It is incorrect to avoid the duplication as the branch below Qb would be lost leading to the wrong result, since this branch fails.

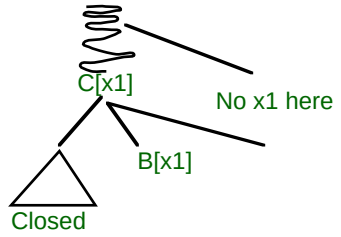


Generalised Closure Rule and Universal Variables (1)

Let T be a partially developed ME tableau and B be an open branch of T.
If free variable x1 occurs in some literal in B and all other occurrences of x1 in an open branch are also in B, then x1 is called *a universal variable in T*.

Features of Universal Variables:

- bindings to x1 cannot affect literals in other open branches;
- once a free variable becomes universal it cannot lose this status as long as the convention for bindings on closure given on Slide 11cvi is followed;



- we'll see that a Universal variable can be treated as if it were universally quantified - as many (different) copies as may be needed are available implicitly;
- as a result any bindings made to a universal variable can be **ignored**, leading to the Generalised Closure Rule

11ci

Generalised Closure Rule and Universal Variables (2)

Let T be a partially developed ME tableau and B be an open branch of T.
If free variable x1 occurs in some literal in B and all other occurrences of x1 in an open branch are also in B, then x1 is called a *universal variable in T*.

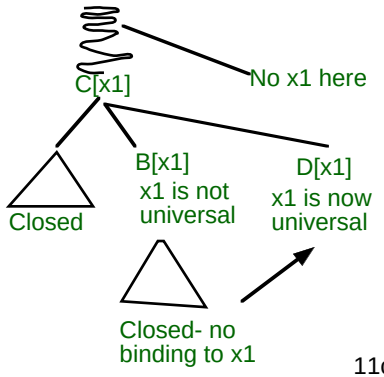
Features of Universal Variables:

• a free variable x1 might not be universal in T when first introduced into a tableau T, but can become so if x1 eventually occurs in a single open branch;

e.g. in S ∨P(x1) ∨Q(x1) ∨R if the branch below P(x1) is closed without binding x1, then x1 in Q(x1) becomes universal.

• some variables are always universal in a clause and hence also when used in a tableau e.g.:

y is universal in S∨P(x,y)∨Q(x)∨R



11cii

11ciii

Generalised Closure Rule:

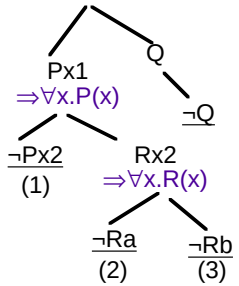
The slides 11b and 11c illustrate and explain an extension for ME-tableaux called the *generalised closure rule*, which exploits the concept of a *universal variable*. For the simplest case, let C be a clause in which variable u occurs in exactly one literal L. u is called a *universal variable*. The quantifier for this variable could (implicitly) be distributed across to L. When the clause is developed, one can treat L as if it were ∀u.L, implicitly including in the tableau branch containing L several copies of L, each with a fresh free variable for u. (Any non-universal variable in L would be substituted by exactly one free variable, the same one in all copies.) The effect is that bindings to u can be ignored since there are available enough copies for each different binding, giving rise to the *generalised closure rule*. *This rule states that in any closure step involving a universal variable u possible bindings to u can be ignored.*

e.g. if L is P(v,u), and u is universal but v is not, then L is regarded as if it were ∀u.P(v,u) and can be copied in the tableau branch as P(v1,u1), P(v1,u2), etc. for example giving possibility of closure with ¬P(a,b)∨¬P(a,c). If L is part of a clause such as L∨ Q(w), the duplication treats the clause as if it had the more general (nnf) form of (P(v,u1)∧P(v,u2))∨Q(w).

More generally, let T be a partially developed ME tableau and B be an open branch of T. *If free variable x1 occurs in some literal in B and all other occurrences of x1 in an open branch are in B,* then x1 is called a universal variable in T.

Example Revisited (1)

11civ



Rx ∨¬Px, Px ∨Q, ¬Ra ∨¬Rb, ¬Q

In Px ∨Q, x is universal.
The clause Px1∨Q is used in the tableau **as the equivalent ∀x.P(x) ∨Q**.

At closure (1) use instance P(x2) of ∀x.P(x).
Rx2 becomes universal and is equivalent to ∀x.R(x).

At closure (2) use instance R(a) of ∀x.R(x).

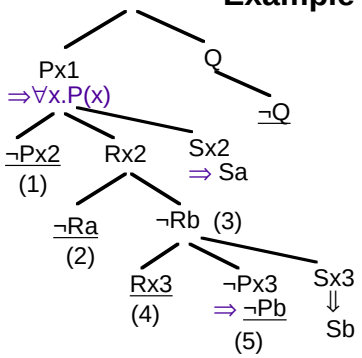
At closure (3) use instance R(b) of ∀x.R(x).

None of the bindings made affects the branch beneath Q, since the universal variables do not appear in it. That branch only needs to be completed once.

Implementing Universal Variables:
Either tag a universal variable and always use a fresh copy, (so here it is x2 that is bound to the fresh copy of x1 - i.e. "fresh copy" ==x2 - not the other way around) or replace by the universal quantifier (easier when working by hand).

Example Revisited (2)

11cv



Rx ∨¬Px ∨ Sx, Px ∨Q, ¬Ra ∨¬Rb, ¬Q

After closure (1) Rx2 does **not** become universal (x2 occurs in sibling Sx2)

At closure (2) use x2==a

At (3) must use a fresh copy of clause Rx ∨¬Px ∨ Sx

At (4) bind x3==b

At (5) can use an instance of ∀x.P(x)

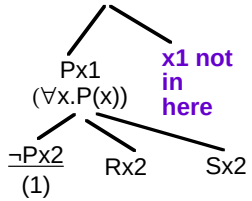
Variable x in Rx ∨¬Px ∨Sx is not universal, so important not to bind x1 to x2 at closure (1) - i.e. NOT x2==x1 - as x1 would then lose its universal status. Instead, use a fresh instance of x1 and bind x2 to that - i.e. "fresh copy==x2. (See next slide)

- **Exercise (Not easy!)**:

Consider whether and how use of the generalised closure rule could be used in a tableau together with either the (re-use) and/or the (merge) closure rules.

An Important Criterion (when implementing GCR)

11cv



In making a closure that involves at least one universal variable always introduce a fresh copy of the universal variable to make the closure.

Example A:

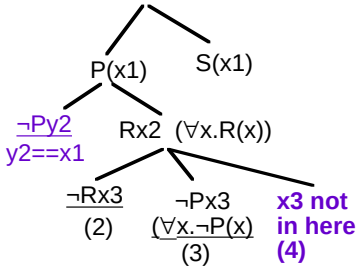
In normal ME, to close at (1) either make binding x1==x2, or x2==x1; it doesn't matter which.

In GCR it is important to introduce a fresh copy of x1 and to bind x2 to this fresh copy. That is "fresh copy"==x2. Achieved by tagging x1 as universal and always using a fresh copy.

If x2 and x1 were simply bound to each other, then x1 would in effect be propagated to both Rx2 and Sx2 and x1 would lose its universal status.

An Important Criterion (when implementing GCR)

11cvi



In making a closure that involves at least one universal variable always introduce a fresh copy of the universal variable to make the closure.

Example B: when closing at (2), remember to use a fresh copy of implicit ∀x.R(x) and set "fresh copy"==x3; then at (3) it is important to use a fresh copy of implicit ∀x.¬P(x) and set "fresh copy"==x1. Otherwise, x2 and x1 would be bound to each other via x3 and x2 would lose its universal status, which would affect the literal Rx2 as well and could compromise the proof beneath (4).

The observations in Example A and B indicate to use the above convention which will guarantee universal variables remain so.

Generalised Closure Rule – Criterion for Maintaining maximal Universality of Variables:

Example: Let x be a universal variable in a leaf of branch B of tableau T, say in Q(x), and some step use the clause instance ¬Q(z1)∨ R(z1)∨ S below it. One of the implicit instances of ∀x.Q(x) is Q(x1) and x1 can be bound to z1. In the literal R(z1), z1 will now be universal, since if x was universal then no occurrences of x occurred in T other than in B, and the same applies to the extension of B ending in leaf node R(z1) (branch B', say). In effect, ∀z.R(z) has been derived in B'. To see this, add the negation ¬∀z.R(z) to the tableau and see that the tableau closes if S closes. A variable becomes universal in this way in a Model Elimination tableau if it does not occur in any leaf literals in open branches to its right in the tableau. In this example that is the case, as explained.

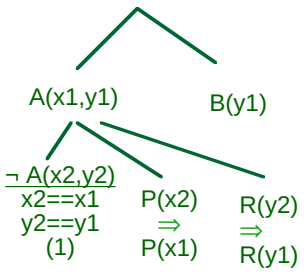
It is assumed that as construction of a tableau progresses variables are implicitly marked as universal whenever possible, in the manner described in the previous example. That is, a free variable x occurring in leaf literal L in an open branch B is marked as *universal* if the only occurrence of x in a leaf literal in an open branch is in L. Observe that (non-universal) occurrences of x could only occur in the branch above L if the occurrence of x in L was originally z (say) and arose because z was bound to x by a clause used in the closure of one of L's left (closed) siblings. Note x would not be classified universal in L in this case. An example (where x is the variable y2) is on slide 11bx, when ¬A(x2,y2) closes with ∀x.A(x,y1), binding y2 to y1. All occurrences of y1 remain non-universal.

Exercise: The criterion for choosing closure bindings is: *In making a closure involving ≥1 universal variables introduce fresh copies of the universal variable(s) to make the closure.* Explain why this assures that any variable declared universal will remain so during subsequent tableau development.

11cvii

Normal Form Representation of a Partial Tableau (or Why does the Generalised Closure Rule work?) 11cviii

Given: (1) ¬P(a)∨ ¬A(b,z) (2) ¬R(c)∨ ¬A(c,c) (3) A(x,y)∨B(y) (4) ¬A(x,y)∨P(x)∨R(y)



Consider the standard ME tableau after closure (1)
The open branches ≡
∀y1∀x1[(A(x1,y1) ∧ (P(x1) ∨ R(y1)) ∨ B(y1)] ≡
∀y1 [(∀x.A(x,y1) ∧ (∀x.P(x) ∨ R(y1)))∨ B(y1)] ≡
∀y[(∀x.A(x,y) ∧∀x.P(x))∨ (∀x.A(x,y) ∧R(y))∨ B(y)]

That is, maximally distribute ∀.

Using universal variables treats universal literals as quantified literals as in the above expression.

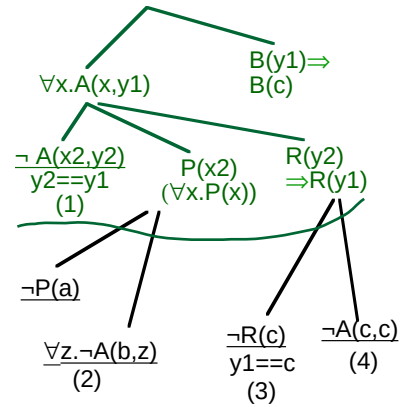
You can see that x1 in A(x1,y1) and in P(x1) are universal, and that y1 is not universal.

See next slide for rest of tableau.

Continued from Slide 11bx 11cix

(1) ¬P(a)∨∀z.¬A(b,z) (2) ¬R(c)∨¬A(c,c) (3) ∀x.A(x,y)∨B(y) (4) ¬A(x,y)∨P(x)∨R(y)

After closure (1) (above the wavy line), and making universal variables explicit, the open branches ≡ ∀y[(∀x.A(x,y) ∧∀x.P(x))∨ (∀x.A(x,y) ∧R(y))∨ B(y)] (*)



Variables z and x in givens (1) and (3) are universal, as indicated.

After closure (1) can ``**forget**'' the bindings to universal variable x in ∀x.A(x,y1); also x2 in P(x2) becomes universal.

At (2) instances A(x3,y1) and A(b,z3) are unified (x3, z3 being the fresh copies of x and z) leaving y1 unbound.

At (3) y1 is bound to c and at (4) ¬A(c,c) unifies with A(x4,c), x4 the fresh copy of x.

The last open branch contains B(c). Notice B(c) can be derived from (*), (1) and (2).

Soundness of the Generalised Closure Rule:

Justification that the generalised closure rule is sound can be made by appealing to the expression represented by the open part of a partial tableau, distributing quantifiers maximally across literals containing universal variables.

The ``open part" of any tableau (i.e. the set of branches not yet closed) typically represents a universally quantified formula in dnf; i.e. a disjunction of (possibly quantified) conjunctions of literals. This was illustrated on slide 11cviii. Suppose a new clause is added to the leftmost open branch. Assume that non-universal variables in the added clause are always renamed as fresh free variables.

When a new clause is added in an extension step there are two options for forming the binding in the closing unifier for a variable x: either x is universal and a fresh copy of x becomes bound, or x is not-universal and is bound in the normal way. Thus e.g. when matching $\forall x.A(x,y1)$ with $\forall z.\neg A(u1,z)$, the universal fresh copy x2 of x in A(x2,y1) is bound to u1, and the fresh copy z2 of z in $\neg A(u1,z2)$ is bound to y1 (ie "fresh copy–x"==u1 and "fresh copy_z"==y1). The open part of the extended tableau can be recomputed and universal quantifiers distributed to maximise occurrences of universal variables.

Assuming the criterion on slide 11cvi is adhered to, it is not hard to show that universal variables remain so. Given the dnf representation of the open part of a partial tableau (call it dnf(T)), it is then easy to show that if T" is derived from T then dnf(T)|=dnf(T").

11cx

Summary of Advantages of ME-style tableaux

11di

- Prolog-like - use stacks for implementation (but: need to detect ancestors, and use the occurs check)
- Prolog technology: compilation, structure sharing, stack maintenance
- Easy to obtain variations
- Easy to implement in Prolog
- Easy to extend to modal logics (and others)

Disadvantages

- Let the original problem be Data $\vdash$ C, where C = P $\rightarrow$ Q. Clausal form of $\neg$(P $\rightarrow$ Q) is P and $\neg$Q. May want to work forwards from P and "backwards " from $\neg$Q. In ME (but not KE) must choose one of them as top clause.
- May be beneficial to resolve some clauses initially if they only resolve with one or two others. i.e. a non-linear beginning.
- As in Prolog, L-R depth first generation of search space may not give the smallest one.
- Quite often the search space contains several variations of the same refutation, in which the clauses are used in different orders. (However, it is this property that makes non-essential backtracking removal a good strategy.)
- In case interest is in finding models, ME tableaux can also help, but not as good as special model checking techniques - e.g. transitivity/symmetry axioms can cause digressions if they occur in the data.

Model Generation (MG) as a Tableau refinement

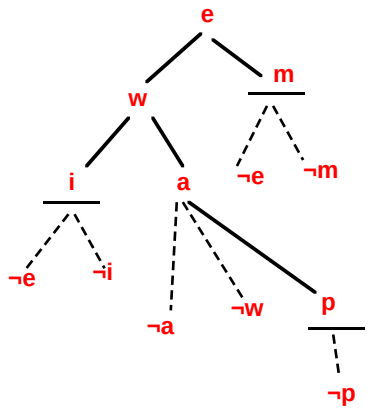
11ei

- Recall (from week 1) the form of a MG tree for checking the existence of models for propositional clauses
- Similar to the tableau method MG looks for models in each branch

Example

$\neg p$ e $\neg e \vee \neg c$ $\neg a \vee \neg w \vee p$ $w \vee m$ $i \vee a$ $\neg e \vee \neg m$

- Top clause is always positive ....
- Can extend a branch by clause C as long as negative literals in C (if any) are complemented in the branch
- All literals in the MG tree (solid lines) are positive
- Leaf literals, in the tableau, (dotted lines) are negative literals
- Differences are mainly cosmetic



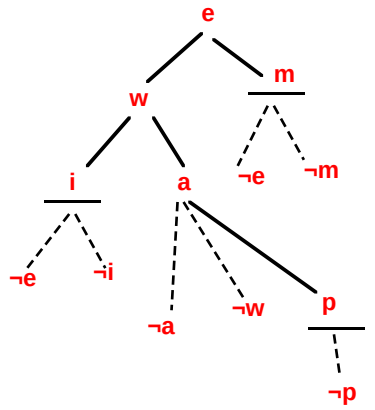
Correctness of MG using correctness of Tableaux (1)

Can we perhaps use the soundness and completeness properties of tableau to show similar properties for MG? **Yes, we Can!**

Soundness is the easiest
If a MG tree for clauses S returns False, then S |= $\perp$

If there is a closed tree, it must be because all branches end by using a negative clause, where all its literals are complemented.

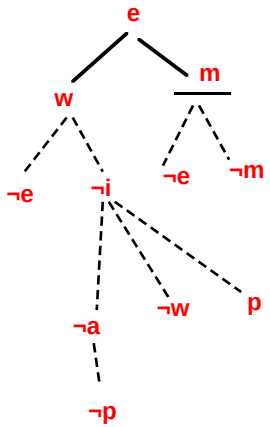
The implicit tableau would also be closed, and so we use Tableau Soundness to conclude S |= $\perp$.



11eii

Correctness of MG using correctness of Tableaux (2)

Completeness uses finiteness of MG tree
For each open branch B of a MG tree
We know that:
All clauses with no negative literals are used
All unused clauses have negative literals



if B were to be extended by any clause with negative literals not yet used in B, there is one extended branch of B which remains open, since all such clauses have one negative literal not matched in B

The open branch is saturated and will yield a model

Example: Remove i ∨ a
Open branch (in tableau version of MG) is {e, w, ¬i, ¬a, ¬p}
Note - dotted lines are not part of MG tree which ends at w

11eiii

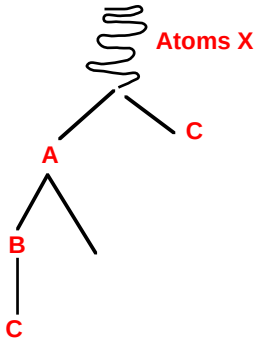
Short cut for MG

11eiv

If (ground) MG tree is developed left -> right

And current open branch includes atoms in an open branch on its right (still to develop)

Then can abandon current open branch



See diagram:

If branch containing C (and atoms in X) has no model, nor will branch containing {A, B, C} and atoms in X have a model

If branch containing C and X does have a model no need to search branch {A,B,C} and X

Question:
Can MG method be generalised to first order clauses?
Consider case when signature has no function symbols

Either: use analogy with free variable tableau
Or: implicitly maintain ground instances of each clause
This is a coursework question!

Summary of Slides 11

11fi

1. The Model Elimination (ME) tableau method can be extended and modified in various ways.

2. The introduction of universal variables and the generalised closure rule that results leads, in many cases, to reduced tableaux. Universal variables are treated in the tableau as being universally quantified, instead of as free variables, so multiple instances are allowed. This leads to the generalised closure rule, which in practce means bindings to universal variables can simply be ignored.

3. A modification of ME, which is related both to logic programming (extending that approach to non-Horn clauses in a simple way) and to hyper-resolution (generating all-positive lemmas), is the Intermediate Lemma Extension.

4.The Intermediate Lemma Extension tends to produce many, but small, search spaces. Even though there is a fair amount of re-computation in the search, the limit to the search space size limits the computation.

5.A variation of the tableau method, called KE-tableau, can be viewed as a generalisation of the Davis Putnam procedure to arbitrary sentences, including first order.

6. KE-tableau have been proved to be computationally more efficient than the standard tableau method. However, although much work has been done on propositional theorem proving in KE, little has been done for first order theorem proving. Thus there are no well known extensions for the method, analogous to the generalised closure rule, for instance.

7. ME style tableau have advantages, especially in that they are easily implementable in Prolog, which is good for testing new ideas. All the Prolog technology is available to build good systems. The method extends to other logics, e.g. modal logic.

8. ME style tableau have disadvantages, mainly related to their being linear.

Question for next week

What do you understand by statements such as

“1+1 = 2”, or “father(william)=charles”?