AUTOMATED REASONING

SLIDES 11:

ASPECTS OF TABLEAU THEOREM PROVING

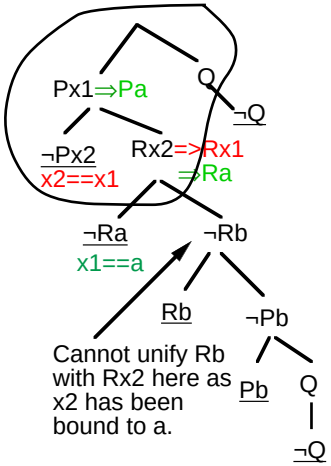
Universal Literals in Model Elimination
KE-tableaux
Intermediate Lemma Extension

KB-AR - 09

An Extension to Basic ME Tableaux (1)

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

Standard ME Tableau



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.

Comparing step 3 with ME:

Instead of standard ME tableau below ¬Rb 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)

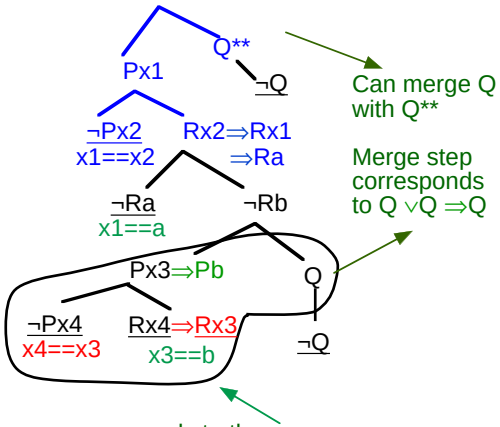
An Extension to Basic ME Tableaux (2)

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

Instead of standard ME tableau below ¬Rb try repeating blue part of the tableau below ¬Rb 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 Rx2). Results in the generalised closure rule.

Notice that x2 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.



Generalised Closure Rule and Universal Variables

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 in T:

- 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 11av is followed ;
- 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,y) ∨Q(x1) ∨R if the branch below P(x1,y) is closed without binding x1, then 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 ;

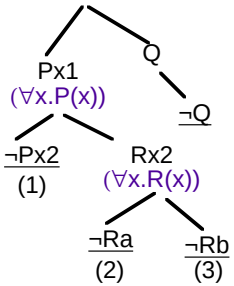
- a Universal variable can be treated as if it were universally quantified - as many (different) copies as may be needed are available implicitly;
- as a result any bindings made to a universal variable can be **ignored**.

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

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

Example Revisited

11aiv



Rx ∨¬Px, Px ∨Q, ¬Ra ∨¬Rb, ¬Q

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

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

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

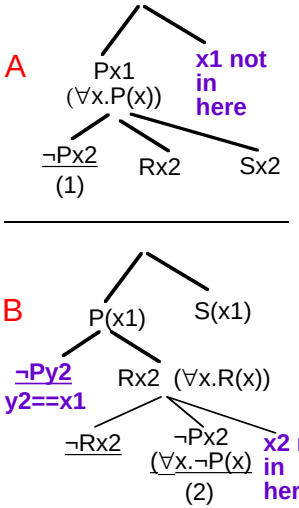
At closure (3) use instance R(b) of ∀x.R(x).

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.

What if clause Rx ∨¬Px happened to be Rx ∨¬Px ∨Sx?
Variable x in {Rx ∨¬Px ∨Sx is not universal, so it is important not to attempt to bind x1 to x2 at closure (1), as x1 would then lose its universal status. Instead, use a fresh instance of x1.

An Important Criterion

11av



In Example A , to close at (1), it is important to introduce instance P(x2) of the implicit ∀x.P(x).
If instead x2 and x1 were simply bound to each other, then x1 would be propagated to both Rx2 and Sx2 and x1 would lose its universal status.

In Example B, when closing at (2) it is important to introduce instance ¬Px1 of implicit ∀x.¬P(x). Otherwise x2 and x1 would be bound to each other and x2 would lose its universal status, which would affect the literal Rx2 as well.

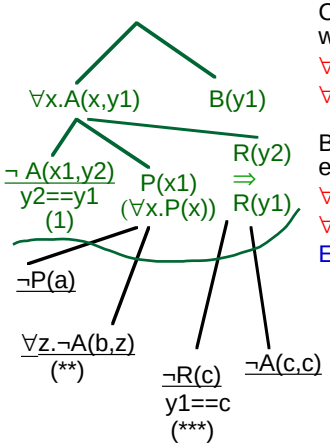
These observations lead to the following convention which will guarantee universal variables remain so.

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

Normal Form Representation of a Partial Tableau

11avi

Given (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)



Consider the tableau after closure (1) (above the wavy line). The open branches ≡
∀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)]

Before treating variables as universal, the expression would have been:
∀y1∀x1[(A(x1,y1) ∧ (P(x1) ∨ R(y1)) ∨ B(y1)] ≡
∀y [∀x[A(x,y) ∧ (P(x) ∨ R(y))]∨ B(y)] ≡ (*)
Effect is to maximally distribute ∀.

Variables x and z in clauses (3)/ (1) are universal. Notice x1 becomes universal and y1 is not universal. Can **``forget''** the bindings to universal variables (they are not shown). Thus instances A(x3,y1) and A(b,z3) are unified at (**) leaving y1 unbound. It is bound to c at (***). After the closures the remaining open branch contains B(c). Notice B(c) can be derived B(c) from (*), (1) and (2).

Generalised Closure Rule:

11avii

The slides 11ai - 11avi illustrate and explain an extension for ME-tableaux called the *generalised closure rule*, which uses 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 variables in L would each have the same free variables substituted 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 as P(v1,u1), P(v1,u2), giving the possibility of closure with ¬P(a,b)∨¬P(a,c), for example. If L is part of a clause L∨ Q(w), the duplication treats the clause as having the more general (nnf) form (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 also in B, then x1 is called a universal variable in T.

(Note: in the Chapter" notes, the generalised closure rule was also called generalised ancestor resolution.)

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

Example: Let x be a universal variable in a 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 B' and see it closes. That a variable becomes universal part way through a Model Elimination tableau derivation can be detected syntactically by noticing that the variable 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 11avi, when ¬A(x1,y2) closes with ∀x.A(x,y1), binding y2 to y1. All occurrences of y1 remain non-universal.

Exercise: Explain why, if the criterion on Slide 11av is adhered to, any variable declared universal will remain so during subsequent development of the tableau.

Soundness of the Generalised Closure Rule:

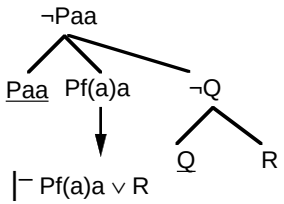
Justification that the generalised closure rule is sound can be made by appealing to the expression represented by a partial tableau, distributing quantifiers maximally across literals containing universal variables.

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. This was illustrated on slide 11avi. 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. There are several options for forming the binding in the closing unifier for a variable x: either x is universal and an instance of x becomes bound, or x is not-universal and is bound in the normal way. Thus e.g. when matching ∀x.A(x,y1) with ∀z.¬A(u1,z), the universal instance x2 of x is bound to u1, and the universal instance z2 of z is bound to y1. The open part of the extended tableau can be recomputed and universal quantifiers distributed to maximise occurrences of universal variables. By the earlier observation, newly designated universal variable occurrences will be in leaf literals only. (**Exercise**: Show the last statement is true.) Given the dnf representation of a partial tableau (call it dnf(T)), it is easy to show that if T' is derived from T then dnf(T)|=dnf(T').

(Soundness can also be shown by demonstrating how to construct an ordinary free variable tableau from one using generalised closure. The re-constructed tableau is not a ME tableau. The construction was informally illustrated on 11aii, but if the generalised closure rule is used in more than one branch, some care must be taken to ensure the construction is made correctly. See the problem sheet.)

A Generalisation of Prolog to arbitrary clauses

Given ¬Paa, ¬Pf(a)a ∨¬Paf(a), Pf(a)a ∨¬Q ∨Paa, Paf(a) ∨ Paa, Q ∨R, ¬R



If positive literals Pf(a)a and R are ignored then tree is complete

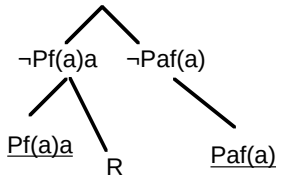
Can derive Pf(a)a ∨ R from above tableau: add ¬(Pf(a)a ∨ R) and derive a closed tableau.

In a similar way can derive Paf(a) by using Paf(a) ∨Paa instead of Pf(a)a ∨¬Q ∨Paa
(Called *Intermediate Lemmas*)

Intermediate Lemma Extension:
1. Select a positive literal from each non-Horn clause as the *conclusion literal* of that clause. Other positive literals are called *lemma literals*.
2. Proceed as in Prolog - begin from an *all-negative* clause, match with "conclusion" literals and ignore other positive literals that arise unless they match the immediate parent.

Step 1: choose (and underline) conclusion literals
Step 2: derive lemmas.

Ground clause example



Form lemma R.

Intermediate Lemma Extension (contd.):
3. Either - find a refutation - no lemma literals left as leaves;
Or - derive a lemma - only lemma literals left as leaves; form a lemma from the disjunction of all leaf literals.
4. Deal with the lemma as in 1, then can try an alternative path in 2, using new lemma.

Step 3: Find lemmas Pf(a)a ∨ R and Paf(a).

Step 4: Use these lemmas to form lemma R

Step 2/3: Find a refutation.

Can piece together the various tableaux used to obtain lemmas to form a closed tableau - it will not be a regular nor a ME tableau. In the example, in the final refutation, replace use of lemma R by tableau used to derive R, then replace use of other lemmas by tableaux used to derive them. (See 11bvii.)

Given: ¬G, G ∨¬Pxy ∨¬Pyx, Pf(u)u ∨Pua, Pvf(v) ∨Pva 11biii

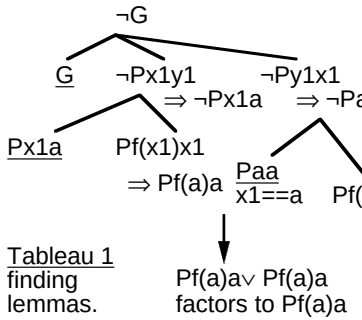
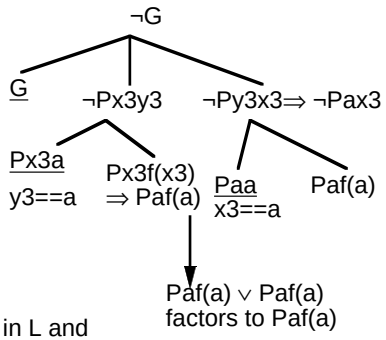


Tableau 1 finding lemmas.

Pf(a)a∨ Pf(a)a factors to Pf(a)a

General clause example



Paf(a) ∨ Paf(a) factors to Paf(a)

Tableau 2 finding lemmas.

If lemma is L∨M (open branches ending in L and M) then can close tableau by placing ¬(L∨M) ≡ ¬L∧¬M as an initial sentence in the tableau. So L∨M really is a lemma. e.g. Pf(a)a is implied by tableau 1 and Paf(a) is implied by tableau 2.

Given: ¬G, G ∨¬Pxy ∨¬Pyx, Pf(u)u ∨Pua, Pvf(v) ∨Pva, 11biv
Lemmas: Pf(a)a, Paf(a) (found as on 11biii)

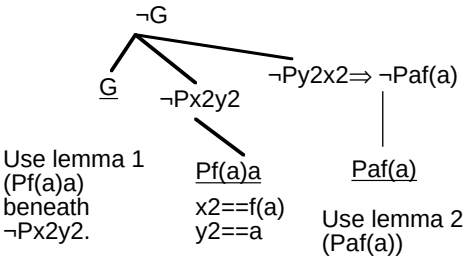


Tableau 3 using lemmas

The method seems to be fairly efficient:

In effect, many sub-tableau of small depth are joined together to form a large tableau. (When a lemma is used the tableau which derived it could be used instead.)

Because the individual search spaces are small the total search is reduced.

In case of a more general lemma, eg open branches ending in P(x) and Q(y,x), the lemma is ∀xy [P(x) ∨ Q(y,x)], since adding ¬ ∀xy [P(x) ∨ Q(y, x)] to the initial set of sentences, which Skolemises to the two facts ¬P(a), ¬Q(b, a) for new a and b, will enable the tableau to close.

Intermediate Lemma Extension (1):

The Intermediate lemma extension described in slides 11b is a hyper-resolution (or Prolog) like extension for tableaux. (Don't confuse with other ME-extensions that lead to so-called *hyper-tableaux* - see TABLEAUX conferences.) In any clause with at least one positive literal, *exactly* one positive literal is marked as the *conclusion literal*. Other positive literals (if any) are called *lemma literals*. (If a clause has exactly one positive literal it is the conclusion.) Each clause with no positive literals is called a goal clause. For uniformity, a new literal "G" can be appended to each goal clause (then all given clauses will have at least one positive literal). The top clause is a new clause, ¬G. A ME-tableau is constructed from the top clause, in which the lemma literals are initially ignored – branches in which they occur as leaf literals are not pursued.

If a tableau closes and there are no ignored lemma literals then this indicates that a refutation has been found. Otherwise, if a tableau closes and there are ignored lemma literals, the lemma literals are put into a new clause (i.e. the new clause is the disjunction of the lemma leaf literals), which is added to the data and called an *intermediate lemma*. This clause will have only positive literals, and one is chosen as the conclusion literal. Only intermediate lemmas that are not subsumed need be added. Clauses that are subsumed by the new intermediate lemma can also be removed. In case an intermediate lemma is retained, a new attempt at a refutation from ¬G can be made using all given clauses and all non-subsumed intermediate lemmas.

Factoring can be incorporated in two different ways. Either: (i) clauses can be (safe-factored) before being accepted (either as a given clause or as an intermediate lemma), or (ii), a positive literal that would otherwise be ignored may be matched with its parent (if possible). I prefer the first method, since it allows for all safe-factors to be found regardless of which positive literal might be selected as a conclusion literal. The second method is more dependent on the selection of conclusion literals to detect safe-factors. However, it is simpler to implement.

11bv

Intermediate lemma Extension (2):

The process of forming lemmas and refutations needs to be controlled in some way, analogous to setting depth limits in the standard ME procedure. The simplest is the following: All possible ways of closing a tableau descended from ¬G are formed and all intermediate lemmas formed, both upto some initial fixed depth. Then the process is repeated, but making use of the new clauses as well. If no further tableaux can be formed at the given depth and no refutation has been found then the process is repeated but to a greater depth.

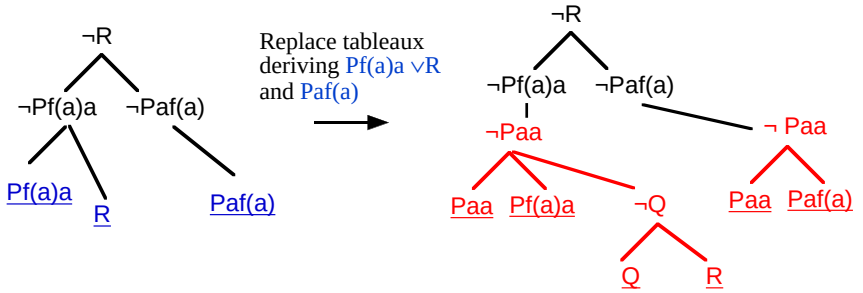
Unfortunately, the depth may need to be increased even though new intermediate lemmas can be found at the current depth. e.g. initial clauses include ¬Q, P(a), Q∨¬P(x)∨P(f(x)). Together, if Q is the conclusion literal in the third clause, these allow for the lemmas P(f(a)), P(f(f(a)), etc. to be formed, all at depth 2, even though none of these lemmas may be the ones required for a refutation. To overcome this problem can make a lemma contribute more than 1 to the depth-count when it is used in a refutation. e.g. make a lemma count exactly 1+number of lemmas used in its derivation. This is consistent with giving an initial clause a count of 1, since it uses no lemmas in its derivation. Hence, at a given depth there is a maximum number of lemmas that can be used and a finite number of possible refutations that could be made.

Once a refutation has been found, a complete tableau can be constructed from the various tableaux used to form the lemmas. Beginning with the last used lemma, the use of each lemma is replaced by the tableau that derived it. All its leaves will match in the same way that the lemma did. This substitution of tableaux can be repeated until all lemmas have been replaced.

In the tableau on 11bi/ii, first the tableau for R is used beneath ¬R. This tableau uses the clauses Pf(a)a ∨R and Paf(a), which are also lemmas. They are replaced by the tableaux which derived them, the leaf literals that formed the lemmas now matching where those of Pf(a)a ∨R or Paf(a) did. See diagram on 11bvii.

11bvi

Intermediate Lemma Extension: Reconstructing a closed tableau from lemmas



Soundness and Completeness of the Intermediate lemma Extension:

Next we show that the method of the Intermediate Lemma Extension is both sound and complete (at the ground level). The ground level tableau can then be lifted to give completeness and soundness at the general level in a similar way to that used for free variable tableau on slides 9.

To show soundness, note that each generated lemma is associated with a sub-tableau. These various sub-tableaux can be used in place of the corresponding lemma, as described on 11bvi. Each closure that was possible using the conclusion literal of the lemma is still possible using the tableau. For the example on 11bi/ii the final closed tableau is shown above.

Completeness of the Intermediate Lemma Extension:

The proof is by induction on the number of lemma literals in the given clauses. Let S be a minimally unsatisfiable set of clauses with a total of k lemma literals. (i.e. if any clause is removed from S then S would become satisfiable.)

If k=0 then the clauses are Horn clauses and since S is unsatisfiable there will be at least one all-negative clause in the set-of-support. Then there is a standard ME tableau starting from this all-negative top clause that will close (by completeness of ME). It is not hard to show the structure of the closed tableau is of the right form (and simulates a Prolog derivation from S).

If k>0, suppose as induction hypothesis (IH) that, for 0≤m<k lemma literals, there is always a set of sub-tableaux formed using the method, which can be pasted together to give a closed and soundly formed tableau. Let C be a clause with a lemma literal L and let C'=C-{L}. Form S'=S-{C}+{C'} and S"=S-{C}+{L}. Each may be made minimally unsatisfiable such that, for the case of S', C' is needed to show unsatisfiability, and in the case of S" {L} is needed. (Show this by using the assumption that S is minimally unsatisfiable). Since in both S' and S" the number of lemma literals <k, by IH there is a well-formed tableau from an all-negative clause for S' and for S".

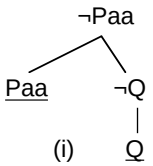
Now take the well-formed tableau for S' and put back L into C', forming C again. The tableau for S' was closed but will now give rise to the lemma L: in the derivation of the tableau for S', use of C (where before C' was used) will include using L, multiple occurrences all being factored to give the lemma L. In the well-formed tableau for S", this tableau deriving lemma L can now be used wherever originally the clause L was used.

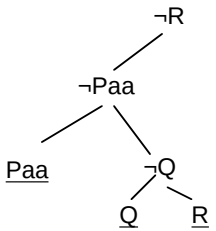
For the example on 11bi/ii the choices made for L are R from Q ∨R, Pf(a)a from Paa ∨Pf(a)a ∨¬Q and Paf(a) from Paa ∨ Paf(a); lemma atoms are non-conclusion atoms. There are 3. (Conclusion atoms are underlined.)

Assume C1 is Paa ∨ Paf(a), giving C1'= Paa. S1'={Q ∨R, ¬R, Paa ∨Pf(a)a ∨¬Q, Paa, ¬Paa, ¬Pf(a)a ∨¬Paf(a)}, which reduces to minimally unsatisfiable {Paa, ¬Paa}, and S1"={Q ∨R, ¬R, Paa ∨Pf(a)a ∨¬Q, Paf(a), ¬Paa, ¬Pf(a)a ∨¬Paf(a)}.

In S1" choose as C2 the clause Paa ∨Pf(a)a ∨¬Q, and use the IH to form tableaux from S2'={Q ∨R, ¬R, Paa ∨¬Q, Paf(a), ¬Paa, ¬Pf(a)a ∨¬Paf(a)} and S2"={Q ∨R, ¬R, Pf(a)a, Paf(a), ¬Paa, ¬Pf(a)a ∨¬Paf(a)}.

For S2' choose C3=Q ∨R and S3'={Q, ¬R,Paa ∨¬Q, Paf(a), ¬Paa, ¬Pf(a)a ∨¬Paf(a)} and S3"={R, ¬R}. The tableaux for S3", S2", S1' are fairly obvious. Tableau (i) below is for S3'. After re-inserting L3=R into S3' can use it in closure for S3", as shown in (ii). Two more constructions are needed to complete the full tableau according to the proof. These are left as an exercise for you.





(ii)

KE-TABLEAUX (D'Agostino, Mondadori, Pitt)

- Generalises Davis Putnam to first order sentences
- Only one splitting rule - called PB (Principle of Bivalence)

Non-splitting rules:

$$\frac{A \to B \quad \neg B}{\neg A} \qquad \frac{A \to B \quad A}{B} \qquad \frac{A \vee B \quad \neg A}{B} \qquad \frac{\neg(A \wedge B) \quad A}{\neg B}$$

Other rules the same as before

PB rule (ground):

for any ground sentence A

- Although quite a lot of theory for KE-tableaux has been developed, there is rather less on theorem proving techniques for it.
- KE is rather like Davis-Putnam, but with a more general splitting rule.
- KE can be simulated by standard tableau if the PB rule is added to the standard ruleset. This can easily be shown to be sound by extending the SATISFY property.
- The re-use rule in ME tableau also allows simulation for clausal data.
- KE can simulate standard tableau (for Skolemised sentences, at least).

Example of KE

11cii

Given data:
1. $a \wedge w \rightarrow p$ 2. $i \vee a$ 3. $\neg w \rightarrow m$, 4. $\neg p$ 5. $e \rightarrow \neg i \wedge \neg m$ 6. e

$\neg p$
e
$\neg i \wedge \neg m$ (5, $\rightarrow$rule)
$\neg i$ ($\wedge$rule)
a (2, $\vee$rule)

	$\neg (a \wedge w)$	$a \wedge w$	(PB)
($\neg\wedge$rule)	$\neg w$	p	(1, $\rightarrow$rule)
($\wedge$rule)	$\neg m$	-------	
(3, $\rightarrow$rule)	m	[]	

	[]		

KE seems to be good for propositional tableaux.

The amount of branching is generally lower than for standard tableaux

The PB rule is often used to introduce the second premise for the non-splitting rules. e.g. see the use of PB on $a \wedge w$ above.

First Order KE rules

11ciii

PB rule (non-ground):

$\neg A[x]$ $A[x]$

for new free variables x in sentence A

- The $\exists$-rule and $\forall$-rule are the same as for ordinary free-variable tableau;
- One method to deal with quantifiers is to draw them into a prefix and use free variable tableaux rules. These are often combined with one of the two-premise rules. See example on next slide.
- Little investigation of techniques for automatic first order KE have been made to date, so far as I'm aware.
- Soundness and Completeness have been shown for the ground case. • The KE-approach has proved useful for modal logics as well.
- Clausal KE is very similar to Davis Putnam, effectively providing a first order version of it.

Given: (1) $\neg(div(g(x),x) \wedge less(1,g(x)) \wedge less(g(x),x)) \rightarrow pr(x)$
(2) $div(u,w) \wedge div(w,z) \rightarrow div(u,z)$ (3) $less(1,x) \wedge less(x,n) \rightarrow div(f(x),x) \wedge pr(f(x))$
(4) $\neg(pr(y) \wedge div(y,n))$ (5) div(x,x) (6) less(1,n)

(u,w,x,y,z are universally quantified)

div(x2,x2) ($\forall$) (5)
$\neg$ pr(n) ($\forall, \neg\wedge$) (4)

(PB)

$\neg(div(g(x1),x1) \wedge less(1,g(x1))$
$\wedge less(g(x1),x1))$
pr(x1) (1, $\forall\rightarrow$)
---
x1==n

$div(g(x1),x1) \wedge less(1,g(x1)) \wedge less(g(x1),x1)$
$\Rightarrow div(g(n),n) \wedge less(1,g(n)) \wedge less(g(n),n)$
div(g(n),n), less(1,g(n)) $\wedge$ less(g(n),n) ($\wedge$)
div(f(g(n)), g(n)) $\wedge$ pr(f(g(n))) (3, $\forall\rightarrow$)
div(f(g(n)), g(n)), pr(f(g(n))) ($\wedge$)

(PB)

div(u1,w1)$\wedge$div(w1,z1)
div(u1,z1) (2, $\forall\rightarrow$)
$\neg$ pr(u1) (4, $\forall\neg\wedge$) z1==n
---
u1==f(g(n))

$\neg(div(u1,w1) \wedge div(w1,z1)) \Rightarrow$
$\neg(div(f(g(n)),w1) \wedge div(w1,n))$
$\neg div(g(n), n)$ ($\neg\wedge$) w1==g(n)

First Order KE example

11civ

KE Tableau Method:

11cv

The KE tableau method is more recent than other techniques, having been investigated in the last 15 years or so. (See "The Taming of the Cut", D'Agostino and Mondadori, and also Endriss: A Time Efficient KE Based Theorem prover.) In the first order case there has been very little work on practical theorem provers, so the example here is illustrative of what could be done. The KE rules can be viewed either as generalisations of the Davis Putnam steps or as variations of ordinary tableau rules, trying to retain much of the ME method, but for arbitrary sentences. There is just one splitting rule, but it is not restricted to atoms. The non-splitting rules are similar to the DP steps which prune atoms, and for clauses they are exactly the same. KE is more efficient than standard tableau (in the worst case), unless re-use (see Slide 10cii) is included in tableaux development, in which case (for clausal form) KE-tableaux can be simulated by ordinary ME-tableau +Re-use. Alternatively, one can soundly (and redundantly) add the splitting rule to the ordinary tableau method. See Problem sheet.

In the example on Slide 11civ it is appropriate to draw the universal quantifiers into a prefix, but more generally, this may not be so. Similar benefits to those provided by universal variables might be obtained if universal quantifiers are distributed as much as possible, but this remains to be investigated. The first step (in the example on 11civ) uses the ($\neg\wedge$) rule, together with the free variable $\forall$ elimination rule to derive $\neg$pr(n) from (4) and div(x2,x2). The next step anticipates the use of the($\rightarrow$) rule and sets this up by a PB application using $\neg(div(g(x1),x1)$ ...). In the second branch of the PB the($\rightarrow$) rule is used several times, in combination with free variables.

The KE method is quite human oriented in the ground case. But it is quite hard in the first order case because of the various ways that the ($\forall$) rule can be combined with other rules. It tends to give rise to less branching than ME tableau. There is another approach to adapting a linear style ME tableau to non-clausal sentences, which first compiles sentences into Q-clauses. This is described in the "Chapter" notes if you are interested.

Soundness and Completeness for Ground KE (Outline) 11cvi

The *soundness* of KE is simple to show; it is sufficient to show the property SATISFY for the rules (as for the standard tableau method) and including the PB rule. SATISFY is obviously true for an application of this rule (say for the sentence A), since the model of the branch before the rule must assign either T or F to A; if it assigns T then the branch below A will still be satisfiable and if it assigns F then the branch below ¬A will still be satisfiable.

It is also quite easy to show correctness (soundness and completeness) in a manner similar to that used for DP on Slides 1. This is not a coincidence, since KE is very similar to DP, especially if all sentences are clauses. (The extension mentioned in Slides 1 for non-clauses is very similar to KE.)

We define the α-rules to be those rules which are α-rules of ordinary tableau, and the β-rules to be the remaining non-splitting rules (e.g. A and ¬(A∧B) ==> ¬B). The minor sentence in a non-branching KE rule application of the β-kind is the smaller sentence (e.g. A in the above example rule). There are then basically 5 cases: a contradiction between a sentence and its negation, no sentences left to develop in a branch, an application of an α-rule, a sentence S used as the minor sentence in a β-rule application, and a PB application.

However, the proof similar to that used for DP requires the KE derivation to make all applications of a β-rule using the chosen minor sentence at once. Since this is not the normal way to make a KE derivation we'll give a different proof for completeness for ground KE. See 11cvii.

Completeness for Ground KE (Continued) 11cvii

Each sentence that occurs as a (sub)formula in the given data is a potential candidate for a β-rule application and is called a *given sub-formula*. (Note that atoms occurring in a given sentence are counted as given sub-formulas.)

An open branch of a KE tableau is called *fully developed* if every given sub-formula in the branch, or its negation, occurs at a node in the branch and no further rule applications are possible. Clearly, there is no need to use PB for any given sub-formula that already appears in a branch.

Let S be a given set of sentences and suppose a KE tableau is found with a fully developed open branch B. From this branch a model for S can be found in a similar way to that described for standard tableau and shown to satisfy the sentences in the branch. The proof for satisfiability is by contradiction from the asumption that some smallest sentence in B is false. For example, suppose such a smallest false sentence was of the form X∨Y. Then both X is false and Y is false. Since B is fully developed either X or ¬X is in B. If it is X then this contradicts the assumption that X∨Y is a smallest false sentence. If it is ¬X then the (∨) rule would have derived Y, again a contradiction. The full details are left as an exercise.

If S is an unsatisfiable set of sentences, then no fully developed open branch can exist and the KE tableau must close. For the first order case the method is similar.

A reasonable way to develop a KE tableau is thus to use the α-rules and β-rules as much as possible, only using PB to introduce the minor sentence for a β-rule application. The amount of splitting is then kept to a minimum.

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 logic

Disadvantages

- Let the original problem be Data $\vdash$ C, where C = P → Q. Clausal form of ¬(P → Q) is P and ¬Q. May want to work forwards from P and "backwards " from ¬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.
- In case interested 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.

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”?