AUTOMATED REASONING

SLIDES 2:

PROPOSITIONAL TECHNIQUES (2)

Implementing the DP procedure
Case Study 1: Model Generation Procedure
Case Study 2: OBDDs
(Ordered Binary Decision Diagrams)

Implementing DP efficiently (1)

The DP algorithm was invented in 1960. It is still widely used for proving unsatisfiability for propositional logic - eg in PVS, Prover9, modern satsolvers

What kind of optimisations are possible if the number of atoms and/or clauses is very large? At least will need to find an efficient way to:

- check if a clause is a new unit clause so can apply (Step 4) or (Step 5)
- check if a clause is deleted by subsumption or empty (can ignore or finished);
- eliminate complements of singletons and choose atom for split in (Step 7)

The system **Chaff (2001) Moskewicz et al** (and extensions) introduced the idea of *watchers*:

• keep a pair of indices associated with every clause that indicate 2 literals in the clause that have not yet been eliminated .

This allows to detect non-singletons, singleton (only 1 literal indicated), empty clauses (no literals indicated)

(They also indicated which was last literal decision affecting clause and whether clause is deleted or not, which helps with choice of selected literal and to restore states when back-tracking to alternative branches.)

Back-tracking is also improved, by remembering why decisions were forced and why branches were made False, and using the information so gained (see 2aiii).

Example: Implementing DP efficiently (2)

a	¬b	e	g	¬h

Suppose e = False and h =True set already
Let a and g be watched

1. If g = True then clause can be ignored
2. If g = False then must select new watched - must select ¬b
3. If b = False then clause can be ignored
4. If b = True then ¬b is set also
5. After 2. Suppose next a = False;
   then only ¬b is watched and also forces b = False

In general:
If a literal in clause becomes True, clause can be ignored;
If a literal in clause becomes False (and status becomes set) then
  if literal is not watched, do nothing
  if literal is watched then choose new watched
    if only one literal left to be watched it is forced to be true

Implementing DP efficiently (3)

(Non-chronlogical back-tracking) (see ppt)

2aiii

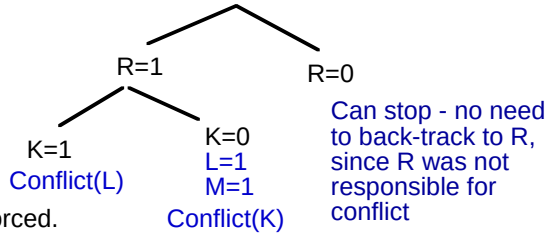
EG: LK ¬L¬K ¬LM ¬MK MR ¬RLM ¬ML¬R ¬MR ¬KL

Idea is to keep track of forced decisions:

Choose R=1

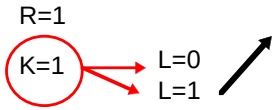
Choose K=1

Undo K=1
Now K=0

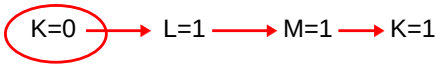


Can stop - no need to back-track to R, since R was not responsible for conflict

When K=1, both L and ¬L are forced.
Add *conflict clause* ¬K



Choose K=0; L, then M, then K are forced. Conflict(K)



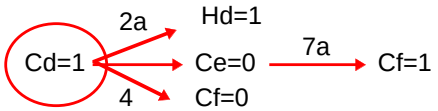
The learned conflict clauses are ¬K and K.
More generally, conflict clauses can be used elsewhere in the search space.

Example: Three girls revisited (see ppt)

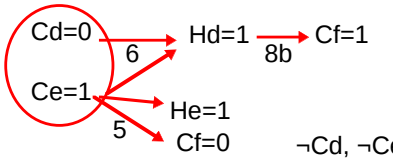
2aiv

• This time check for a model - i.e. don't add ¬Cf

(1) subsumed
(2a) ¬C(d) ∨ H(d)
(2b) ¬C(e) ∨ H(e)
(2c) ¬C(f) ∨ H(f)
(3) ¬ C(d) ∨ ¬C(e)
(4) ¬ C(d) ∨ ¬C(f)
(5) ¬ C(e) ∨ ¬C(f)
(6) C(d) ∨ H(d) ∨ ¬C(e)
(7a) C(e) ∨ C(f)
(7b) subsumed
(7c) subsumed
(8a) tautology
(8b) C(f) ∨ ¬H(d) ∨ C(d)
(8c) C(f) ∨ ¬H(d) ∨ ¬C(e)



Conflict clause ¬Cd



Conflict clause Cd ∨ ¬Ce
==> ¬Ce (using ¬Cd)
==> Cf (by 7a)
==> Hf (by 2c)

¬Cd, ¬Ce, Cf, Hf is a model as all clauses are satisfied!

Case Study 1: Model Generation Method

2bi

The Model Generation (MG) method has not been as widely investigated as DP **BUT** it is a very easy method for humans to apply.

What does it do?
Similar to DP, MG returns True if given clauses are **satisfiable**, else False, and constructs a tree where each branch is a partial model of the initial clauses

Branches of MG trees contain only atoms (+ve literals): the atoms in a branch give a partial model. Branches that cannot be extended to a full model are called *closed*.

Basic Idea

Consider a branch with atoms A and B and clauses ¬A ∨ C, ¬A ∨ ¬D, ¬A ∨ ¬B

We could extend the branch with atom C, since any model that makes A true will be forced to make C true because of ¬A∨C

We do not extend the branch with ¬D, since ¬D is a negative literal

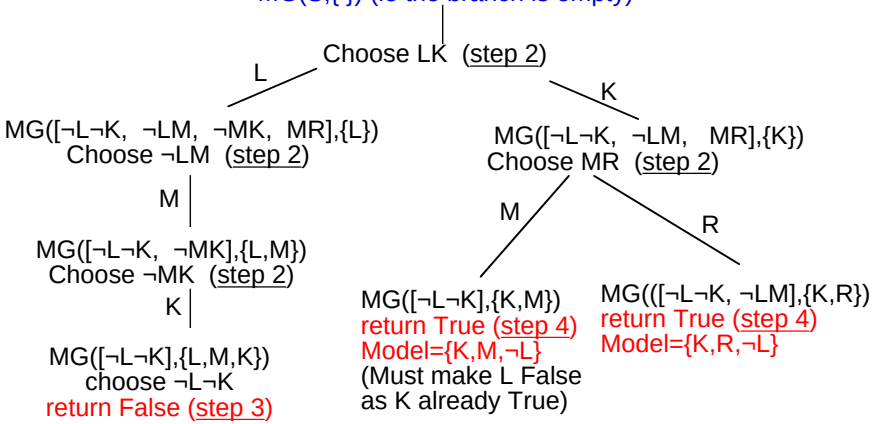
However, we can **close** the branch since no model that makes A and B true can make ¬A∨¬B true.

Branches that are not yet closed are called *open*.
A branch is *completed* if all initial clauses have been considered by the branch.

MG is related to the tableau method (coming later). Proof of correctness left until then

Model Generation Example (tree form) (see ppt) 2bii

Given clauses S: LK, ¬L¬K, ¬LM, ¬MK, MR (ie L∨K etc.)
Checks: no tautologies and no subsumed clauses, (initial checks)
at least one negative literal (step 1)
MG(S,{ }) (ie the branch is empty)



Return (False or True or True) =True

Model Generation Procedure 2biii

procedure MG(S,B): boolean

%S are clauses still to make true and B is the branch (model) so far

1. If no clause in S contains a negative literal then return true.
(B is a partial model of the initial clauses)

2. If S contains a clause C with ≥1 positive literals such that for all negative literals ¬L in C (if any), L occurs in B,
then return disjunction of MG(S(Ai),B+Ai) , for each positive literal Ai in C,
where S(Ai) = S - {X|X in S and X made true by assignments in B+Ai}

3. If S contains a clause C with only negative literals such that for all literals ¬L in C, L occurs in B, then return false.
(B makes C false)

4. If none of 1, 2 or 3 occurs then return true.
(Every remaining clause has ≥1 literal ¬L, where L is not in B)

MG is initially called with S=given clauses and B= [ ].

MG(S,B) halts with *true* if S+B has a model and returns a partial model that includes B (ie makes atoms in B true) and which can be extended to a complete model for S+B (see Notes on the procedure).

MG(S,B) halts with *false* if S +B has no models.

Notes about MG 2biv

MG will be covered again when we look at tableau methods. These notes are to give you some intuition as to why it works. You can assume there are no tautologies or subsumed clauses at the start and that all identical literals have been merged.

procedure MG(S,B): boolean

1. If no clause in S contains a negative literal return true.

Note1: B satisfies all clauses so far removed from the original argument S. B can be made into a model of the remaining clauses by assigning true to any atoms not yet assigned, since they either occur positively in S, or do not occur at all. (Covers the case when S is empty.)

2. If C = E ∨ D (where all literals Ai in E are positive and ¬Lj in D are negative) is in S and for all ¬Lj in D, Lj is in B, then return disjunction of MG(S(Ai),B+Ai) , for each Ai in E

Note2: The branches of the form B+Ai extending B are the only ways to make C true. Can remove any clauses in S also made true by B+Ai (for the Ai branch)

3. If C in S has only negative literals and for all literals ¬Li in C, Li is in B, then return false.

Note3: B cannot be extended to make C true as it already makes C false.

4. If none of 1, 2 or 3 occurs then return true.

Note4: Every remaining clause in S has ≥1 unassigned negative literal ¬L. Assign false to all such L to make the remaining clauses true. Can assign either true or false to any remaining atoms in the language not yet assigned.

Case Study 1: Model Generation Procedure (MG): 2bv

Another procedure appropriate for propositional clauses is *model generation* (MG), which is a special case of the tableau method (to be covered in detail later in the course). The MG method can be generalised to arbitrary propositional sentences, although there are better methods in that case. MG is easy for humans to use as it is more restricted than DP (hence its inclusion here).

The MG procedure attempts to build partial models. It constructs a tree in which each branch tries to maintain a partial model of the initial sentences (but in a different way to DP).

The tree built by the MG procedure for an initial set of clauses S consists of nodes each labelled by an atom. There are 2 kinds of branches, called *open* and *closed*. The atoms in each open branch B give a model of the set of clauses processed in B; when all of S have been processed in an open branch B then the branch is called *completed* and the nodes in B give rise to a (partial) model of S. A closed branch B is also called completed and indicates that the atoms in B cannot be extended to be a model of S (i.e. they make some clause in S false).

Let closed branches be labelled by false and completed open branches be labelled by true. Then the given clauses S have no model if the disjunction of all labels in a tree in which all branches are completed is false. There is a model if the disjunction is true, as at least one branch must then have been completed and open.

Proof of the correctness of the MG procedure is left until we look at tableau methods.

Example:
{LK, ¬L¬K, ¬LM, ¬MK, MR} (a slightly different tree than that shown on 2bii)

(Step 2) Choose MR (could also have chosen LK instead as on Slide 2bii).
Return MG([LK,¬L¬K, ¬MK], {M}) **or** MG([LK,¬L¬K, ¬LM, ¬MK], {R});
i.e. start a tree with two branches.
Consider the first disjunct and (Step 2), clause ¬MK.
Return MG([¬L¬K], {M,K}). ie still to make ¬L⁄¬K true

(Step 4) return true - assign false to L.
Check that {M,K,¬L} is a (partial) model for the initial clauses.

If the second disjunct had been chosen, then apply (Step 2) and clause L⁄K and return MG([¬L¬K, ¬LM, ¬MK], {R,L}) or MG([¬L¬K, ¬LM], {R,K});
i.e. extend the second branch of the tree by two branches.

The first disjunct will eventually return false (show this yourself)
and the second will return the model {R,K,¬L}.
Check that {R,K,¬L} is a (partial) model for the initial clauses.

Exercises:
1)Use MG on the "three little girls" problem.
2) Consider how a purity rule might be included into MG
3) Can you suggest any efficiency short cuts for MG? 2bvi

"The three little girls" problem again! 2ci

The data

(1) C(d) ∨ C(e) ∨ C(f) One of the threee girls was the culprit
(2) C(x) → H(x) { C(d) → H(d), C(e) → H(e), C(f) → H(f) }
To convert into propositional form
(3) ¬(C(d) ^ C(e))
(4) ¬(C(d) ^ C(f))
(5) ¬(C(f) ^ C(e))

Only one of the three girls was the culprit
(6) C(d) ∨ H(d) ∨ ¬C(e) (Dolly's statement negated)
(7) C(e) ∨ C(f) ∨ ¬(C(e) → (C(d) ∨ H(d))) (Ellen's negated)
(8) C(f) ∨ ¬H(d) ∨ ¬((H(d) ^ C(d)) →C(e)) (Frances's negated)

This time we'll try to find a model and return True (and we hope the model will make C(f) true).

We convert to clauses and can remove any tautologies or subsumed clauses at the start. Also merge identical literals

Solution to "The three little girls" by MG (see ppt) 2cii

(1) C(d) ∨ C(e) ∨ C(f) (2a) ¬C(d) ∨ H(d) (2b) ¬C(e) ∨ H(e)
(2c) ¬C(f) ∨ H(f) (3) ¬ C(d) ∨ ¬C(e) (4) ¬ C(d) ∨ ¬C(f)
(5) ¬ C(e) ∨ ¬C(f) (6) C(d) ∨ H(d) ∨ ¬C(e) (7a) C(e) ∨ C(f) ∨ C(e)
(7b) C(e) ∨ C(f) ∨ ¬C(d) (7c) C(e) ∨ C(f) ∨ ¬ H(d) (8a) C(f) ∨ ¬H(d) ∨ H(d)
(8b) C(f) ∨ ¬H(d) ∨ C(d) (8c) C(f) ∨ ¬H(d) ∨ ¬C(e)

Merge literals in (7a) to obtain C(e) ∨ C(f) and remove (8a) (tautology);
(7a) subsumes (7b), (7c), (1);

call MG([2a, 2b, 2c, 3, 4, 5, 6, 7a, 8b, 8c], []);
Apply (Step 2) on 7a (C(e) ∨ C(f)); evaluate
MG([2a, 2b, 2c, 3, 4, 5, 6, 8b,8c], [C(e)]) (i) **or** MG([2a,2b,2c,3,4,5,6], [C(f)]) (ii);

In (i): apply (Step 2) to 2b and call MG([2a,2c,3,4, 5, 6,8b, 8c], [H(e), C(e)]);
Apply (Step 2) to 6 and evaluate
MG([2a,2c,3,4, 5,8c], [C(d), H(e), C(e)]) **or** MG([2c,3,4,5,8b, 8c], [H(d),H(e), C(e)]);
Both eventually return false.

In (ii): apply (Step 2) to (2c) and call MG([2a, 2b, 3, 4, 5, 6], [H(f), C(f)]);
Apply (Step 4) to return true; get the partial model [H(f), C(f), ¬C(d), ¬C(e)].
H(e) and H(d) can be either true or false.

Case Study 2:Ordered Binary Decision Diagrams (OBDDs)

- Both the DP and MG methods process clauses
- Simplifying steps (e.g. subsumption) are performed on the whole set
- The method of OBDDs can process **any** set of propositional sentences

• OBDDs convert a formula into a binary decision diagram, which is equivalent to a large *If-Then-Else* formula. Either the formula is shown to be equivalent to False, or a model can be read from the diagram

OBDDs are usually computed "Bottom-up":

Sub-formulas of the given formula are converted to *If-Then-Else* triples and triples are then combined until the whole formula is an *If-Then-Else* formula

OBDDs are based on the If –Then – Else connective:
(eg $A \wedge B$ ≡ "If A then B else false" and denoted if(A, B, false)

- A ≡ If A Then true Else false
- ¬ A ≡ If A Then false Else true
- $A \vee B$ ≡ If A Then true Else B
- $A \leftrightarrow B$ ≡ If A Then B Else (¬B)
- $A \rightarrow B \equiv \neg A \vee B \equiv$ if(A, B, true) (transformed because → is not symmetric)

OBDDs may also be computed "Top-down" (see problem sheet)
See Moore, JAR, 1994 for this logic-based approach.

2di

Basis of Bottom-up conversion

2dii

• In an OBDD the order of atoms is uniform throughout (hence they are ordered)

• Two OBDDs can be combined:

if(A, X, Y) $\wedge/\vee/\leftrightarrow$ if(A, W, Z) ≡ if(A, (X $\wedge/\vee/\leftrightarrow$ W), (Y $\wedge/\vee/\leftrightarrow$ Z))
if(A, X, Y) $\wedge$ B (A not tested in B) ≡ if(A, (X $\wedge$ B), (Y $\wedge$ B)), and for $\vee/\leftrightarrow$
if(A, X, X) ≡ X (A simplification step)

• The properties follow from the *Shannon Equivalence*:

W ≡ (X $\wedge$ W[X==True]) $\vee$ (¬X $\wedge$ W[X==False]) for any atom X
where W[X==True] results from replacing X by True in W and simplifying, and similarly for W[X==False]

Equivalently
W ≡ if (X, W[X==True], W[X==False])

eg1: $A \leftrightarrow B \equiv$ if (A, (A ↔ B)[A==True], (A ↔ B)[A==False]) ≡ if(A, B, ¬B)

eg2: if(A,X,Y) ↔ if(A,U,V)
≡ if(A, (If(T,X,Y) ↔ if(T,U,V)), (if(F,X,Y) ↔ if(F,U,V)))
≡ if(A, (X ↔ U), (Y ↔ V))

Example of Bottom-up conversion (see ppt)

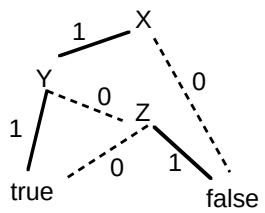
2diii

if(A, X, Y) $\wedge/\vee/\leftrightarrow$ if(A, W, Z) ≡ if(A, (X $\wedge/\vee/\leftrightarrow$ W), (Y $\wedge/\vee/\leftrightarrow$ Z))
if(A, X, Y) $\wedge$ B (A not tested in B) ≡ if(A, (X $\wedge$ B), (Y $\wedge$ B)), and for $\vee/\leftrightarrow$
if(A, X, X) ≡ X (A simplification step)

e.g. OBDD for X $\wedge$ (Y $\vee$ ¬ Z):
(put T and F for true and False and choose atoms in the order X, Y, Z)

Y $\vee$ ¬Z
= if(Y, T, F) $\vee$ if(Z, F, T)
= if(Y, (T $\vee$ if(Z, F, T)), (F $\vee$ if (Z, F,T)))
= if (Y, T, if(Z, F, T))

X $\wedge$ (Y $\vee$¬ Z)
= if (X, T, F) $\wedge$ if (Y, T, if(Z, F, T))
= if (X, (T $\wedge$ if (Y …)), (F $\wedge$ if (Y …)))
= if (X, if (Y, T, if (Z, F, T)), F)



What are the models?

An Algorithm for generating an OBDD

2div

Let W be a formula and Z the corresponding OBDD given some atom order "<"

OBDD(W) = Z
W = T or W = F | Z = W
W = atom X | Z = if(X,T,F)
W = ¬ X and X is an atom | Z = if(X,F,T)
W = ¬X and X is not an atom | Z = OBDD(X) with terminal T/F reversed
W = X op Y | Z = APPLY (op, X1, Y1),
where OBDD(X) = X1 and OBDD(Y) = Y1

APPLY(op, X, Y) = Z
X = T or X = F or Y= T or Y = F | Z = X op Y
root(X) <root(Y) and X = if(root(X), X1, X2) |
Z = REDUCE(if(root(X), APPLY(op, X1,Y), APPLY(op, X2, Y)))
root(Y)<root(X) and Y = if(root(Y), Y1, Y2) |
Z = REDUCE(if(root(Y), APPLY(op, X, Y1), APPLY(op, X, Y2)))
X = if(root(X), X1, X2), Y=if(root(X), Y1, Y2) |
Z = REDUCE(if(root(X), APPLY(op, X1, Y1), APPLY(op, X2, Y2)))

REDUCE(if(X,Y,Y)) = Y

OBDD Example : [LK, ¬L¬K, ¬LM, ¬MK, MR] 2dv

(All clauses anded together; t is true, f is False;
atoms tested alphabetically; If X Then Y Else Z represented as if(X,Y,Z))

(1) LK = if(K, t, if(L , t , f))
(2) ¬L¬K = if(K, if(L , f , t) , t)
(3) ¬LM = if(L, if(M ,t, f) , t)
(4) ¬MK = if(K , t , if(M ,f ,t))
(5) MR = if(M , t , if(R, t , f))

(1) ∧ (2): if(K, if(L, f, t), if(L, t, f))

∧ (4): if(K, if(L, f, t), if(L, t, f) ∧ if(M, f, t)) = if(K, if(L, f, t), if(L, if(M, f, t), f))

∧ (3): if(K, if(L, f,t) ∧ (3), if(L, if(M, f, t) ∧ if(M, t, f), f))
= if(K, if(L, f, t) , if(L, if(M, f, f), f))
= if(K, if(L, f, t) , if(L, f, f)) = if(K, if(L, f, t) , f)

∧ (5): if(K, if(L, f, t) ∧ if(M, t, if(R, t, f)) , f)
= if(K, if(L, f, if(M, t, if(R, t, f))), f)

Gives models [K, ¬ L, M] or [K, ¬L, ¬M, R]
Can you see how to read off the model from the final expression?

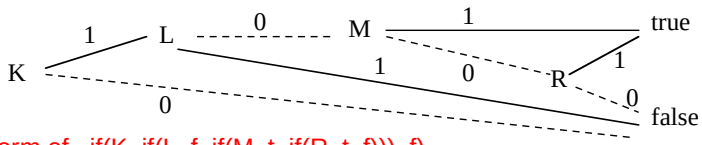
Exercises:
1. Draw the graphical form of the final result (Answer on next slide).
2. In a clausal conversion, where is the best place in the literal order for pure literals?

Some Properties of OBDDs

2dvi

They preserve equivalence: OBDD(S) ≡ S
If OBDD(S) = False then S has no models
if OBDD(S) ≠ False then S has models that can be read from the tree: follow the decisions on any path from the root to the True node
if OBDD(S) = True then S is a tautology.

Notes:
OBDDs are one of several BDD normalisation procedures for ground clauses
See Moore, JAR, 1994 for logic-based approach. Their improved algorithm stored OBDDs in a table and used look-up to avoid duplications.
Another important normal-form uses Decomposable NNFs (Darwiche)



Graphical form of if(K, if(L, f, if(M, t, if(R, t, f))), f)

Summary of Slides 2

2ei

1. Some ideas for Implementation of DP were discussed.

2. The Model Generation (MG) method for testing satisfiability of propositional clauses was described; it is good for humans, but less well investigated in optimised implementations.

3. MG returns True (and a partial model) for given clauses S if there is one. It returns False if there are no models of S. The returned model (if any) can be extended to all atoms by assigning T or F to unasigned atoms.

4. The state of MG can be represented as a tree, in which each node N is labelled by the current set of clauses S (those still to be made True in the branch B ending at N). The arcs of the tree are labelled by atoms and the atoms labelling arcs in B are a partial assignment that satisfy those clauses in the initial set which are not still in S (for branch B). This property is maintained as an invariant.

5. MG was used to solve the “Three Little Girls” problem.

2eii

6. The OBDD (Ordered Binary Decision Tree) was introduced.

7. An algorithm for generating the OBDD form of a proposition was given.

8. An OBDD can be represented as a graph, in which each node is labelled by an atom, the one “decided” at that node. A given order of atoms is used in any particular problem. The arcs in a branch are labelled by 1 or 0. Two special nodes are True and False.

9. For a given proposition S:
- if no arcs in OBDD(S) enter False then S ≡ true (i.e. is a tautology);
- if at least one arc in OBDD(S) enters True then S has a model that can be read from the graph;
- if no arcs in OBDD(S) enter True then S ≡ false and has no models.

10. The correctness property of OBDD relies on the preservation of equivalence by the simplication rules. Various optimisations are possible.

A Solution to Aunt Agatha's Burglary

2fi

Note: (x and y are implicitly universally quantified)

(1) s(m,a) (someone stole from Agatha)
(2) x=a ∨ x=b ∨ x=j (The only people are Agatha, James and the butler)
(3) s(x,y) → (d(x,y) ^ ¬ r(x,y))
(thieves dislike, and are not richer than, their victims)
(4) d(a,x) → ¬d(j,x) (James dislikes no-one whom Agatha dislikes)
(5) ¬ x= b → d(a,x) (Agatha dislikes all but the butler)
(6) ¬r(x,a) → d(b,x) (butler dislikes anyone not richer than Agatha)
(7) d(a,x) → d(b,x) (and also anyone Agatha dislikes)
(8) ¬(∀z d(x,z)) (No-one dislikes everyone)
(9) ¬ a=b
(10) Conclusion: s(a,a) (Agatha burgled herself)

First simplify (8) to ∃z ¬d(x,z) and then to ¬d(x, f(x)).
(For each x, f(x) is a person x doesn't dislike)

Put x as m in (2); m=b and m=j will lead to contradictions forcing m=a.

2fii

m=j: s(j,a) (by equality substitution in (1)) and hence d(j,a) and ¬r(j,a) from (3);
From (5) and (9) d(a,a) hence ¬d(j,a) by (4);
Contradiction so m≠j.

m=b: ¬d(b,f(b)) (*) (put x as b in (8));
hence ¬d(a,f(b)) by (7) and ¬ ¬ f(b)=b by (5);
Therefore f(b) = b and ¬d(b,b) from (*);
s(b,a) (by equality substitution in (1)) and hence d(b,a) and ¬r(b,a) from (3);
Hence d(b,b) from (6);
Contradiction so m≠b;

Therefore m=a;
s(a,a) (by equality substitution in (1)) which is the conclusion.

(See slides further in the course for other approaches.)

A (Natural) Solution to the "three little girls"

2fiii

(1) C(d) ∨ C(e) ∨ C(f)
(2) C(x) → H(x) (x is implicitly universally quantified)
(3) ¬(C(d) ^ C(e))
(4) ¬(C(d) ^ C(f))
(5) ¬(C(f) ^ C(e))
(6) C(d) ∨ H(d) ∨ ¬C(e) (Dolly's statement negated)
(7) C(e) ∨ C(f) ∨ ¬(C(e) → (C(d) ∨ H(d))) (Ellen's negated)
(8) C(f) ∨ ¬H(d) or ¬((H(d) ^ C(d)) → C(e)) (Frances's negated)

By (1) it must have been C(d), C(e) or C(f).

Case 1: C(d). (Suppose Dolly did it.)
(7) cannot then be true as all 3 parts lead to a contradiction.
So Dolly is not the culprit.

(¬(X → Y) true means X true and Y false.)

Case 2: C(e) Suppose Ellen did it. Then H(d) follows from(6)
(remember ¬C(d) from Case 1);
then (8) leads to contradiction.
So Ellen is not the culprit.

Hence Frances was the culprit. (See slides for other approaches.)

A Solution to the Mathematical problem

2fiv

(1) a o b = c
(2) o is an associative operator:x o (y o z) = (x o y) o z
(3) x o x = e
(4) x o e = e o x = x (e is the identity of o)

Show b o a = c

x o (a o b) = (x o a) o b	(put y as a and z as b in (2))
x o c = (x o a) o b	(use a o b = c)
c o c = (c o a) o b	(put x as c)
e = (c o a) o b	(use (3) c o c = e)
e o b = e o b	(property of =)
e o b = ((c o a) o b) o b	(use e = (c o a) o b)
b = ((c o a) o b) o b	(use (4) e o b = b)
b = (c o a) o (b o b)	(use (2))
b = (c o a) o e	(use (3) b o b = e)
b = c o a	(use (4) (c o a) o e = c o a)

(See slides near the end of the course for other approaches.)