# AUTOMATED REASONING

# SLIDES 1:

# PROPOSITIONAL TECHNIQUES

**Basic Concepts**
**Davis Putnam Procedure**
**Solving the “Three Little Girls” Problem**
**Heuristics for Implementing DP**

**KB - AR - 13**

## Techniques and Examples for Propositional Clauses

For propositional sentences there are some special techniques
Here we look at (the famous) **Davis Putnam procedure** using clausal form

**A few notations for ground clauses (no quantifiers):**
- the propositional *language* L names the atoms that may occur in a clause
- a *ground clause* is a disjunction of ground literals
- a *ground literal* is a ground atom or negated ground atom
- a *singleton* clause is called a *fact* or *unit* clause
- a clause C in which literals X and ¬ X occur is a *tautology* (always true)

**Examples**: Let the language be {A, B, C, D}
- A and ¬B are *literals*
- A∨¬B∨¬D is a *clause*. So is C.
- C is a fact, also called a *unit* clause.
- is the *empty* clause, (an empty disjunction) which is always False.
- A∨¬B∨¬A is a *tautology*, which is always true.

***Note:*** A∨B∨C may be written as, and identified with, {A, B, C} (or ABC);
if X and Y are sets of clauses their union may be written X + Y.

1ai

## Valuations for Propositional Clauses (1)

1aii

Let L be a propositional language
- a *valuation over L* assigns True(T) or False(F) to every atom in L
- a valuation over L assigns T/F to literal ¬X if it assigns F/T to atom X
- a valuation over L *satisfies* clause C if it assigns T to ≥1 literal X in C

Examples: Given the clauses A∨B, ¬A∨C∨¬D, ¬C
Which clauses are satisfied by
(1) the valuation A=B=C=D= False **?**
(2) the valuation A=B=True; C=D= False **?**

- a valuation over L is a *model* of a set of clauses S written in L
  **iff** it satisfies all clauses in S.
- S is *satisfiable* if it has a model

Valuation (2) above is a model of the given clauses
The given clauses are *satisfiable*

- S is *unsatisfiable* if it is not satisfiable; ie if it does not have a model

## Valuations for Propositional Clauses (2)

1aiii

Let L be a propositional language
- let S be a set of clauses and G be a clause in L.

Then S |= G iff for all valuations M over L,
if M is a model of S, then M is a model of G;

- S+{¬ G} is *unsatisfiable* iff S |= G
  (because every model of S satisfies G and falsifies ¬G;
  if G is a clause then ¬G is a set of facts)

Examples:
A∨B, ¬A, ¬B are *unsatisfiable*. **Why?**
A∨B, ¬A |= B since every model of A∨B and ¬A is a model of B
A∨B |= A∨B (¬(A∨B) is *equivalent* to (≡) ¬A and ¬B)

## Valuations for Propositional Clauses (3)

1aiv

• Let S1 and S2 be sets of clauses written in L;
S1 is equivalent to S2 (written S1 ≡ S2) iff
(M is a model of S1 iff M is a model of S2)

In other words, S1 and S2 have the same models

Example:
{¬(A∨B)} is equivalent to {¬A, ¬B} **Why?**

• clause X *subsumes* clause Y if X ⊆ Y. Note X $\models$ Y; **Why?**

(Remember clauses can be considered as a set of literals)

Example:
¬A∨¬D *subsumes* ¬A∨C∨¬D. Note ¬A∨¬D $\models$ ¬A∨C∨¬D; **Why?**

## Davis Putnam Method

1bi

The **Davis Putnam** decision procedure is used to decide whether ground clauses S are satisfiable or unsatisfiable.

It attempts to show that S has no models by reducing satisfiability of S to satisfiability of sets of simpler clauses derived from S

• DP is called with 2-arguments:
• Arg1 is a partial model of clauses processed so far, and
• Arg2 is the list of clauses still to process.
*Either* no model exists and false is returned,
*or* all clauses are processed and true is returned with a model in Arg1.

• Initial call is DP([],S).
Since no clauses processed so far, Arg1 is empty and Arg2 =given clauses.

• It is usual to first remove tautologies and *merge* identical literals in a clause, (e.g. A∨A∨B becomes A∨B), and then call DP([], S)

The algorithm is on 1biii and an example is on 1bii

(Also called the Davis Putnam Loveland Logemann (DPLL) method)

## DP Example written as a tree (see ppt slides)

1bii

No tautologies and no subsumed clauses
DP([], S) = DP([],[LK, ¬L¬K, ¬LM, ¬MK, MR])

R is ***pure*** so delete MR (step 4) and make R true:
DP([R], S') = DP([R],[LK, ¬L¬K, ¬LM, ¬MK ])

Choose L (step 7)

Make L true
DP([L,R],S') = DP([L,R],[¬K, M,¬MK])

Choose ¬K (step 6)
DP([¬K,L,R],[M, ¬M])

return False(step 2)

Make L false; DP([¬L,R], S'')=
DP([¬L,R], [K, ¬MK])

K subsumes ¬MK (step 3)
DP([¬L,R],[K])

K is pure (step 4)
DP( [K,¬L,R],[])

return True (step 1)

Finally return (False **or** True) = True (so data is satisfiable) and recover a **partial** model from the right branch = {K, ¬L, R}. Can assign either T or F to M

## The DP procedure

```
procedure DP(M, S) : boolean;
%M is a possible model so far and S are clauses still to process
1. If S is empty  record M and return true;                %M is  a (partial) model
2. If S contains clauses X and ¬X return false;              % S has no models
3. If C is  a subsumed clause in S return DP(M,S-C);
4. If P is literal in  C with no complement in S (called a pure literal)
   then  return DP([P|M], S'), where   S' = S -  { D | P in D};      %Make P true
5. If A is a fact in S return DP([A|M], S'),  where S' =  S processed as follows:
      remove clauses containing A and remove ¬A from rest
6. If ¬A is a fact in S return DP([¬A|M ], S"), where  S"= S processed as
      follows: remove clauses containing ¬A and remove A from rest.
7. Otherwise select an atom A in a non-unit clause in S and form
     S' and S" as in Steps 5 and 6;  return  DP([A|M ], S') ∨ DP( [¬A|M],  S")
```

## The DP procedure (Notes)

On slide (1biii) [P|M] is the Prolog list notation for a list with head P and tail M.

The partial model M from step 1 is formed by assigning T to atom A if A is in M, and assigning F to atom A if ¬A is in M.

Since M need not have assigned either T or F to some atoms in the language of the given clauses it is a partial model, and the assignment must be extended to make it a model.

The partial model can be extended to be a model by assigning either of T or F to any  still unassigned atom in the language.

As an example, carry out the procedure on the initial clause A ∨ B. You will get the partial model A assigned T, or the partial model B assigned T.

Often, a model is represented by the set of atoms assigned T by the valuation.

**Davis Putnam Procedure:**

When computed by hand the sets of clauses that are arguments to calls of DP can be maintained in a tree.  For the inital clauses S = {LK, ¬L¬K, ¬LM, ¬MK, MR} we might get the  tree shown on Slide 1bii.  (There are others, it depends on the choice of literals in steps 4 and 7.) The initial node contains the initial set S and an empty partial model.

R is pure, so remove MR (Step 4). The tree is extended by a node containing the set {LK, ¬L¬K, ¬LM, ¬MK} and R is added to the partial model. Next use (Step 7) and choose M (note: for illustration this is a different choice than shown on Slide 1bii, but the final decision about satisfiability will be the same); the tree is extended by 2 branches, one getting model {R,M} and reduced clauses {LK, ¬L¬K, K} and the other getting model {R,¬M} and {LK, ¬L¬K, ¬L}. From {LK, ¬L¬K, K} use (Step 5) for K and get a new node below it with model {R,M,K} and reduced clauses {¬L} and from {LK, ¬L¬K, ¬L} use (Step 3) to remove ¬L¬K, and then (Step 6)  for ¬L and get a new node beneath it with model {R,¬M,¬L} and reduced clauses {K}.  In case 1, ¬L is pure and in case 2, K is pure. Removing either leads to an empty set of clauses and the procedure returns true so the initial clauses are satisfiable.

The first branch returns the (partial) model {R,M,K,¬L} and the second branch returns the (partial) model {R, ¬M, ¬L, K}. You can check these both satisfy the initial clauses.

Note for example, that if we add L¬K to the initial set of clauses, then we would have got {¬L,L} and/or {K, ¬K} in the last nodes. Both  return false showing the set of clauses {LK, ¬L¬K, ¬LM, ¬MK, MR, L¬K} is unsatisfiable.

**Various properties can be proved for DP:**

As the Davis Putnam procedure progresses, the first argument M is maintained as a partial model of clauses already processed. If the procedure ends with an empty set of clauses then M will be a partial model of the initial set of clauses S. It may have to be extended to the whole signature, for instance if it doesn't include assignments for all atoms. The second argument is simplified at each step.

The following two invariant properties are proved on Slides 1ciii and 1civ.

(1) At each step any literal that appears in M will not occur positively, or negatively, in S. This is clearly true vacuously at the start.

(2a) At each single branching step
M∪S is satisfiable iff M' ∪S' is satisfiable,
where M' is the resulting value of Arg1.

(2b) At each double branching step
M∪S is satisfiable iff M' ∪S' is satisfiable or M" ∪S" is satisfiable,
where M' and M" are the resulting values of Arg1.

***Note:*** If it is required to know only whether a set of clauses S is satisfiable or not, there is no need for the argument that maintains the model and the invariant (1) can be dropped, and (2a) and (2b) simplified by dropping reference to M, M' and M".

## Why Does DP Work? (1)

1ci

We'll consider a simple version where we don't bother with finding a model.

The idea behind the procedure:
**each step** should maintain satisfiability/unsatisfiability of clauses S:

i.e. in the call DP(S) S is satisfiable
iff (iff is shorthand for if, and only if)
in the next call DP(S') S' is satisfiable

*This is equivalent to S is unsatisfiable iff S' is unsatisfiable*

$$\begin{tikzcd}
\text{S1 (un)satisfiable} \arrow[dr, leftrightarrow] & & \text{S3 (un)satisfiable} \arrow[dr, leftrightarrow] & & {} \\
& \text{S2 (un)satisfiable} \arrow[ur, leftrightarrow] & & \text{S4 (un)satisfiable} \arrow[ur, dashed, leftrightarrow] &
\end{tikzcd}$$

Until:
Sn is **clearly** unsatisfiable –
Sn includes A and ¬A
or:
Sn is **clearly** satisfiable –
Sn is empty

**EG**: Want to show S1 is (un)satisfiable.
After a sequence of steps may reach the simpler Sn.....
*Conclude* S1 is unsatisfiable if Sn is shown to be unsatisfiable, and
S1 is satisfiable if Sn is empty

## Why Does DP Work? (2) (see ppt)

1cii

**EG Step 4 – P is pure: delete clauses including P from S to give S'**

(1) S satisfiable ==> S' is satisfiable:
Let I be a model of S.
S' is smaller than S so I must be a model of S'.

(2) S' satisfiable ==> S is satisfiable:
Suppose I is a model of S'
In order to make S true I must also satisfy the deleted clauses like C = P∨D
Since P is pure in S, ¬P does not occur in any clause in S or S' - Why?
So I does not have to assign to P to satisfy S'
Hence we can arbitrarily assign true to P in I and make true all clauses like C

Hence the property holds for step 4.

For the branching step 7, the (simplified) invariant is a bit more complicated:

S is satisfiable iff S' is satisfiable or S" is satisfiable ≡
S is unsatisfiable iff S' is unsatisfiable and S" is unsatisfiable. (See 1ciii - 1cv)

## Proof of Correctness of DP:

1ciii

In the following proof, some details are left as exercises, labelled Exercise 1, Exercise 2, etc.

Exercise 1: Let S be a set of clauses. Show that
S is satisfiable iff S' is satisfiable is equivalent (≡ ) S is unsatisfiable iff S' is unsatisfiable.
Hint: S is unsatisfiable ≡ not (S is satisfiable)

First is shown that the simplified invariant property holds, and then is shown how this leads to the property that DP(S) returns False iff S is unsatisfiable.

The simplified invariant property states that the clauses S before the step are satisfiable *iff* the clauses after the step S' are satisfiable. For example, the case for (Step 4) is given on Slide 1cii.

Exercise 2: Show the invariant holds for (Step 3), (Step 5), (Step 6)

Exercise 3: Show that the invariant holds for (Step 7): the clauses S before the step are satisfiable *iff* at least one of the clauses after the step, S' or S", are satisfiable

Using the invariant property we now show by induction on the number of propositional symbols occurring in S that DP(S)=False iff S is unsatisfiable.

*Case 1: S has no proposition symbols*; S is empty, hence satisfiable, and result = True by (Step 1) is correct
*Case 2: S contains one proposition symbol*; either (Step 4) is possible and the clauses are satisfiable and the correct result of True will be returned by (Step 1) or S ={L}+{¬L} (two clauses) and is unsatisfiable so result = False by (Step 2) is correct. (Note that if tautologies are initially removed they will never appear in the argument S, so S cannot be the *clause* L∨¬L.)

## Proof of Correctness of DP continued:

1civ

*Case 3: S has k>1 proposition symbols.* Assume for induction hypothesis (IH) that if S has <k proposition symbols then DP(S) returns the correct result (i.e. False when S is unsatisfiable, otherwise True). For each of Steps 3-6 S'/S" has fewer atoms than S and so by (IH) DP(S')/DP(S") returns the correct result, which is also the correct result for S according to the invariant. For Step 7, the (IH) states that DP(S') and DP(S") give the correct result for S' and S". The disjunction will be false when both are false, i.e. when S' and S" are both unsatisfiable. By the invariant S is unsatisfiable and so the disjunction gives the correct result (false) for S.

Exercise 4: Show that when the disjunct is true the correct result is also given.

Next we show that another useful property from Slide 1bv (for the 2 argument variant) holds:
if a literal is in M then neither it nor its complement occurs in S.
Assume this is true for a call DP(M,S). If any literal L is added to M then all occurrences of L and ¬L are removed when forming S' / S" and no literals are added to S' / S" that were not in S. Hence the property still holds. It clearly holds for the initial call DP({ }, S).

Next we show the invariant property for the full procedure that returns a model. The subsequent induction part is quite similar to the proof just given and will be omitted.

Exercise 5: Proofs of the invariant are given for Steps 4 and 7 on slide 1cv. Give the proofs for Steps 3, 5 and 6.

## Proof of Correctness of DP concluded:

1cv

To show the invariant property you must show for each step that M∪S is satisfiable iff M' ∪S' is satisfiable. For example, the case for (Step 4) is as follows.

Assume P is pure, then the procedure adds P to M to give M'. We assume that literals in M do not occur in S, which implies P is not in M.

First we show M∪S is satisfiable ==> M' ∪S' is satisfiable. Let I be a model of M and of S. Since S ⊃S' I is a model of S'. If I makes P true then I satisfies M'=M ∪ {P}. If I makes P false, let C = P∨D be a clause in S (P occurs only in such clauses). I makes D true, hence can form I' from I by reassigning P to true in I. P is not in M or S', so I' will still satisfy M ∪ {P} ∪ S' (ie M' ∪ S').

Next we show that M' ∪S' is satisfiable ==> M∪S is satisfiable. Suppose I is a model of M' and S'. Since P is in M' I makes P true and hence makes S true as it satisfies the deleted clauses like C = P∨D. I makes M true since M' ⊇M. Hence the property holds for step 4.

For (Step 7) assume atom L is chosen. Let I be a model of M∪S. If I makes L true then we show I satisfies S' and M'=M∪{L}. The analogous case for when I makes L false using S" is similar. I clearly makes M' true. (Remember that L is in S and so by assumption L does not occur in M.) Consider the exemplifying clause ¬L∨B∨C in S. Since L is true in I, B∨C is forced to be true in I, as required to satisfy S'. Clauses in S not including ¬L or L are unaffected and are still true (in S'). On the other hand, if M' ∪S' has a model I, then I satisfies L and hence all the clauses deleted from S to form S'. I still satisfies M and clauses in S such as ¬L∨B∨C since I also satisfies B∨C in S'. Similarly, if M" ∪S" has a model.

## "The three little girls" problem

1di

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)) — Only one of the three girls was the culprit
(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) ∨ ¬((H(d) ^ C(d)) →C(e)) — (Frances's negated)

Here we include a negated conclusion ¬C(f) and look for False.
The other case, to look for True (and C(f) in every model), is left to you.

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

(Next week's notes will include a systematic algorithm for conversion to clauses. For now we do it by hand.)

## Solution to "The three little girls" by DP (see ppt)

1dii

| | | |
|---|---|---|
| (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) | (9) ¬C(f) ( neg conc) |

**Merge literals in (7a) to obtain C(e) ∨ C(f) and remove (8a) (tautology);**
(7a) subsumes (7b), (7c), (1); (9) ¬C(f) subsumes (2c), (4) and (5);

**call DP( [ ], [2a, 2b, 3, 6, 7a, 8b, 8c, 9] );**
Apply (Step 6) on ¬C(f) and then apply (Step 4) as H(e) is pure;
**call DP([H(e) , ¬C(f)], [2a, 3, 6, C(e), ¬H(d) ∨ C(d), ¬H(d) ∨ ¬C(e)]);**

Apply (Step 5) on C(e) and then apply (Step 3) as ¬H(d) subsumes {¬H(d),C(d)};
**call DP([H(e) , ¬C(f) , C(e)], [2a, ¬C(d), H(d) ∨ C(d), ¬H(d)]);**

Apply (Step 6) on ¬C(d);
**call DP([H(e), ¬C(f), C(e), ¬C(d)], [H(d),¬H(d)]);**

Apply (Step 2) on H(d) - terminate and return False.

## Theorems for DP

1ei

DP([ ],S) halts with *false* if S has no models

DP([ ], S) halts with *true* and returns at least one model M if S is satisfiable

In fact, M is a partial model

Atoms A s.t. neither A nor ¬A occur in M can be either true or false

EG: DP([ ], [A∨B]) will return either the model {A} or the model {B}.

The model {A} can be extended to the model {A,B} or to {A, ¬B} as both satisfy the clause A∨B. Analogously for the model {B}.

Arguments of calls to DP(M,S) satisfy the *invariant*:

M +S has a model iff either
 M' + S' has a model, (in single call cases), or
 at least one of M' + S' or M'' + S'' has a model (in otherwise case).

Also, if literal L(or ¬L) occurs in M then neither L nor ¬L occurs in S.

## Implementing DP efficiently (1)

1fi

The DP algorithm was invented in 1960. It is still widely used for proving unsatisfiability for propositional logic - eg in PVS, Prover9, all 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 when a clause becomes a unit clause so can apply (Step 5) or (Step 6)
- check when a clause is subsumed (can ignore it)
- check when a clause is empty (branch finished);
- eliminate complements of singletons (Steps 5, 6 or 7)
- and choose atom for split in (Step 7)

## Implementing DP efficiently (2)

1fii

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 (2 literals so indicated)
- singleton (only 1 literal indicated)
- empty clauses (no literals indicated)

They also recorded the last literal decision affecting a clause and whether the clause is deleted or not.

This

- helps with choice of selected literal
- aids restoration of states when back-tracking to alternative branches

Back-tracking was improved by using information about

- why decisions were forced (Steps 5/6)
- which atoms are contradictory in a False branch

## Implementing DP efficiently: Example (see ppt)

1fiii

$\boxed{a}$ $\boxed{\neg b}$ $\boxed{e}$ $\boxed{g}$ $\boxed{\neg h}$

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

1. If g set to True then clause can be ignored
2. If g set to False then select new watched
 - must choose ¬b as it is the only literal not yet set and not yet watched
3. If b set to False then clause can be ignored
4. If b set to True then ¬b is set also - a and g are still the watched literals
5. After 2: Suppose next a set to False;
 then only ¬b is watched so forces b = False

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

## Implementing DP efficiently (3)

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

1fiv

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

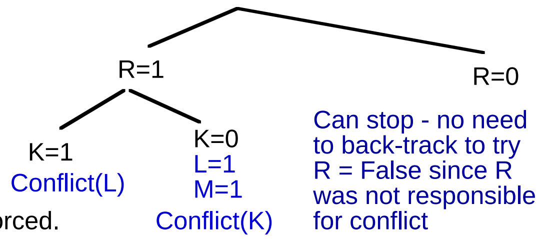
Idea is to keep track of forced decisions:

Choose R=1 (True)

Choose K=1

Undo K=1

Now K=0 (False)



Can stop - no need to back-track to try R = False since R was not responsible for conflict

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

R=1

$$\begin{tikzcd} \boxed{K=1} \arrow[r] \arrow[dr] & L=0 \\ & L=1 \end{tikzcd}$$

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

$$\begin{tikzcd} \boxed{K=0} \arrow[r] & L=1 \arrow[r] & M=1 \arrow[r] & K=1 \end{tikzcd}$$

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)

1fv

• 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) | |

$$\begin{tikzcd} & Hd=1 & \\ \boxed{Cd=1} \arrow[ur, "2a"] \arrow[r, "3"] \arrow[dr, "4"] & Ce=0 \arrow[r, "7a"] & Cf=1 \\ & Cf=0 & \end{tikzcd}$$

Conflict in Cf due to Cd=1
==> conflict clause ¬Cd

$$\begin{tikzcd} Cd=0 \arrow[r, "6"] & Hd=1 \arrow[r, "8b"] & Cf=1 \\ Ce=1 \arrow[r, "3"] \arrow[dr, "5"] & He=1 & \\ & Cf=0 & \end{tikzcd}$$

Conflict in Cf due to Cd=0 and Ce=1
clause Cd ∨ ¬Ce
 ==> ¬Ce (using ¬Cd)
 ==> Cf (by 7a)
 ==> Hf (by 2c)

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

## Summary of Slides 1

1gi

**1.** Definitions (for propositional logic) of *ground atom, ground literal, valuation, ground clause, satisfiable, unsatisfiable*, (ground) *subsumes, tautology, merge, pure literal* and *logical implies* (|=) were given.

**2.** If S |=G then every model in the language of S and G that satisfies S also satisfies G and hence does not satisfy ¬G. Therefore there is no model of {S,¬G}.

**3.** The Davis Putnam (DP) method for testing satisfiability of propositional clauses was described.

**4.** DP returns True for given set of clauses S if S has no models. It returns False if there S has a model and will then also return a model for S – i.e. an assignment of T/F to atoms in S that makes every clause in S true. This assignment can be extended to all atoms by assigning T or F to any remaining unassigned atoms.

**5.** The state of DP can be represented as a tree, in which each node is labelled by the current set S and current partial assignment. Each call (and subcalls) of DP(M,S) maintains an invariant: M is a partial model of S iff M' is a partial model of S', where the subcall is DP(M',S').

**6.** The correctness property of DP is proved by induction on the number of atoms occurring in the current clause set.

**7.** DP was used to solve the "Three Little Girls" problem.

**8.** Heuristics for efficient implementation of DP were given (Chaff).

## A Solution to Aunt Agatha's Burglary

Note: (x and y are implicitly universally quantified)

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

First simplify (8) to $\exists z\, \neg d(x,z)$ and then to $\neg d(x, f(x))$.
(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.

<u>m=j</u>: $s(j,a)$ (by equality substitution in (1)) and hence $d(j,a)$ and $\neg r(j,a)$ from (3);
From (5) and (9) $d(a,a)$ hence $\neg d(j,a)$ by (4);
Contradiction so $m \neq j$.

<u>m=b</u>: $\neg d(b,f(b))$ (*) (put x as b in (8) );
hence $\neg d(a,f(b))$ by (7) and $\neg\neg f(b)=b$ by (5);
Therefore $f(b) = b$ and $\neg d(b,b)$ from (*);
$s(b,a)$ (by equality substitution in (1)) and hence $d(b,a)$ and $\neg r(b,a)$ from (3);
Hence $d(b,b)$ from (6);
Contradiction so $m \neq b$;

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"

(1) $C(d) \lor C(e) \lor C(f)$
(2) $C(x) \rightarrow H(x)$ (x is implicitly universally quantified)
(3) $\neg(C(d) \wedge C(e))$
(4) $\neg(C(d) \wedge C(f))$
(5) $\neg(C(f) \wedge C(e))$
(6) $C(d) \lor H(d) \lor \neg C(e)$ (Dolly's statement negated)
(7) $C(e) \lor C(f) \lor \neg(C(e) \rightarrow (C(d) \lor H(d)))$ (Ellen's negated)
(8) $C(f) \lor \neg H(d)$ or $\neg((H(d) \wedge C(d)) \rightarrow C(e))$ (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.

( $\neg(X \rightarrow Y)$ true means X true and Y false.)

**Case 2:** C(e) Suppose Ellen did it. Then H(d) follows from(6)
(remember $\neg C(d)$ 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

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

Show $b \circ a = c$

| | |
|---|---|
| $x \circ (a \circ b) = (x \circ a) \circ b$ | (put y as a and z as b in (2)) |
| $x \circ c = (x \circ a) \circ b$ | (use $a \circ b = c$) |
| $c \circ c = (c \circ a) \circ b$ | (put x as c) |
| $e = (c \circ a) \circ b$ | (use (3) $c \circ c = e$) |
| $e \circ b = e \circ b$ | (property of =) |
| $e \circ b = ((c \circ a) \circ b) \circ b$ | (use $e = (c \circ a) \circ b$) |
| $b = ((c \circ a) \circ b) \circ b$ | (use (4) $e \circ b = b$) |
| $b = (c \circ a) \circ (b \circ b)$ | (use (2)) |
| $b = (c \circ a) \circ e$ | ( use (3) $b \circ b = e$) |
| $b = c \circ a$ | (use (4) $(c \circ a) \circ e = c \circ a$) |

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