AUTOMATED REASONING

SLIDES 1:

PROPOSITIONAL TECHNIQUES (1)
Basic Concepts
Davis Putnam Procedure
Solving the "Three Little Girls" Problem

KB - AR - 09

Techniques and Examples for Propositional Data

For propositional sentences there are some special techniques - e.g.
Davis Putnam, simple model generation (use clausal form) OBDDs

A few notations for ground clauses (no quantifiers):

- the *language* L names the atoms that may occur in any 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

Interpretations for Propositional Data

1aii

- an *interpretation over L* assigns True(T) or False(F) to every atom in L;
- an interpretation over L assigns T/F to literal ¬X if it assigns F/T to atom X;
- an interpretation over L satisfies clause C if it assigns T to ≥1 literal X in C;
- an interpretation 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;
- let S be a set of clauses and G be a clause in L. Then S $\models$ G iff for all interpretations M over L, if M is a model of S, then M is a model of G;
- S+{¬ G} has no models (is *unsatisfiable*) iff S $\models$ G;
- clause X *subsumes* clause Y if X $\subseteq$ Y. Note X $\models$ Y; Why?

Examples:
Given the clauses A∨B, ¬A∨C∨¬D, ¬C
The interpretation A=B=C=D= False is **not** a model. **Why?**
The interpretation A=B=True; C=D= False **is** a model. **Why?**
The given clauses are therefore *satisfiable*

The clauses A∨B, ¬A, ¬B are *unsatisfiable*. **Why?**
A∨B, ¬A $\models$ B since every model of A∨B and ¬A is a model of B.

¬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.

Basically it attempts to show S has no models, by investigating whether simpler sets of clauses derived from S have no models.
Either no model exists and false is returned,
or a model can be found and true is returned together with a model.

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.
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), then call DP([], S)

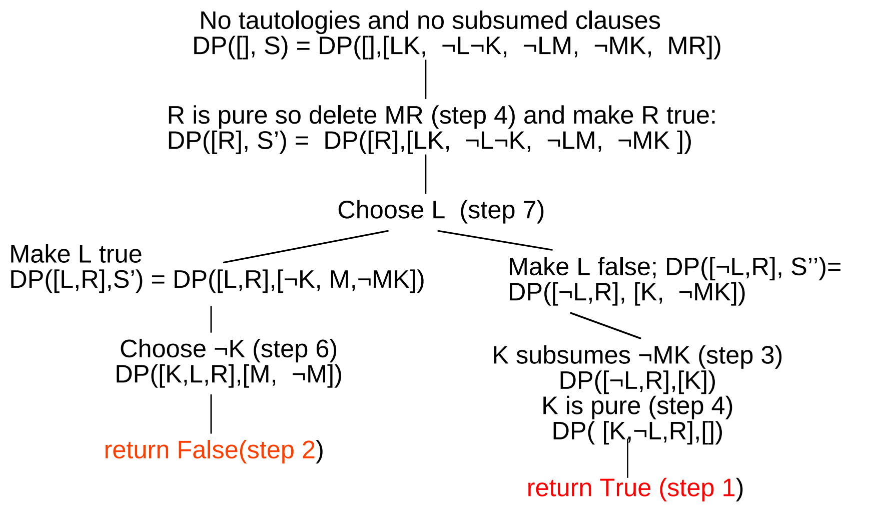
The algorithm is on 1bii and an example is on 1biii.

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  print 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 (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")
```

(Note that [P|M] is the Prolog list notation for a list with head P and tail M.)
The partial model M from step 1 (ie. if A is in M then atom A is assigned T and if ¬A is in M then atom A is assigned F) can be extended to be a model by assigning either of T or F to any still unassigned atom in the language.

Davis Putnam Example written as a tree



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])
Make L false; DP([¬L,R], S'')=
DP([¬L,R], [K, ¬MK])
Choose ¬K (step 6)
DP([K,L,R],[M, ¬M])
K subsumes ¬MK (step 3)
DP([¬L,R],[K])
K is pure (step 4)
DP([K,¬L,R],[])
return False(step 2)
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

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 1biii. (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 1biii, but the final answer 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 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 had included L¬K in 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 if it doesn't include assignments for all atoms. The second argument is simplified at each step.

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

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

(2a) At each single branching step M∪S is satisfiable iff M' ∪S' is satisfiable.

(2b) At each branching step M∪S is satisfiable iff M' ∪S' is satisfiable or M'' ∪S'' is satisfiable.

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.

Why Does it Work? (1)

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

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

i.e. in the call DP(S) S is satisfiable
iff
in the next call DP(S') S' is satisfiable

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

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

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 it Work? (2)

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 ¬P does not occur in any clause in S'
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

Proof of Correctness of DP:

It is quite easy to show the simplified invariant property. For each step you must show that the clauses before the step are satisfiable iff the clauses after the step are satisfiable. For example, the case for (Step 4) is given on Slide 1cii. *The other cases are left as exercises.*

Using the invariant you can then show DP(S)=False iff S is unsatisfiable by induction on the number of occurrences of atoms (positive literals) in S. If S has no atoms, then S is either empty, hence satisfiable and result = True by (Step 1) is correct, or S contains only negative literals. In that case (Step 4) can be applied to remove all clauses from S until S is empty. If S contains one atom and (Step 4) is not possible, then S ={L}+{¬L} (two clauses) and is unsatisfiable so result = False by (Step 2) is correct. (Note that if tautologies abe initially removed they will never appear in the argument S, so S cannot be the *clause* L∨¬L.) Suppose S has k>1 atoms. The induction hypothesis (IH) states that if S has <k atoms then DP(S) returns the correct result (i.e. False when S is unsatisfiable, otherwise True). For cases 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. In case 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. The argument when the disjunct is true is left to you.

Next we show the invariant property for the full procedure that returns a model. The induction part is similar to the proof just given and will be omitted. First we show that the property on Slide 1biv 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 is added to M then all occurrences are removed from the clauses in S' and no literals are added to S' that were not in S. Hence the property still holds. It clearly holds for the initial call DP({ }, S).

Proof of Correctness of DP continued:

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. 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 is such clauses). I makes D true, hence can reassign P t true in I. P is not in M or S', so I will still satisfy M∪{P}∪S' since S⊇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 L, B∨C is forced to be true in I as required to satisfy S'. Clauses in S' not including ¬L are unaffected and still true. 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 satisfies B∨C which is in S'. Similarly if M'' ∪S'' has a model.

The other cases are left as exercises.

"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 propsitional 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)

Either include a negated conclusion ¬C(f) and look for False, or look for True with C(f) in the model. Here we add ¬C(f).

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

Next week we'll see a systematic algorithm for conversion to clauses. For now we do it by hand.

Solution to "The three little girls" by DP

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.

For the non-clausal case (see Slide 1eii) similar properties hold.
e.g. try DP([], S), for the tautology ((A -> B) -> A) -> A.

The Davis Putnam procedure can be generalised to apply to arbitrary propositional sentences as outlined below. The same kind of argument as given for the clausal version shows it to be correct.

We say A is +ve/-ve in A/¬A if A is an atom. A is +ve/-ve in ¬C if A is -ve/+ve in C.
A is +ve/-ve in C∧D,C∨D, if A is +ve/-ve in C or in D.
A is +ve/-ve in C→D if A is +ve/-ve in D or A is -ve/+ve in C.
A is +ve and -ve in C<->D if A occurs in C or D.

Simplify(S) applies laws of logic for True/False to clauses in S. eg True∧X≡X; False∨X≡X.
The DP-procedure is amended as shown below. Note that the subsumption test is optional.
Tautologies are only removed if they are obvious, and similarly for merging.

```
procedure DP(M, S) : boolean;
%M is a model so far and S are sentences to process
1. If S is empty print M and return false;          %M is a partial model
2. If S contains literals X and ¬X return true;     % S has no models
(3. If C is a subsumed sentence in S return DP(M,S-C);)
(4. If P is a pure literal in S (i.e. occurs only +ve or only -ve in S) return DP([P|M], S1),
where S1 is as in Step 5)
5. If A is a fact in S return DP([A|M], S'),
   where S'=Simplify(S1) and S1={C|C in S and A is replaced by True in C}
6. If ¬A is a fact in S return DP([¬A|M ], S"),
    where S"=Simplify(S2) and S2={C|C in S and A is replaced by False in C};
7. Otherwise select an atom A in S and form S' and S" as in Steps 5 and 6;
     return DP([A|M ], S') ∧ DP( [¬A|M], S")
```

(Normal forms other than clauses are possible, and procedures analogous to DP devised.)

1eii

Implementing DP efficiently (1)

1fi

The DP algorithmwas invented in 1960. It is still widely used for proving unsatisfiability for propositional logic - eg in PVS, Otter, Prover9, etc.

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 determine:

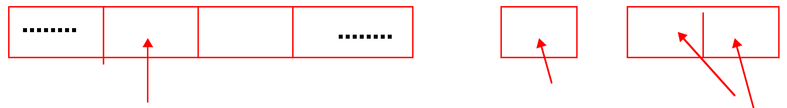
- if a clause is a new unit clause (so can apply (Step 4) or (Step 5));
- if a clause is deleted by subsumption or empty (can ignore or finished);
- eliminate complements of singletons and choose atom for split in (Step 7)

In the system **Chaff (2001) Moskewicz et al** keep a pair of indices associated with every clause that indicate 2 literals in the clause that have not yet been eliminated. Also indicate which was last literal decision affecting clause and whether clause is deleted or not.

This allows to detect non-singletons, singleton (only 1 literal indicated), empty clauses (no literals indicated) and help with choice of selected literal. It also helps restore states when back-tracking to alternative branches.

Example: Implementing DP efficiently (2)

1fii



Current value of atom after last step:
0 = atom not present
1 = pos atom present atom not set
2 = neg atom present atom not set
3 = pos/neg atom set

eg A1 $\vee$ ¬A3 initially coded as [1,0,2]

Clause deleted?
0 = not deleted
n>0 = deleted at n

Monitors

Also record Step#, step type, literal selected; Initial step# = 0

Step 7 will update record with decision by altering all clause entries that are not in a deleted clause and for which atom is present.

eg if step sets atom 5 to true then 5th entry is inspected in all undeleted clauses; if 1 clause deleted (at step#); otherwise 0-->0; 2-->3. If 5th entry is monitored and went from 2-->3 then must get new monitor. If none, this will trigger step 5 (or 6) depending on the other monitor.

Step 5 will update record in a similar way to above. If a monitor pair in a non-deleted clause becomes empty then branch closed so backtrack. Entails undoing earlier steps until reach a Step 7 with a second branch to investigate. If all clauses deleted ==> model found. **Exercise:** How can it be read from state?

Implementing DP efficiently (3)

1fiii

EG: 1. A$\vee$B 2. ¬A$\vee$B 3. ¬B$\vee$C 4. ¬B$\vee$¬C

Code as: [1,1,0], [2,1,0], [0,2,1], [0,2,2]
Monitors are AB, AB, BC, BC

Initial Trace:
1. Step 7 set A to true: (affects clauses 1, 2) (1) deleted at #1
2--> [3,1,0] Monitor B (2, 3, 4 left)
2. Step 5 set B to true: (2) deleted at #2
3--> [0,3,1] Monitor C 4--> [0,3,2] Monitor C (3, 4 left)
3. Step 6 set C to false: (4) deleted at #3 3-->[0,3,3] No Monitor
No monitor to 3 so backtrack redo last Step 7

1. Step 7 Set A to False: (affects clauses 1,2) (2) deleted at #4
1--> [3,1,0] Monitor B (1,3,4 left)
2. Step 5 set B to true: (1) deleted at #5
3--> [0,3,1] Monitor C 4--> [0,3,2] Monitor C (3, 4 left)
4. Step 5 set C to true: (3) deleted at #6 4-->[0,3,3] No Monitor
No monitor for 4 and no more choices

Return no models.

Exercise:
1.Check the backtracking can be made from the various state values.
2. A better choice at Step 1 is to select B. Try it!

Summary of Slides 1

1gi

1. The Davis Putnam(DP) method for testing satisfiability of propositional clauses was described. DP can be extended to arbitrary propositional sentences.

2. DP returns True for given set of clauses S if there are no models of S. It returns False if there is at least one model of S and will in that case 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.

3. 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').

4. The correctness property of DP is proved by induction on the number of atom occurrences in the current clause set.

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

6. Definitions of the terms ground atom, ground literal, assignment, ground clause, satisfiable, unsatisfiable, (ground) subsumes, tautology, merge, pure literal and logical implies (|=) were given. 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}.

7. Some ideas for Implementation were discussed.