AUTOMATED REASONING

SLIDES 1:

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

KB - AR - 12

Techniques and Examples for Propositional Clauses

For propositional sentences there are some special techniques
First up is **Davis Putnam** 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 and ¬(A∨B) is equivalent to ¬A and ¬B

- clause X *subsumes* clause Y if X ⊆ Y. Note X |= Y; **Why?**

Example:

¬A∨¬D *subsumes* ¬A∨C∨¬D. Note ¬A∨¬D |= ¬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 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

1bii

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")

(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 Procedure:

1biv

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 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 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:

1bv

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 positively, or negatively, in S. This is clearly true at the start.

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

(2b) At each double branching step
$M \cup S$ is satisfiable iff $M' \cup S'$ is satisfiable or $M'' \cup S''$ 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.

Why Does DP Work? (1)

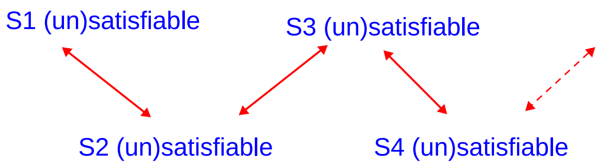
1ci

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 (iff is shorthand for if, and only if, or <-->)
in the next call DP(S') S' is satisfiable

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



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 atoms 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 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 continued: 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 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 with C(f) in the model, is left to you.

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 (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.

Summary of Slides 1

1eii

1. Definitions of the terms *ground atom, ground literal, valuation, 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}.

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

3. 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.

4. 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').

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

6. DP was used to solve the “Three Little Girls” problem.