Propositional Logic and Satisfiability

Slides draw upon material from:
Prof. Bart Selman
Cornell University

Brian C. Williams
16.410-13
October 13th, 2010

Assignments

- Assignment:
  - Problem Set #5: Activity Planning, due today Wednesday, October 13th, 2010.
  - Problem Set #6: Propositional Logic and Satisfiability, out today; due October 27th, 2010 (in 2 weeks).
- Reading:
  - Today: *[AIMA] Ch. 7, 8*
  - Monday: *TBD*
- Exam:
  - Mid-Term - October 20th.

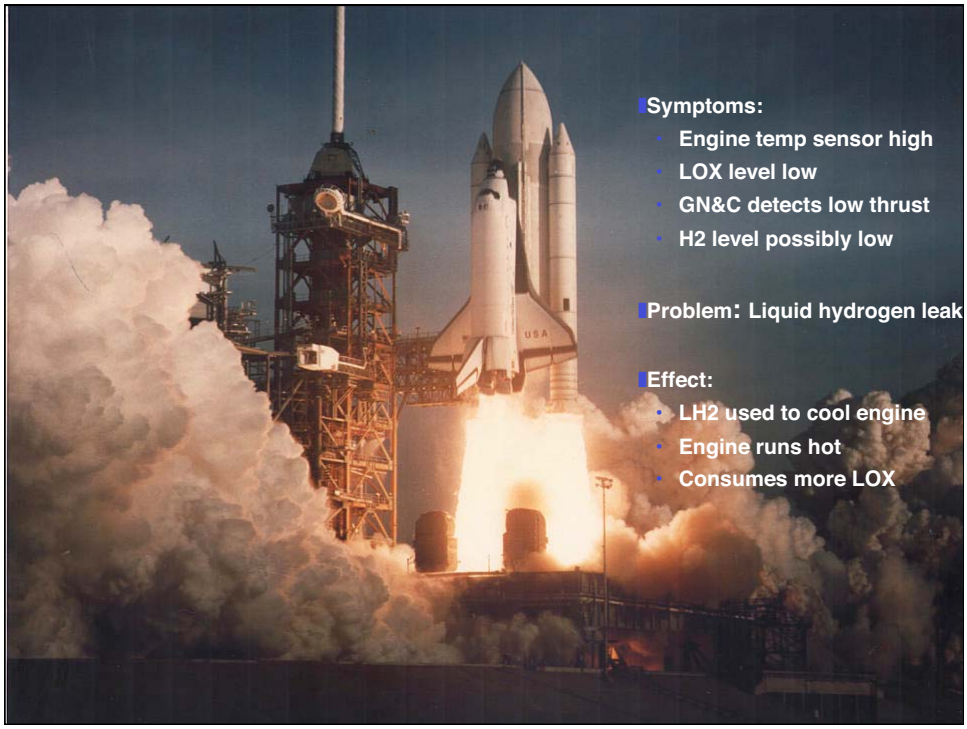
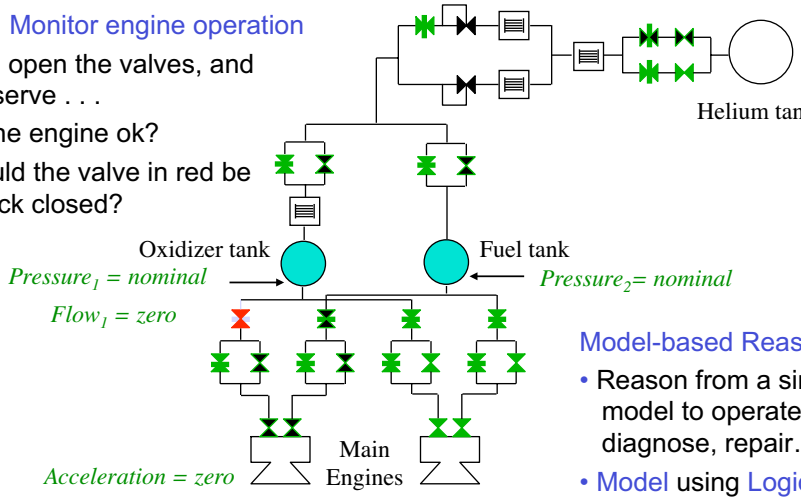


Image credit: NASA.

How Do We Reason About Complex Systems using Commonsense Models?

Task: Monitor engine operation

- You open the valves, and observe . . .
- Is the engine ok?
- Could the valve in red be stuck closed?



Model-based Reasoning:

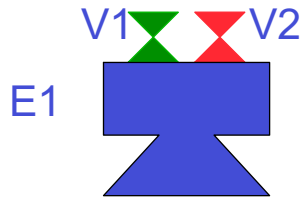
- Reason from a single model to operate, diagnose, repair…
- Model using Logic.
- Reason using Sat.

Brian Williams, Fall 10

4

Modeling an Engine in Propositional Logic

"An Engine E1 can either be okay, or broken in some unknown way. When E1 is okay, it will thrust when there is a flow through V1 and v2."



(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown) and
(mode(E1) = ok implies
  (thrust(E1) = on if and only if flow(V1) = on and flow(V2) = on))

Reasoning From the Model

Monitoring:
Are the observations O consistent with model M?

Fault Diagnosis:
What fault modes of M are consistent with O?

Reconfiguration:
What component modes of M produce behavior G?

⇒ Propositional Satisfiability:

Find a truth assignment that satisfies some logical sentence S:

1. Reduce S to clausal form.
2. Perform search similar to MAC = (BT+CP)
   [Davis, Logmann & Loveland, 1962]

Propositional Satisfiability

Find a truth assignment that satisfies logical sentence T:

- Reduce sentence T to clausal form.
- Perform search similar to MAC = (BT+CP)
  [Davis, Logmann & Loveland, 1962]

Propositional satisfiability testing:

1990: 100 variables / 200 clauses (constraints)
1998: 10,000 - 100,000 vars / 10^6 clauses
2010: millions

Novel applications:

e.g. diagnosis, planning, software verification, circuit testing, machine learning, and protein folding

What Formal Languages Exist for Describing Constraints?

• Algebra	values of variables
• Probability	degree of belief
• Propositional logic	truth of facts
• Temporal logic	time, ….
• Modal logics	knowledge, belief …
• First order logic	facts,objects,relations

Outline

- Propositional Logic
  - Syntax
  - Semantics
  - Reduction to Clauses
- Propositional Satisfiability
- Empirical, Average Case Analysis
- Appendices

Logic in General

- Logic
  - A formal language for representing information that can be used to draw conclusions.
  - About the truth of statements and their consequences.
- Syntax
  - Defines the expressible sentences in the language.
- Semantics
  - Defines the "meaning" of these sentences

  ⇨ truth of a sentence in some world.

Logic Example: Arithmetic

- Syntax – legal sentences
  - "X + 2 > Y" is a legal sentence.
  - "X 2 + Y >" is not a legal sentence.
- Semantics - truth in world
  - "X + 2 > Y" is true iff the number X + 2 is not less than or equal to the number Y
  - "X + 2 > Y" is true in a world where X = 7, Y = 1
  - "X + 2 > Y" is false in a world where X = 0, Y = 6

Propositional Logic: Syntax

Propositions

- A statement that is true or false
  - (valve v1)
- Assignments to finite domain variables - State Logic
  - (= voltage high)

Propositional Sentences (S)

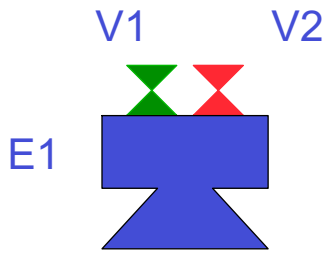
- S ::= proposition |
- (NOT S) |
- (OR S1 ... Sn) |
- (AND S1 ... Sn)

Defined Constructs

- (implies S1 S2) => ((not S1) OR S2)
- (IFF S1 S2) => (AND (IMPLIES S1 S2)(IMPLIES S2 S1))

Propositional Sentences: Engine Example

(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown) and
(mode(E1) = ok implies
  (thrust(E1) = on if and only if
    flow(V1) = on and flow(V2) = on))



Outline

- Propositional Logic
  - Syntax
  - Semantics
  - Reduction to Clauses
- Propositional Satisfiability
- Empirical, Average Case Analysis
- Appendices

Propositional Logic: Semantics

Interpretation I of sentence S assigns true or false to every proposition P in S.

- S = (A or B) and C
- I = {A=True, B=False, C=True}
- I = {A=False, B=True, C=False}

All Interpretations →

A	B	C
True	True	True
True	True	False
True	False	True
True	False	False
False	True	True
False	True	False
False	False	True
False	False	False

Propositional Logic: Semantics

The truth of sentence S wrt interpretation I is defined by a composition of Boolean operators applied to I:

- "Not S" is True iff "S" is False

Not S	S
False	True
True	False

Propositional Logic: Semantics

The truth of sentence S_i wrt Interpretation I:

- "Not S" is True iff "S" is False
- "S_1 and S_2" is True iff "S_1" is True **and** "S_2" is True
- "S_1 or S_2" is True iff "S_1" is True **or** "S_2" is True

S1 and S2	S1	S2
True	True	True
False	True	False
False	False	True
False	False	False

S1 or S2	S1	S2
True	True	True
True	True	False
True	False	True
False	False	False

Propositional Logic: Semantics

The truth of sentence S_i wrt Interpretation I:

- "Not S" is True iff "S" is False
- "S_1 and S_2" is True iff "S_1" is True **and** "S_2" is True
- "S_1 or S_2" is True iff "S_1" is True **or** "S_2" is True
- "S_1" implies "S_2" is True iff "S_1" is False or "S_2" is True
- "S_1 iff S_2" is True iff "S_1implies S_2" is True and "S_2 implies S_1" is True

Example: Determining the Truth of a Sentence

(mode(E1) = ok implies
  [(thrust(E1) = on if and only if (flow(V1) = on and flow(V2) = on)) and
   (mode(E1) = ok or mode(E1) = unknown) and
   not (mode(E1) = ok and mode(E1) = unknown)])

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Example: Determining the Truth of a Sentence

(True implies
  [(False if and only if (True and False)) and
   (True or False) and
   not (True and False)])

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Example: Determining the Truth of a Sentence

(True implies
[(False if and only if (True and False)) and
(True or False) and
not (True and False)])

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Example: Determining the Truth of a Sentence

(True implies
[(False if and only if (True and False)) and
(True or False) and
not False])

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Example: Determining the Truth of a Sentence

(True implies
[(False if and only if (True and False)) and
(True or False) and
True])

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Example: Determining the Truth of a Sentence

(True implies
[(False if and only if False) and
True and
True])

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Example: Determining the Truth of a Sentence

(True implies
[(False if and only if False) and
True and
True])

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Example: Determining the Truth of a Sentence

(True implies
[(False implies False) and (False implies False)) and
True and
True])

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Example: Determining the Truth of a Sentence

(True implies
[(not False or False) and (not False or False)) and
True and
True])

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Example: Determining the Truth of a Sentence

(True implies
[(True or False) and (True or False)) and
True and
True])

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Example: Determining the Truth of a Sentence

(True implies
[(True and True) and
True and
True])

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Example: Determining the Truth of a Sentence

(True implies
[True and
True and
True])

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Example: Determining the Truth of a Sentence

(True implies
True)

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Example: Determining the Truth of a Sentence

(not True or
True)

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Example: Determining the Truth of a Sentence

(False or
True)

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Example: Determining the Truth of a Sentence

True!

If a sentence S evaluates to True in interpretation I, then:

- "I *satisfies* S"
- "I is a *Model* of S"

Interpretation:

mode(E1) = ok	is True
thrust(E1) = on	is False
flow(V1) = on	is True
flow(V2) = on	is False
mode(E1) = unknown	is False

Satisfiability versus Validity

Satisfiable

A sentence is satisfiable if there is an interpretation (a truth assignment) that makes the clause true.

- (not A or B) *is satisfiable.*
- (A implies not B) and (A implies B) *is unsatisfiable.*

Valid

A sentence is valid if it is true for all interpretations.

- Is (not A or A or B) valid?

  Yes, it is valid over all possible interpretations.
- Is (A or B) valid with respect to the interpretations {A=true, B=false} and {A=false, B=false}?

Outline

- Propositional Logic
  - Syntax
  - Semantics
  - Reduction to Clauses
- Propositional Satisfiability
- Appendices

Propositional Clauses: A Simpler Form

- Literal: A proposition or its negation.
  - B, Not A
- Clause: A disjunction ("or") of literals.
  - (not A or B or E)
- Conjunctive Normal Form:
  A conjunction ("and") of clauses.
  - Φ = (A or B or C) and
    (not A or B or E) and
    (not B or C or D)
  - Represented by a set of clauses.

Reduction to Clausal Form: Engine Example

(mode(E1) = ok implies
(thrust(E1) = on iff (flow(V1) = on and flow(V2) = on))) and
(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown)

not (mode(E1) = ok) or not (thrust(E1) = on) or flow(V1) = on;
not (mode(E1) = ok) or not (thrust(E1) = on) or flow(V2) = on;
not (mode(E1) = ok) or not (flow(V1) = on) or not (flow(V2) = on)
or thrust(E1) = on;
mode(E1) = ok or mode(E1) = unknown;
not (mode(E1) = ok) or not (mode(E1) = unknown);

Reducing Propositional Formula to Clauses (CNF)

See Appendix for Detailed Example:

1) Eliminate iff and implies
   - E1 iff E2 => (E1 implies E2) and (E2 implies E1)
   - E1 implies E2 => not E1 or E2

2) Move negations in, towards propositions, using De Morgan's Theorem:
   - not (E1 and E2) => (not E1) or (not E2)
   - not (E1 or E2) => (not E1) and (not E2)
   - not (not E1) => E1

3) Move conjunctions out using Distributivity
   - E1 or (E2 and E3) =>(E1 or E2) and (E1 or E3)

Outline

- Propositional Logic
  - Syntax
  - Semantics
  - Reduction to Clauses
- Propositional Satisfiability
- Empirical, Average Case Analysis
- Appendices

Propositional Satisfiability

Input: A Propositional Satisfiability Problem is a pair <P, Φ >, where:

- P is a finite set of propositions.
- Φ is a propositional sentence over P
  - We assume it is reduced to a set of clauses.

Output: True iff there exists a model of Φ.

Is an instance of a CSP:

- Variables: Propositions
- Domain: {True, False}
- Constraints: Clauses

Models of <P, Φ >

- An interpretation is a truth assignment to all propositions P.
- A model is an interpretation such that all clauses are satisfied:
  - A clause is satisfied iff at least one literal is true.
  - A clause is violated iff all literals are false.

Example: C1: Not A or B
C2: Not C or A
C3: Not B or C

Testing Satisfiability of <P, Φ >

1. Apply systematic, complete procedure
   - BT + unit propagation, shortest clause heuristic
     - [Davis, Logmann, & Loveland 1962; Crawford & Auton 1997; Nayak & Williams, 1997]
2. Apply stochastic, incomplete procedure
   - [Minton et al. 90; Selman et. al 1993] – see Appendix
3. Apply exhaustive clausal resolution
   - [Davis, Putnam 1960; Dechter Rish 1994]

Testing Satisfiability of <P, Φ >

1. Apply systematic, complete procedure
   - BT + unit propagation, shortest clause heuristic
     [Davis, Logmann, & Loveland 1962]
   - State-of-the-art implementations:
     - ntab [Crawford & Auton, 1997]
     - itms [Nayak & Williams, 1997]
     - many others! *See SATLIB 1998 / Hoos & Stutzle*
2. Apply stochastic, incomplete procedure (Appendix)
   - MinConflict [Minton et a. 90]
   - GSAT/WalkWat [Selman et. al 1993)]– see Appendix
3. Apply exhaustive clause resolution (Not Covered)
   - [Davis, Putnam, 1960]

Outline

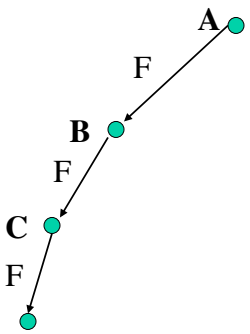
- Propositional Logic
- Propositional Satisfiability
  - Backtrack Search
  - Unit Propagation
  - DPLL: Unit Propagation + Backtrack Search
- Empirical, Average Case Analysis
- Appendices

Propositional Satisfiability using Backtrack Search

- Assign true or false to an unassigned proposition.
- Backtrack as soon as a clause is violated.

Example:

- C1: Not A or B **satisfied**
- C2: Not C or ~~A~~ **satisfied**
- C3: Not B or C **satisfied**

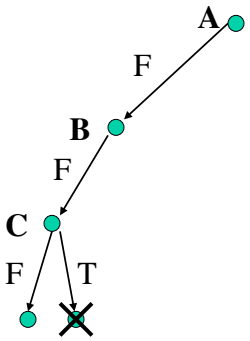


Propositional Satisfiability using Backtrack Search

- Assign true or false to an unassigned proposition.
- Backtrack as soon as a clause is violated.

Example:

- C1: Not A or B **satisfied**
- C2: Not C or A **violated**
- C3: Not B or C **satisfied**

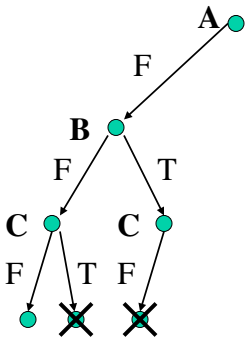


Propositional Satisfiability using Backtrack Search

- Assign true or false to an unassigned proposition.
- Backtrack as soon as a clause is violated.

Example:

- C1: Not A or B **satisfied**
- C2: Not C or A **satisfied**
- C3: Not B or C **violated**

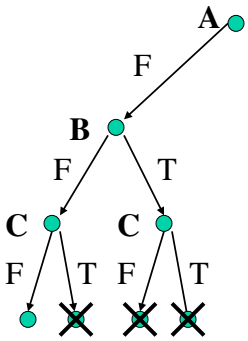


Propositional Satisfiability using Backtrack Search

- Assign true or false to an unassigned proposition.
- Backtrack as soon as a clause is violated.

Example:

- C1: Not A or B — satisfied
- C2: Not C or A — violated
- C3: Not B or C — satisfied

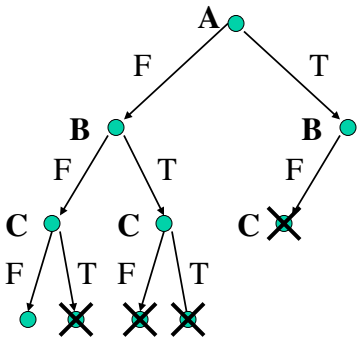


Propositional Satisfiability using Backtrack Search

- Assign true or false to an unassigned proposition.
- Backtrack as soon as a clause is violated.

Example:

- C1: Not A or B — violated
- C2: Not C or A — satisfied
- C3: Not B or C — satisfied

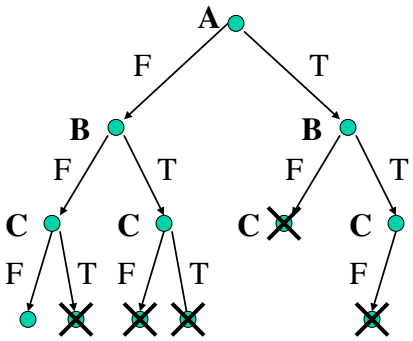


Propositional Satisfiability using Backtrack Search

- Assign true or false to an unassigned proposition.
- Backtrack as soon as a clause is violated.

Example:

- C1: ~~Not~~ A or B **satisfied**
- C2: Not C or A **satisfied**
- C3: ~~Not~~ B or ~~C~~ **violated**

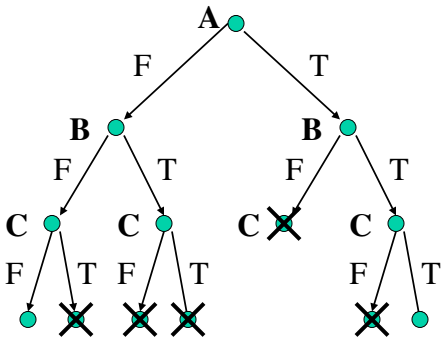


Propositional Satisfiability using Backtrack Search

- Assign true or false to an unassigned proposition.
- Backtrack as soon as a clause is violated.

Example:

- C1: ~~Not~~ A or B **satisfied**
- C2: Not C or A **satisfied**
- C3: ~~Not~~ B or C **satisfied**



Clausal Backtrack Search: Recursive Formulation

Procedure: BT(Φ, A)
Input: A *cnf* theory Φ,
An assignment A to some propositions in Φ.
Output: true if Φ is satisfiable; false otherwise.

If a clause in Φ is violated, Return false;
Else If all propositions in Φ are assigned by A, Return true;
Else Q = some proposition in Φ unassigned by A;
Return (BT(Φ, A[Q = True]) or
BT(Φ, A[Q = False]))

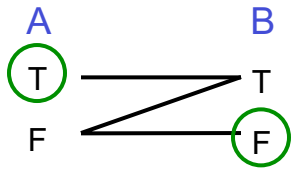
Outline

- Propositional Logic
- Propositional Satisfiability
  - Backtrack Search
  - Unit Propagation
  - DPLL: Unit Propagation + Backtrack Search
- Empirical, Average Case Analysis
- Appendices

Unit Clause Resolution

Idea: Apply arc consistency (AC-3) to binary clauses

Clause: (not A or B)



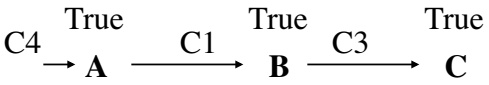
Unit clause resolution (aka unit propagation rule):
If all literals are false save L, then assign true to L:

- $\frac{\text{(not A)} \quad \text{(not B)} \quad \text{(A or B or C)}}{\text{C}}$
- Unit propagation = repeated application of rule.

Unit Propagation Examples

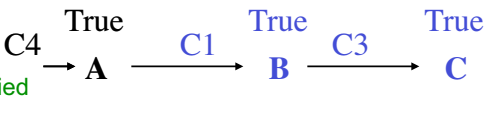
- C1: Not A or B Satisfied
- C2: Not C or A Satisfied
- C3: Not B or C Satisfied
- C4: A Satisfied

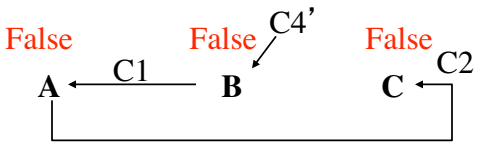
Support



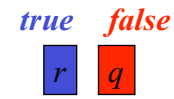
Unit Propagation Examples

- C1: Not A or B — Satisfied
- C2: Not C or A — Satisfied
- C3: Not B or C — Satisfied
- C4: A
- C4': Not B — Satisfied





Unit Propagation



$C_1 : \neg r \vee q \vee p$

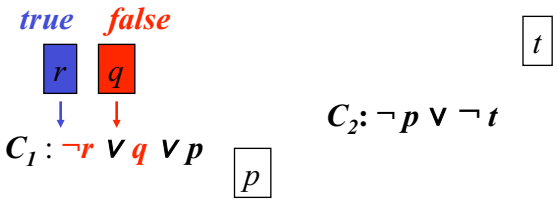
p

t

$C_2 : \neg p \vee \neg t$

Procedure: *propagate*(**C**) // **C** is a clause
  if all literals in **C** are false except **L**, and **L** is unassigned
  then assign true to **L** and
    record **C** as a support for **L** and
    for each clause **C'** mentioning "not **L**",
      propagate(**C'**)
end *propagate*

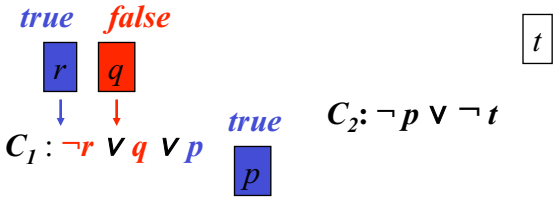
Unit Propagation



Procedure: *propagate*(**C**) // **C** is a clause

➡ **if** all literals in **C** are false except **L**, and **L** is unassigned
then assign true to **L** and
record **C** as a support for **L** and
for each clause **C′** mentioning "not **L**",
propagate(**C′**)
end *propagate*

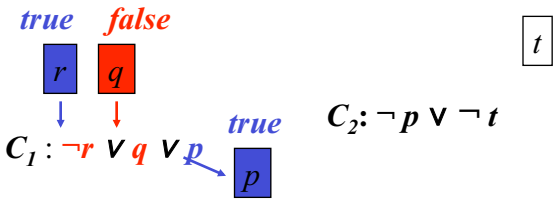
Unit Propagation



Procedure: *propagate*(**C**) // **C** is a clause
if all literals in **C** are false except **L**, and **L** is unassigned

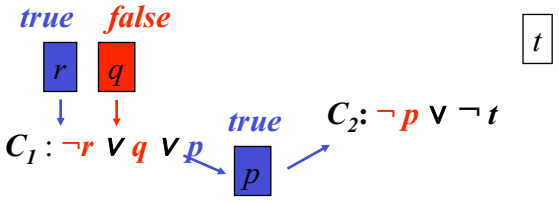
➡ **then** assign true to **L** and
record **C** as a support for **L** and
for each clause **C′** mentioning "not **L**",
propagate(**C′**)
end *propagate*

Unit Propagation



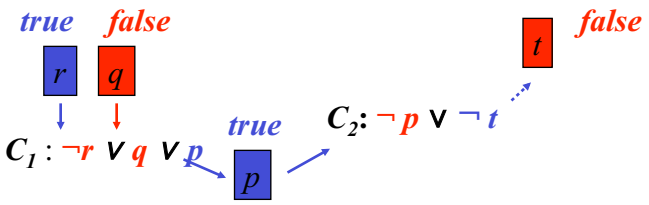
Procedure: *propagate*(**C**) // **C** is a clause
  if all literals in **C** are false except **L**, and **L** is unassigned
  then assign true to **L** and
⇨ record **C** as a support for **L** and
  for each clause **C′** mentioning "not **L**",
    propagate(**C′**)
end *propagate*

Unit Propagation



Procedure: *propagate*(**C**) // **C** is a clause
  if all literals in **C** are false except **L**, and **L** is unassigned
  then assign true to **L** and
  record **C** as a support for **L** and
⇨ **for each** clause **C′** mentioning "not **L**",
    propagate(**C′**)
end *propagate*

Unit Propagation



Procedure: *propagate*(**C**) // **C** is a clause
  if all literals in **C** are false except **L**, and **L** is unassigned
  then assign true to **L** and
    record **C** as a support for **L** and
    for each clause **C′** mentioning "not **L**",
⇨ propagate(**C′**)
end *propagate*

Outline

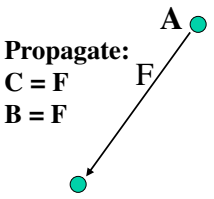
- Propositional Logic
- Propositional Satisfiability
  - Backtrack Search
  - Unit Propagation
  - DPLL: Unit Propagation + Backtrack Search
- Empirical, Average Case Analysis
- Appendices

Propositional Satisfiability using DPLL
[Davis, Logmann, Loveland, 1962]

Initially:
- Unit propagate.

Repeat:
1. Assign true or false to an unassigned proposition.
2. Unit propagate.
3. Backtrack as soon as a clause is violated.
4. Satisfiable if assignment is complete.

Example:
- C1: Not A or B **satisfied**
- C2: Not C or A **satisfied**
- C3: Not B or C **satisfied**

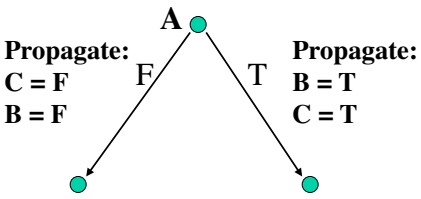


Propositional Satisfiability using DPLL
[Davis, Logmann, Loveland, 1962]

Initially:
- Unit propagate.

Repeat:
1. Assign true or false to an unassigned proposition.
2. Unit propagate.
3. Backtrack as soon as a clause is violated.
4. Satisfiable if assignment is complete.

Example:
- C1: Not A or B **satisfied**
- C2: Not C or A **satisfied**
- C3: Not B or C **satisfied**



How Do We Fold Unit Propagation into Backtracking?

Procedure: BT(Φ, A)
Input: A *cnf* theory Φ,
An assignment A to some propositions in Φ
Output: A decision of whether Φ is satisfiable.

If a clause in Φ is violated, Return false;
Else If all propositions of Φ are assigned in A, Return true;
Else Q = some unassigned proposition in Φ;
Return (BT(Φ, A[Q = True]) or
BT(Φ, A[Q = False]))

Hint: Like MAC and Forward Checking:

- limited inference
- apply inference after assigning each variable.

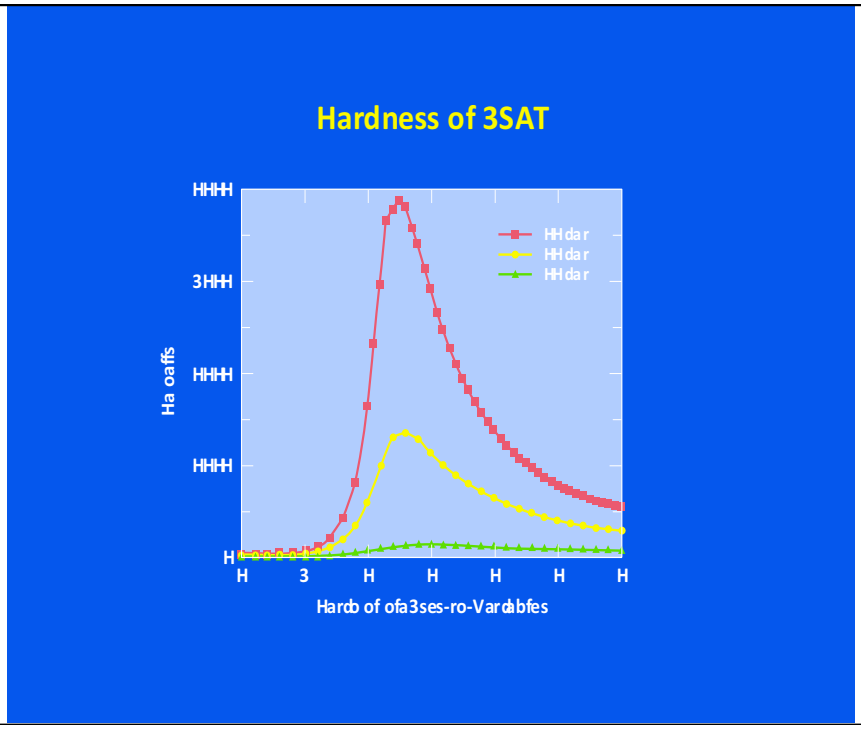
D(P)LL Procedure
[Davis, Logmann, Loveland, 1961]

Procedure: DPLL(Φ, A)
Input: A *cnf* theory Φ,
An assignment A to propositions in Φ
Output: A decision of whether Φ is satisfiable.

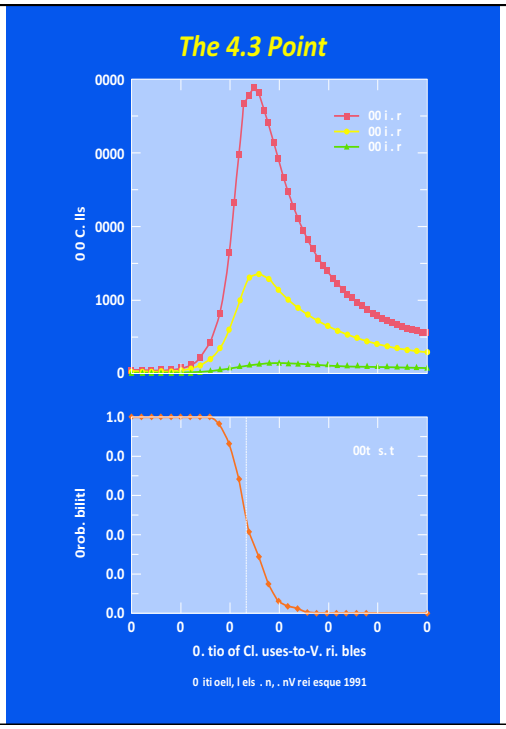
A' = propagate(Φ);
If a clause in Φ is violated, given A' Return false;
Else If all propositions of Φ are assigned in A', Return true;
Else Q = some unassigned proposition in Φ;
Return (DPLL(Φ, A' [Q = True]) or
DPLL(Φ, A' [Q = False]))

Outline

- Propositional Logic
- Propositional Satisfiability
  - Backtrack Search
  - Unit Propagation
  - DPLL: Unit Propagation + Backtrack Search
- Empirical, Average Case Analysis
- Appendices



Courtesy of Bart Selman. Used with permission.



Courtesy of Bart Selman. Used with permission.

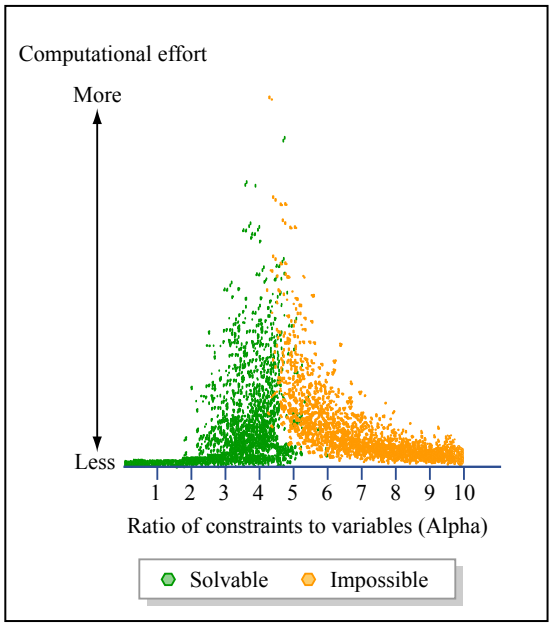


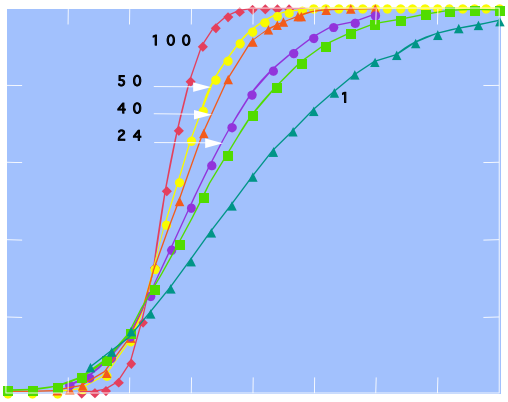
Image by MIT OpenCourseWare.

Brian Williams, Fall 10

Intuition

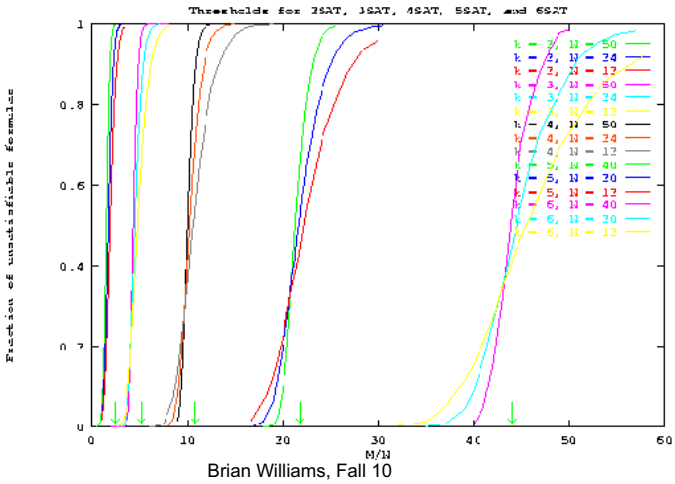
- At low ratios:
  - few clauses (constraints)
  - many assignments
  - easily found
- At high ratios:
  - many clauses
  - inconsistencies easily detected

Phase Transitions for Different Numbers of Variables



Courtesy of Bart Selman. Used with permission.

Phase Transitions: 2, 3 4, 5 and 6-SAT



Brian Williams, Fall 10

75

Courtesy of Bart Selman. Used with permission.

Required Appendices

You are responsible for reading and knowing this material:

1. **Local Search using Min_Conflict and GSAT**
2. **Reduction to Clausal Form**

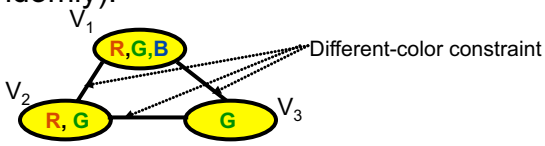
Incremental Repair (Min-Conflict Heuristic)

Spike Hubble Telescope Scheduler [Minton et al.]

1. Initialize a candidate solution using "greedy" heuristic – get solution "near" correct one.
2. Repeat until consistent:
   1. Select a variable in a conflict (violated constraint)
   2. assign it a value that minimizes the number of conflicts (break ties randomly).

Graph Coloring

Initial Domains



GSAT

- C1: Not A or B
- C2: Not C or Not A
- C3: or B or Not C

1. Init: Pick random assignment
2. Check effect of flipping each assignment, by counting violated clauses.
3. Pick assignment with fewest violations,
4. End if consistent, Else goto 2

C1, C2, C3 violated

True	False	True
A	**B**	**C**
False	True	False
C3 violated	C2 violated	C1 violated

GSAT

1. Init: Pick random assignment
2. Check effect of flipping each assignment, counting violated clauses.
3. Pick assignment with fewest violations,
4. End if consistent, Else goto 2

- C1: Not A or B
- C2: Not C or Not A
- C3: or B or Not C

C1 violated

True	False	False
A	**B**	**C**
False	True	True
Satisfied	Satisfied	C1,C2,C3 violated

GSAT

1. Init: Pick random assignment
2. Check effect of flipping each assignment, counting violated clauses.
3. Pick assignment with fewest violations,
4. End if consistent, Else goto 2

- C1: Not A or B
- C2: Not C or Not A
- C3: or B or Not C

Satisfied

True	True	False
A	**B**	**C**

Problem: Pure hill climbers get stuck in local minima.

Solution: Add random moves to get out of minima (WalkSAT)

Required Appendices

You are responsible for reading and knowing this material:

1. **Local Search using Min_Conflict and GSAT**
2. **Reduction to Clausal Form**

Reduction to Clausal Form: Engine Example

(mode(E1) = ok implies
  (thrust(E1) = on iff flow(V1) = on and flow(V2) = on)) and
(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown)

not (mode(E1) = ok) or not (thrust(E1) = on) or flow(V1) = on;
not (mode(E1) = ok) or not (thrust(E1) = on) or flow(V2) = on;
not (mode(E1) = ok) or not (flow(V1) = on) or not (flow(V2) = on) or
  thrust(E1) = on;
mode(E1) = ok or mode(E1) = unknown;
not (mode(E1) = ok) or not (mode(E1) = unknown);

Reducing Propositional Formula to Clauses (CNF)

1) Eliminate IFF and Implies:

- E1 iff E2 => (E1 implies E2) and (E2 implies E1)
- E1 implies E2 => not E1 or E2

Eliminate IFF: Engine Example

(mode(E1) = ok implies
(thrust(E1) = on iff (flow(V1) = on and flow(V2) = on))) and
(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown)

(mode(E1) = ok implies
((thrust(E1) = on implies (flow(V1) = on and flow(V2) = on)) and
((flow(V1) = on and flow(V2) = on) implies thrust(E1) = on))) and
(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown)

Eliminate Implies: Engine Example

(mode(E1) = ok implies
  ((thrust(E1) = on implies (flow(V1) = on and flow(V2) = on)) and
   ((flow(V1) = on and flow(V2) = on) implies thrust(E1) = on))) and
(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown)

(not (mode(E1) = ok) or
  ((not (thrust(E1) = on) or (flow(V1) = on and flow(V2) = on)) and
   (not (flow(V1) = on and flow(V2) = on)) or thrust(E1) = on))) and
(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown)

Reducing Propositional Formula to Clauses (CNF)

2) Move negations in towards propositions using De Morgan's Theorem:

- not (E1 and E2) => (not E1) or (not E2)
- not (E1 or E2) => (not E1) and (not E2)
- not (not E1) => E1

Move Negations In: Engine Example

(not (mode(E1) = ok) or
  ((not (thrust(E1) = on) or (flow(V1) = on and flow(V2) = on)) and
   (**not (flow(V1) = on and flow(V2) = on)**) or thrust(E1) = on))) and
(mode(E1) = ok or mode(E1) = unknown) and
not (mode(E1) = ok and mode(E1) = unknown)

(not (mode(E1) = ok) or
  ((not (thrust(E1) = on) or (flow(V1) = on and flow(V2) = on)) and
   (not (flow(V1) = on) or not (flow(V2) = on)) or thrust(E1) = on)) and
(mode(E1) = ok or mode(E1) = unknown) and
(**not (mode(E1) = ok) or not (mode(E1) = unknown)**))

Reducing Propositional Formula to Clauses (CNF)

3) Move conjunctions out using distributivity:

- E1 or (E2 and E3) => (E1 or E2) and (E1 or E3)

Move Conjunctions Out: Engine Example

(not (mode(E1) = ok) or
  ((not (thrust(E1) = on) or (flow(V1) = on and flow(V2) = on)) and
   (not (flow(V1) = on) or not (flow(V2) = on) or thrust(E1) = on))) and
(mode(E1) = ok or mode(E1) = unknown) and
(not (mode(E1) = ok) or not (mode(E1) = unknown))

(not (mode(E1) = ok) or
  (((not (thrust(E1) = on) or flow(V1) = on) and
    (not (thrust(E1) = on) or flow(V2) = on)) and
   (not (flow(V1) = on) or not (flow(V2) = on) or thrust(E1) = on))) and
(mode(E1) = ok or mode(E1) = unknown) and
(not (mode(E1) = ok) or not (mode(E1) = unknown))

Move Conjunctions Out: Engine Example

(not (mode(E1) = ok) or
  (((not (thrust(E1) = on) or flow(V1) = on) and
    (not (thrust(E1) = on) or flow(V2) = on)) and
   (not (flow(V1) = on) or not (flow(V2) = on) or thrust(E1) = on))) and
(mode(E1) = ok or mode(E1) = unknown) and
(not (mode(E1) = ok) or not (mode(E1) = unknown))

(not (mode(E1) = ok) or not (thrust(E1) = on) or flow(V1) = on) and
(not (mode(E1) = ok) or not (thrust(E1) = on) or flow(V2) = on)) and
(not (mode(E1) = ok) or not (flow(V1) = on) or not (flow(V2) = on)
  or thrust(E1) = on) and
(mode(E1) = ok or mode(E1) = unknown) and
(not (mode(E1) = ok) or not (mode(E1) = unknown))

Reducing Propositional Formula to Clauses (CNF)

4) "Simplify by Equivalence"

remove double negations
- (not not E1) => E1

apply commutativity and associativity
- (E1 or (E3 or (not E1))) => (E1 or (not E1) or E3)

remove duplicate literals
- (E1 or E1) => E1

remove duplicate clauses
- (E1 or (not E2)) and (E1 or (not E2)) => (E1 or (not E2))

reduce by tautology
- (E1 or … or (not E1)) => true

definition of and/or
- true and E1 => E1
- false and E1 => false
- (true or E1) => true
- (false or E1) => E1

Reducing Propositional Formula to Clauses (CNF)

1) Eliminate IFF and Implies
- E1 iff E2 => (E1 implies E2) and (E2 implies E1)
- E1 implies E2 => not E1 or E2

2) Move negations in towards propositions using De Morgan's Theorem:
- not (E1 and E2) => (not E1) or (not E2)
- not (E1 or E2) => (not E1) and (not E2)
- not (not E1) => E1

3) Move conjunctions out using Distributivity
- E1 or (E2 and E3) => (E1 or E2) and (E1 or E3)

4) Simplify by Equivalence

16.410 / 16.413 Principles of Autonomy and Decision Making

Fall 2010