CS2002 Logic Lecture 4

Proving things in Logic
DPLL Algorithm

Ian Gent

Last Time

- Truth Tables
- More on Formalisation
- Boolean Algebra (not quite finished)
- (We didn’t have time for …) A murder mystery…
- (We didn’t have time for …) How To Prove Things in Logic
-

This Time

- A Murder Mystery
- How to prove things in Logic
- Reductio ad Absurdum
- DPLL Algorithm
  - Conjunctive Normal Form (CNF)
  - The algorithm
- Solving the whodunnit!
  - Figuring out the murderer

Larger Example of Formalisation

- More complicated logical problem
- Encode in Logic
- Look at later stages next time

Example Problem

There are three suspects for a murder: Adams, Brown, and Clark. Adams says "I didn't do it. The victim was an old acquaintance of Brown's. But Clark hated him." Brown states "I didn't do it. I didn't know the guy. Besides I was out of town all the week." Clark says "I didn't do it. I saw both Adams and Brown downtown with the victim that day; one of them must have done it." Assume that the two innocent men are telling the truth, but that the guilty man might not be. Write out the facts as sentences in Propositional Logic, and use propositional reasoning to solve the crime.

Formalisation (1)

Variables
A: **A**dams is guilty
B: **B**rown is guilty
C: **C**lark is guilty
K: Brown **K**new the victim
H: Clark **H**ated the victim
O: Brown was **O**ut of town

Formalisation (2)

Adams Says: $\neg A \land K \land H$
Brown Says: $\neg B \land \neg K \land O$
Clark Says: $\neg C \land K \land \neg O \land (A \lor B)$

But … one of them may be lying …

Formalisation (3)

Guilty man *may* be lying
I.e. non guilty people telling truth:

$\neg A \rightarrow (\neg A \wedge K \wedge H)$

$\neg B \rightarrow (\neg B \wedge \neg K \wedge O)$

$\neg C \rightarrow (\neg C \wedge K \wedge \neg O \wedge (A \vee B))$

Formalisation (4)

One of them is guilty

$A \vee B \vee C$

Two of them are innocent

I.e. *Only* one is guilty

$\neg(A \wedge B)$

$\neg(A \wedge C)$

$\neg(B \wedge C)$

Complete set of statements

$\neg A \rightarrow (\neg A \wedge K \wedge H)$

$\neg B \rightarrow (\neg B \wedge \neg K \wedge O)$

$\neg C \rightarrow (\neg C \wedge K \wedge \neg O \wedge (A \vee B))$

$A \vee B \vee C$

$\neg(A \wedge B)$

$\neg(A \wedge C)$

$\neg(B \wedge C)$

How to PROVE Things in Logic

- Key word is “prove”
- I.e. give a guarantee that something is correct
- This is typically the point of converting to logic
- E.g. Hardware verification
  - We want to KNOW that the hardware computes what we think it does
  - Given the assumptions we make hold true
  - E.g. we might assume cosmic rays don’t affect how gates work

How to Prove Things in Logic

- We want absolute certainty
- Which is of course hard to get in general
- But Logic is well suited to providing
- Remember we even talked about some terms for certainty
  - “valid” and ”tautology” give certainty of truth
  - “unsatisfiable” gives certainty of falsity

Reminder: Some key terms

- A logic formula can be one or more of these
  - **Valid**
    - Every way of assigning the variables T/F makes the formula true
  - **Tautology**
    - Exactly the same as valid
  - **Invalid**
    - Not valid
    - There is at least one assignment which makes the formula untrue
  - **Satisfiable**
    - At least one assignment to T/F makes formula true
  - **Unsatisfiable**
    - No assignment can make the formula true

A Valid Argument

- “Valid” has another meaning as well as synonym for tautology
- An argument is VALID if the conclusion necessarily follows from the assumptions
- In terms of Propositional Logic
  - If we have a set of assumptions A1, A2, A3, …
    - Each a formula of propositional logic, maybe very complex
  - And we have a conclusion C
    - Again possibly complex
  - We have a valid argument that the conclusion follows from the assumptions iff it is impossible for all of A1, A2, A3, … to be true and C to be false

Valid Argument = Valid Formula

- We have a valid argument that the conclusion follows from the assumptions

  iff it is impossible for all of A1, A2, A3, … to be true and C to be false
- Which is true iff the following formula is valid (a tautology)

  $(A1 \wedge A2 \wedge A3 \wedge \ldots) \rightarrow C$

- Note the tight link between the meanings of valid

Reductio ad absurdum in general

- "Reduction to an absurdity"
    - Also called "proof by contradiction"
- If the argument is valid, then $(A1 \land A2 \land A3 \land \dots) \rightarrow C$ is a tautology
- The only way $(A1 \land A2 \land A3 \land \dots) \rightarrow C$ can be false is if all the A's are true and C is false
- If this is impossible then $A1 \land A2 \land A3 \land \dots \land \neg C$ is unsatisfiable
- If we deduce falsity (an absurdity) then this must be unsatisfiable
- So one way to prove an argument valid is to prove falsity from the assumptions and the negation of the conclusion
- We'll see this in DPLL later

How to PROVE Things in Logic

- Key word is “prove”
- I.e. give a guarantee that something is correct
- This is typically the point of converting to logic
- E.g. Hardware verification
  - We want to KNOW that the hardware computes what we think it does
  - Given the assumptions we make hold true
  - E.g. we might assume cosmic rays don’t affect how gates work

How to Prove Things in Logic

- We want absolute certainty
- Which is of course hard to get in general
- But Logic is well suited to providing
- Remember we even talked about some terms for certainty
    - “valid” and ”tautology” give certainty of truth
    - “unsatisfiable” gives certainty of falsity

Reminder: Some key terms

- A logic formula can be one or more of these
  - **Valid**
    - Every way of assigning the variables T/F makes the formula true
  - **Tautology**
    - Exactly the same as valid
  - **Invalid**
    - Not valid
    - There is at least one assignment which makes the formula untrue
  - **Satisfiable**
    - At least one assignment to T/F makes formula true
  - **Unsatisfiable**
    - No assignment can make the formula true

A Valid Argument

- “Valid” has another meaning as well as synonym for tautology
- An argument is VALID if the conclusion necessarily follows from the assumptions
- In terms of Propositional Logic
  - If we have a set of assumptions A1, A2, A3, …
    - Each a formula of propositional logic, maybe very complex
  - And we have a conclusion C
    - Again possibly complex
  - We have a valid argument that the conclusion follows from the assumptions iff it is impossible for all of A1, A2, A3, … to be true and C to be false

Valid Argument = Valid Formula

- We have a valid argument that the conclusion follows from the assumptions

  iff it is impossible for all of A1, A2, A3, … to be true and C to be false
- Which is true iff the following formula is valid (a tautology)

  $(A1 \land A2 \land A3 \land \ldots) \rightarrow C$

- Note the tight link between the meanings of valid

Reductio ad absurdum in general

- “Reduction to an absurdity”
    - Also called “proof by contradiction”
- If the argument is valid, then $(A1 \land A2 \land A3 \land \dots) \rightarrow C$ is a tautology
- The only way $(A1 \land A2 \land A3 \land \dots) \rightarrow C$ can be false is if all the A’s are true and C is false
- If this is impossible then $A1 \land A2 \land A3 \land \dots \land \neg C$ is unsatisfiable
- If we deduce falsity (an absurdity) then this must be unsatisfiable
- So one way to prove an argument valid is to prove falsity from the assumptions and the negation of the conclusion
- We’ll see this in DPLL later

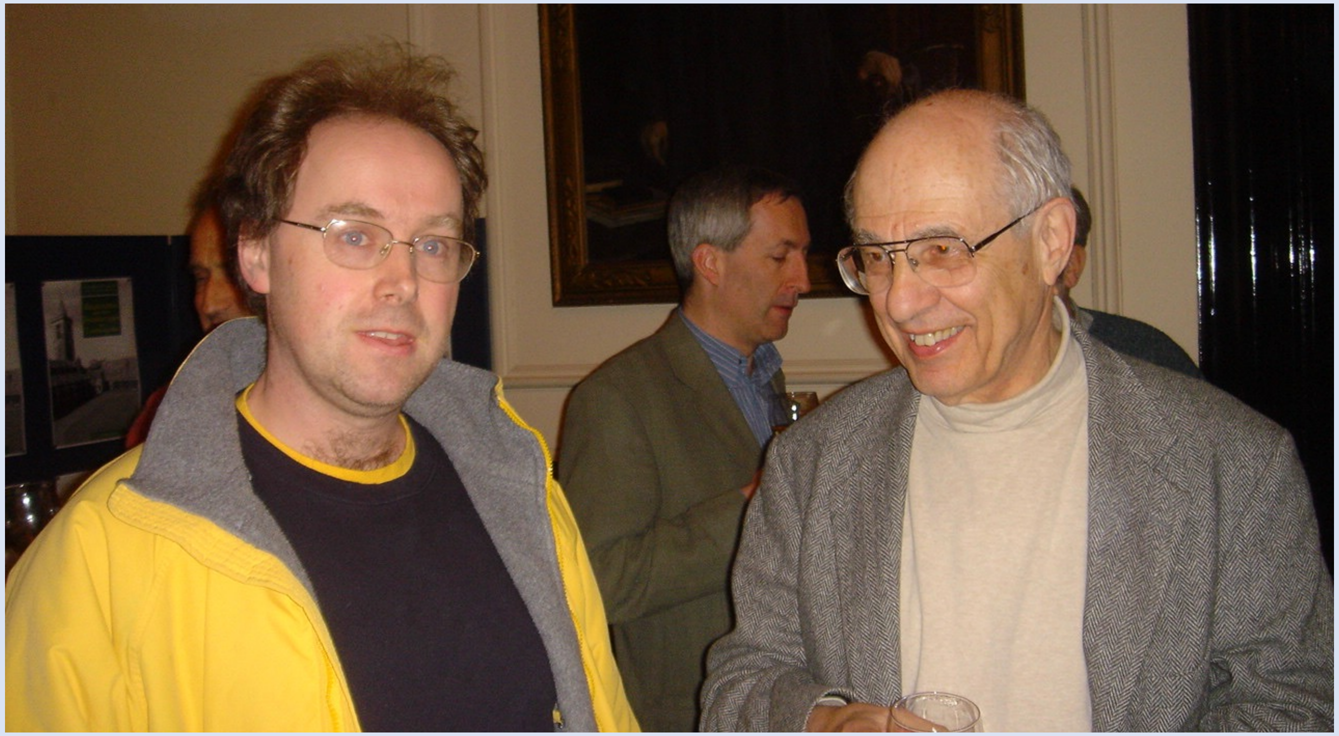
The DPLL Algorithm

DPLL Algorithm

- A key Computer Logic algorithm
- Now more than 60 years old
- Named for Four Computer Scientists
- Martin Davis, Hilary Putnam, George Logemann, Donald Loveland

- I'm happy to have met two of them
    - Hilary Putnam, 1926-2016
        - https://en.wikipedia.org/wiki/Hilary_Putnam
    - Donald Loveland, 1934-
        - https://en.wikipedia.org/wiki/Donald_W._Loveland

Ian Gent and Hilary Putnam



Putnam gave a talk in St Andrews, Feb 2005.

Conjunctive Normal Form

- DPLL works on formulas in “Conjunctive Normal Form”
- We first convert other propositional logic formulas into it.

Conjunctive Normal Form

- A formula in CNF is the conjunction ($\wedge$) of clauses
- A clause is the disjunction ($\vee$) of literals
- A literal is an atom or a negated atom
- Any propositional formula has an equivalent CNF but it may be much bigger
- Order of clauses and literals does not matter, nor do repeats
- A clause containing 1 or both p and ¬p can be omitted
- 0 can be omitted from a clause
- A CNF including an empty clause is unsatisfiable (always false)
- An empty CNF is a tautology (always true)

Empty CoNFusion?

- An empty *clause* is **false**
- An empty *set of clauses* is **true**
- **Understandably confusing!**
  - Think of the difference between having *nothing* and an *empty bag*
- **Try to remember the difference**
  - As you add literals to a clause it gets *easier* to make true
    - (or b or c or d…)
    - So the empty clause is hardest (false)
  - As you add clauses to a set, they get *harder* to make true
    - (and … and … and..)
    - So the empty set is easiest (true)

De Morgan’s Laws

- Named after Augustus De Morgan
- Swap negations inside/outside brackets and swap $\land$ and $\lor$
- The following two formulas are equivalent to each other

$$\neg (A \land B)$$
$$\neg A \lor \neg B$$

- And so are the following two formulas

$$\neg (A \lor B)$$
$$\neg A \land \neg B$$

- Exercise: convince yourself this is correct (e.g. using truth tables)

Distributivity Laws

- We can “distribute” $\land$ inside $\lor$ and vice versa
- The following two formulas are equivalent to each other

$$A \lor (B \land C)$$

$$(A \lor B) \land (A \lor C)$$

- And so are the following two formulas

$$A \land (B \lor C)$$

$$(A \land B) \lor (A \land C)$$

- Exercise: convince yourself this is correct (e.g. using truth tables)

Converting to CNF

Stage	Type of Formula	Next Stage	Rules
1	Any Boolean Formula	Eliminate →, ↔ , ⊕	D→E becomes ¬D∨E D↔E becomes (D∧E)∨(¬D∧¬E) D⊕E becomes (D∧¬E)∨(¬D∧E)
2	Atoms, ∧, ∨, ¬, 1, 0	Move Negation inwards	¬(E∧G) becomes ¬E∨¬G ¬(E∨G) becomes ¬E∧¬G ¬¬E becomes E
3	atoms, negated atoms, ∧, ∨, 1, 0	distribute ∨ over ∧	E∨(G∧H) becomes (E∨G)∧(E∨H)
4	Conjunction of Disjunction of atoms, negated atoms, 1 and 0	Clean up 1s and 0s	Zero and Unit Laws 0 and 1 complement Laws
5	Conjunction of Disjunction of atoms, negated atoms	**Conjunctive Normal Form (CNF)**	

Example reduction to CNF

$\neg(a \rightarrow (b \vee c)) \wedge (b \rightarrow (a \wedge c))$

$\neg(\neg a \vee (b \vee c)) \wedge (\neg b \vee (a \wedge c))$	{replace implies}
$(\neg\neg a \wedge \neg(b \vee c)) \wedge (\neg b \vee (a \wedge c))$	{De Morgan}
$(a \wedge \neg(b \vee c)) \wedge (\neg b \vee (a \wedge c))$	{double negation}
$(a \wedge (\neg b \wedge \neg c)) \wedge (\neg b \vee (a \wedge c))$	{De Morgan}
$(a \wedge (\neg b \wedge \neg c)) \wedge ((\neg b \vee a) \wedge (\neg b \vee c))$	{Distribution}

$a \wedge \neg b \wedge \neg c \wedge (\neg b \vee a) \wedge (\neg b \vee c)$

Example Ctd

Five clauses

a∧¬b∧¬c ∧(¬b∨a)∧ (¬b∨c)

Five **clauses**

Three with one literal

Two with two

Often written without the ∧

And with comma for ∨

a

¬b

¬c

¬b,a

¬b,c

- Convenient standard form for formulae
  - easy to represent on computer
  - basis for various algorithms to decide satisfiability e.g. DPLL

Unit Propagation

(¬a∨c∨d)∧(b∨¬c)∧(¬d∨b)∧(¬b)

- There's a *unit clause* above, ¬b
- To satisfy the clause set we MUST set b false
- That clause is satisfied and can be deleted
  - As could any other clauses involving ¬b
- And b is guaranteed unsatisfied wherever it appears
  - So we can delete it from clauses it appears in
  - (¬a∨c∨d)∧(¬c)∧(¬d)
- Now we have two unit clauses ¬c and ¬d
  - Simplify again and we will just have
  - (¬a)

DPLL Algorithm

- Decides satisfiability of propositional formulas
  - I.e. if there is a *satisfying assignment*
    - *DPLL will return one*
  - If there is no satisfying assignment
    - *DPLL will return failure*
- DPLL exponential in principle but practical for quite large problems
- Descendants of this algorithm used today with problems with *millions* of clauses
- We will give a simplified version of the classic algorithm.

Literals

- A literal is either a positive or negative occurrence of a variable in a clause
  - E.g. in $\neg a \vee \neg b \vee c$ the literals are $\neg a$, $\neg b$, c
- If we set variable a to true then literal $\neg a$ is false
- If we set variable a to false then literal $\neg a$ is true

DPLL Algorithm

Basic idea is simple but works very well

If we have a clause with one literal, that literal **must** be true

- so assign it and resimplify
- **Unit propagation**

Otherwise pick any variable

- first try assuming it’s true, then assuming it’s false
- In each case, simplify clauses, and then pick another variable if necessary

DPLL Simplification

- We assign variables to True or False
  - E.g. we have a clause (-a,-b,c) (shorthand for $\neg a \vee \neg b \vee c$)
  - We assign a = True we have (False,-b,c)
  - Which we can simplify to (-b,c)
  - We assign -b = True we have (True,c)
  - We can discard this clause completely as it is always true
- In general: Whenever we assign a value to a literal:
  - Discard clauses containing True
  - Delete False from clauses
  - Fail if we get an empty clause

DPLL Algorithm

```
DPLL(S)
    IF S empty THEN return TRUE
    ELSE IF S contains empty clause return FALSE
    ELSE IF S contains unit clause <x>
        return DPLL(Simplify-literal(x))
    ELSE
        Pick any literal x in S [e.g. First lit in first clause]
        IF DPLL(Simplify-literal(x)) THEN return TRUE
        ELSE return DPLL(Simplify-literal(-x))
```

Extended Example

- More complicated logical problem
- Encode in English
- Reduce to CNF
- Apply DPLL
- Find out who *could* have done the murder
- Decide if we can *prove* they did the murder

(From L3) Example Problem

There are three suspects for a murder: Adams, Brown, and Clark. Adams says "I didn't do it. The victim was an old acquaintance of Brown's. But Clark hated him." Brown states "I didn't do it. I didn't know the guy. Besides I was out of town all the week." Clark says "I didn't do it. I saw both Adams and Brown downtown with the victim that day; one of them must have done it." Assume that the two innocent men are telling the truth, but that the guilty man might not be. Write out the facts as sentences in Propositional Logic, and use propositional reasoning to solve the crime.

(From L3) Formalisation (1)

Variables
A: **A**dams is guilty
B: **B**rown is guilty
C: **C**lark is guilty
K: Brown **K**new the victim
H: Clark **H**ated the victim
O: Brown was **O**ut of town

(From L3) Formalisation (2)

Adams Says: $\neg A \wedge K \wedge H$
Brown Says: $\neg B \wedge \neg K \wedge O$
Clark Says: $\neg C \wedge K \wedge \neg O \wedge (A \vee B)$

But … one of them may be lying …

(From L3) Formalisation (3)

Guilty man *may* be lying
I.e. non guilty people telling truth:

$\neg A \rightarrow (\neg A \wedge K \wedge H)$
$\neg B \rightarrow (\neg B \wedge \neg K \wedge O)$
$\neg C \rightarrow (\neg C \wedge K \wedge \neg O \wedge (A \vee B))$

(From L3) Formalisation (4)

One of them is guilty

$A \lor B \lor C$

Two of them are innocent

I.e. *Only* one is guilty

$\neg(A \land B)$

$\neg(A \land C)$

$\neg(B \land C)$

(From L3) Complete set of statements

$\neg A \rightarrow (\neg A \wedge K \wedge H)$

$\neg B \rightarrow (\neg B \wedge \neg K \wedge O)$

$\neg C \rightarrow (\neg C \wedge K \wedge \neg O \wedge (A \vee B))$

$A \vee B \vee C$

$\neg(A \wedge B)$

$\neg(A \wedge C)$

$\neg(B \wedge C)$

Convert to CNF

One of them is guilty A∨B∨C, At most one is guilty ¬(A∧B), ¬(A∧C), ¬(B∧C)	**(a,b,c)** **(-a,-b)** **(-a,-c)** **(-b,-c)**	Clauses in () - for ¬ , for ∨
Guilty man may be lying: ¬A→ (¬A∧K∧H) ¬b→ (¬B∧¬K∧O) ¬C→ (¬C∧K∧¬O∧(A∨B))	**(a,k)** **(a,h)** **(b,-k)** **(b,o)** **(c,k)** **(c,-o)**	Omit tautologies e.g. (-a,a) Omit duplicates e.g. (c,a,b)

DPLL for our Example 1

Clauses	Notes	Assigned Clauses	Simplified Clauses	More Notes
(a,b,c)	We don’t have	(a,b,c)	(a,b,c)	Remember
(-a,-b)	any unit	(-a,-b)	(-a,-b)	
(-a,-c)	clauses.	(-a,-c)	(-a,-c)	If this doesn’t
(-b,-c)		(-b,-c)	(-b,-c)	work we have
(a,k)	So we pick any	(a,False)	(a)	to try k true
(a,h)	literal to make	(a,h)	(a,h)	later
(b,-k)	true	(b,True)		
(b,o)		(b,o)	(b,o)	
(c,k)	Try –k True	(c,False)	(c)	
(c,-o)		(c,-o)	(c,-o)	
	Same as			
	making k False			

DPLL for our Example 2

Clauses	**Notes**	**Assigned Clauses**	**Simplified Clauses**	**More Notes**
(a,b,c)	We have unit	(True,b,True)		We have the
(-a,-b)	clauses a and c	(False,-b)	(-b)	Empty clause
(-a,-c)		(False,False)	**()**	**()**
(-b,-c)	For speed let’s	(-b,False)	(-b)	
(a)	set them both	(True)		Remember
(a,h)	True at once	(True,h)		this is false.
(b,o)		(b,o)	(b,o)	We have failed
(c)		(True)		and have to
(c,-o)		(True,-o)		backtrack
				Try k true

DPLL for our Example 3

Clauses	**Notes**	**Assigned Clauses**	**Simplified Clauses**	**More Notes**
(a,b,c)	We tried –k	(a,b,c)	(a,b,c)	We have got
(-a,-b)	True and it	(-a,-b)	(-a,-b)	the unit clause
(-a,-c)	failed	(-a,-c)	(-a,-c)	(b)
(-b,-c)		(-b,-c)	(-b,-c)	
(a,k)	Now we try –k	(a,True)		So Brown did
(a,h)	False	(a,h)	(a,h)	it!
(b,-k)		(b,False)	(b)	
(b,o)	i.e. k True	(b,o)	(b,o)	But … it may
(c,k)		(c,True)		turn out there
(c,-o)	If this fails	(c,-o)	(c,-o)	is no solution
	there is no			
	solution			So let’s
				continue

DPLL for our Example 4

Clauses	Notes	Assigned Clauses	Simplified Clauses	More Notes
(a,b,c) (-a,-b) (-a,-c) (-b,-c) (a,h) (b) (b,o) (c,-o)	Assign b true because it is a unit clause	(a,True,c) (-a,False) (-a,-c) (False,-c) (a,h) (True) (True,o) (c,-o)	 (-a) (-a,-c) (-c) (a,h) (c,-o)	We have got the unit clauses (-a) (-c) I.e. Adams and Clark are innocent And they are telling the truth

DPLL for our Example 5

Clauses	Notes	Assigned Clauses	Simplified Clauses	More Notes
(-a)	Assign –a true and –c true	(True)		Just left with two unit clauses.
(-a,-c)		(True, True)		
(-c)	Same as a false and c false	(True)		
(a,h)		(False,h)	(h)	
(c,-o)		(False,-o)	(-o)	

DPLL for our Example 6

Clauses	Notes	Assigned Clauses	Simplified Clauses	More Notes
	Assign h true and –o true Same o false			Empty set of clauses so we have a solution:
(h)		(True)		a false b true c false h true
(-o)		(True)		k true o false

So one solution is…

- a false
- b true
- c false
- h true
- k true
- o false

- Brown was guilty (b),
- Brown knew the victim (k),
- Brown was not out of town (-o)
- Clark hated the victim (h)

- Q: How do we know if Brown is the only possible murderer?

Reductio ad absurdum in DPLL

"Reduction to an absurdity"

If we deduce false then the original clause set must be unsatisfiable

Remember the empty clause is false

So (hold your breath)…

- Add the opposite of what you want to prove
- If you deduce the empty clause then the opposite is unsatisfiable
- So what you wanted to prove must be true

In this example let's see if we can prove b [i.e Brown is guilty]

- by adding the clause **(-b)**
- seeing if we get the empty clause using DPLL

Reductio ad absurdum in DPLL

In this example let's see if we can prove b [i.e Brown is guilty]

by adding the clause **(-b)**

seeing if we get the empty clause using DPLL

SPOILER: Yes this will lead to a contradiction.

Exercise: do this (you should be able to get the empty clause)

Exercise: can you see any other way we could have deduced this was the only possible solution?

Next time

- Digital Logic
- Sequential Circuits