CS2002 Logic Lecture 5

DPLL Algorithm
Digital Logic

Ian Gent

Last Time

- A Murder Mystery
- How to prove things in Logic
- Reductio ad Absurdum
- DPLL Algorithm
  - Conjunctive Normal Form (CNF)
  - (we didn’t have time for…) The algorithm
- Solving the whodunnit!
  - (we didn’t have time for…) Figuring out the murderer

This Time

- DPLL Algorithm
  - (reminder) Conjunctive Normal Form (CNF)
  - The algorithm
- Solving the whodunnit!
  - Figuring out the murderer
- Digital Logic
  - Basics
  - Gates and simple circuits
  - Combinational Circuits

The DPLL Algorithm

(Reminder) 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

(Reminder) Conjunctive Normal Form

- A formula in CNF is the conjunction (∧) of clauses
- A clause is the disjunction (∨) 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)

(Reminder) Converting to CNF

Stage	Type of Formula	Next Stage	Rules
1	Any Boolean Formula	Eliminate $\rightarrow, \leftrightarrow, \oplus$	$D\rightarrow E$ becomes $\neg D \vee E$ $D\leftrightarrow E$ becomes $(D\wedge E)\vee(\neg D\wedge\neg E)$ $D\oplus E$ becomes $(D\wedge\neg E)\vee(\neg D\wedge E)$
2	Atoms, $\wedge$, $\vee$, $\neg$, 1, 0	Move Negation inwards	$\neg(E\wedge G)$ becomes $\neg E\vee\neg G$ $\neg(E\vee G)$ becomes $\neg E\wedge\neg G$ $\neg\neg E$ becomes E
3	atoms, negated atoms, $\wedge$, $\vee$, 1, 0	distribute $\vee$ over $\wedge$	$E\vee(G\wedge H)$ becomes $(E\vee G)\wedge(E\vee H)$
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

a∧¬b∧¬c ∧(¬b∨a)∧ (¬b∨c)

Five **clauses**

Three with one literal

called “**unit clauses**”

Two with two

Often written without the ∧

And with comma for ∨

Five clauses

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

$(\neg a \lor c \lor d) \land (b \lor \neg c) \land (\neg d \lor b) \land (\neg b)$

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

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 ¬a∨¬b∨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))
```

(Reminder) 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.

(Reminder) 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

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

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)	Assign b true because it is a unit clause	(a,True,c)		We have got the unit clauses (-a) (-c)
(-a,-b)		(-a,False)	(-a)	
(-a,-c)		(-a,-c)	(-a,-c)	
(-b,-c)		(False,-c)	(-c)	
(a,h)		(a,h)	(a,h)	I.e. Adams and Clark are innocent
(b)		(True)		
(b,o)		(True,o)		
(c,-o)		(c,-o)	(c,-o)	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?

Digit

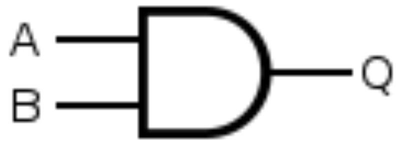
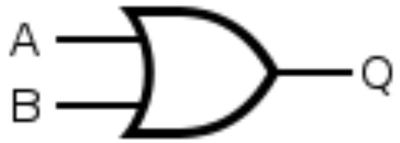
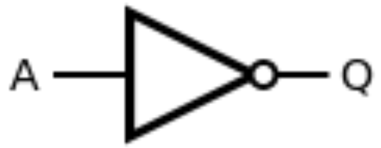
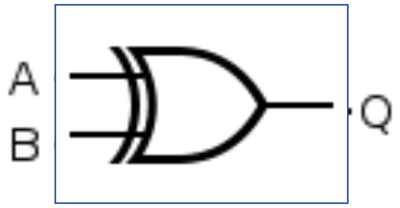
Logic Gates

- Digital circuits (e.g., that perform arithmetic operations or make choices in a computer) are constructed from a number of primitive elements called logic gates
- A gate is a small electronic device that computes various functions of two-valued signals (1 or 0, true or false)
- A gate implements a boolean expression

Logic Gates

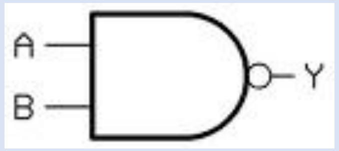
- A gate has one or more inputs and one or more outputs
- The gates are connected to each other by links
- Each link carries a bit of information 1 or 0, true or false
- Each gate combines its inputs according to some rule and sends the result to its outputs
- AND, OR, XOR gates -- two inputs, one output
- NOT gate -- one input one output.

Four Key Gates

	AND Gate Q = A ∧ B	Two or more inputs One output
	OR Gate Q = A∨B	Two or more inputs One output
	NOT Gate Q = ¬A	One Input One Output
	XOR Gate Q = A ⊕ B	Two or more inputs One output

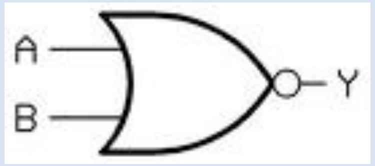
Two More Key Gates

NAND:
NOT AND



A	B	A ∧ B	A nand B A ↑ B
F	F	F	**T**
F	T	F	**T**
T	F	F	**T**
T	T	T	**F**

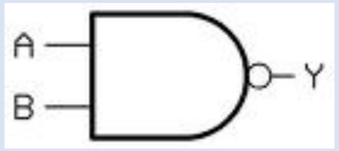
NOR:
NOT OR



A	B	A ∨ B	A nor B A ↓ B
F	F	F	**T**
F	T	T	**F**
T	F	T	**F**
T	T	T	**F**

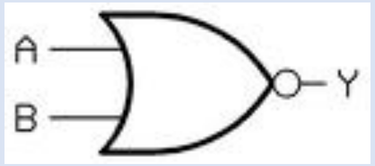
Two **Universal** Gates

NAND:
NOT AND



A	B	A ∧ B	A nand B A ↑ B
F	F	F	**T**
F	T	F	**T**
T	F	F	**T**
T	T	T	**F**

NOR:
NOT OR



A	B	A ∨ B	A nor B A ↓ B
F	F	F	**T**
F	T	T	**F**
T	F	T	**F**
T	T	T	**F**

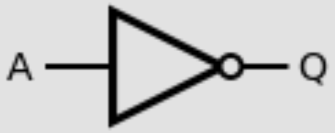
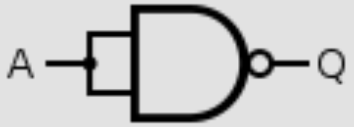
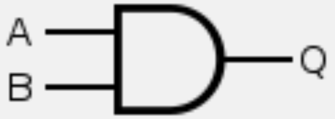
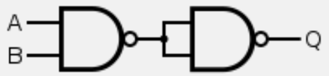
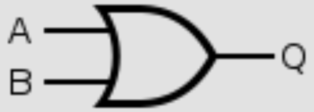
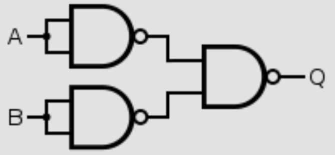
Why Universal?

- Why did I call NAND and NOR Universal?
  - NAND is universal
  - NOR is universal
  - (Each one on its own)

- A truth function is universal if
  - we can express **ANY** truth function using only that function

- We can do this for NAND
- To show this let's show we can express NOT, AND, OR
- Which we know can express all propositional logic

Completeness of NAND

Desired Formula	Desired Gate	NAND Gate Equivalent	NAND Formula Equivalent
Q = ¬A			Q = (A↑A)
Q = A ∧ B			Q = (A↑B) ↑ (A↑B)
Q = AVB			Q = (A↑A) ↑ (B↑B)

Images from Wikipedia

Exercises:

- Show that XOR can be expressed in NANDs
- Show that NOR is complete

Common Operations to implement in Chips

- Moving data from one part of the machine to another
- Selecting data from one of several sources
- Routing data from a source to one of several destinations
- Comparing data arithmetically with other data
- Manipulating data arithmetically or logically, for example, summing two binary numbers

Combinational vs Sequential Circuits

- There are two kinds of digital circuits: “combinational” and “sequential”
- In a combinational circuit the output is always based entirely on the inputs, i.e. in some way on the combination of inputs
- A circuit can have several outputs, each of them described by a Boolean expression
- A combinational circuit has no internal storage capability or “memory”
- Later we’ll see sequential circuits in which output depends on what happened before, i.e. on the sequence of inputs

Adding Binary Numbers in Hardware

Half adder

- adding two binary digits A and B
- Their sum is going to be TWO bits
  - call these the “sum bit” and “carry bit”
- e.g. 1+1 = 10 (binary)
  - sum bit 0 and carry bit 1
- e.g. 0+1 = 01 (binary)
  - sum bit 1 and carry bit 0

Half adder

- Step 1
- Form truth table to get functionality we want
- Step 2
- Figure out how to do this with logic gates we have
- In this case ...
- Notice sum = XOR, carry = AND

inputs		outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half adder

- Step 3
- Build circuit with right logic
- In this case
- XOR, AND and wire splits
- A/B inputs
- S = sum output
- C = carry output

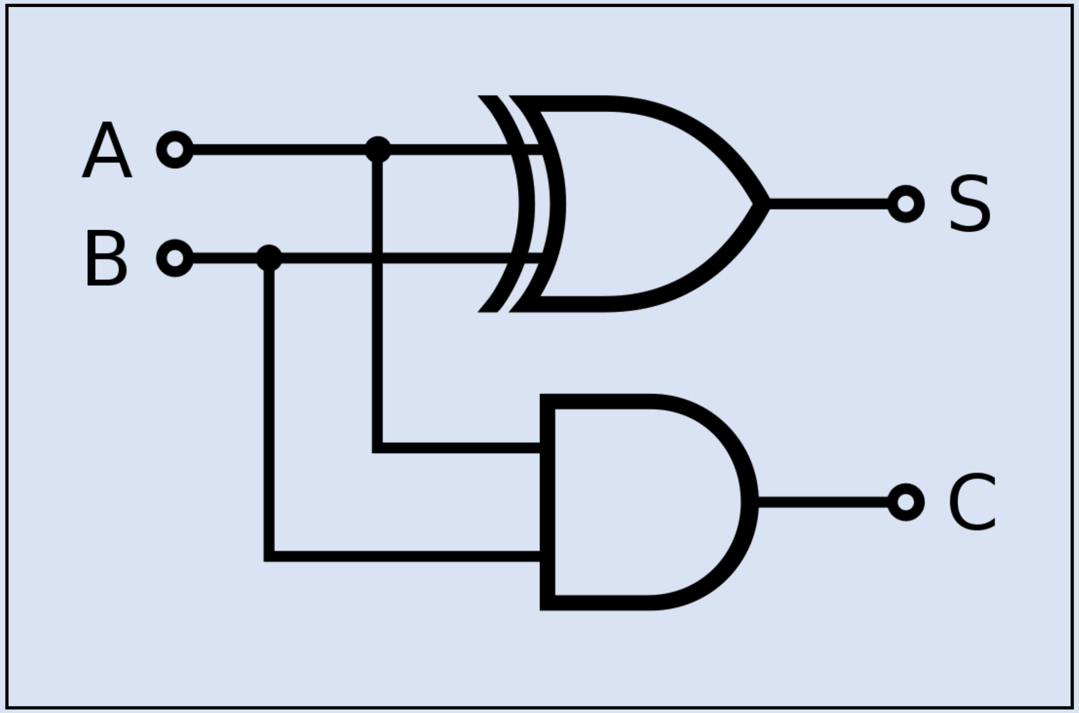


image: wikipedia

Next time

- More Combinational Circuits
  - More addition in hardware
    - Full Adder for 1 bit
    - Ripple adder for n bits
  - Non-arithmetic logic functions in hardware