Propositional Logic

Louis Oliphant
oliphant@cs.wisc.edu
cs540 section 2

Logic Defined

(by dictionary.com)

- Logic - The study of the principles of reasoning, especially of the structure of propositions as distinguished from their content and of method and validity in deductive reasoning.
- Several types of logic:
  - propositional logic (boolean logic)
  - first order logic (first order predicate calculus)

Important Concepts

- Structure of Propositions
  - syntax : what is a correctly formed string
  - semantics : What do correctly formed strings "mean"
- Principles of Reasoning
  - entailment : Given some knowledge(α) what logically follows(β)

Socrates is a man
All men are mortal
Socrates is mortal

$$\alpha \models \beta$$

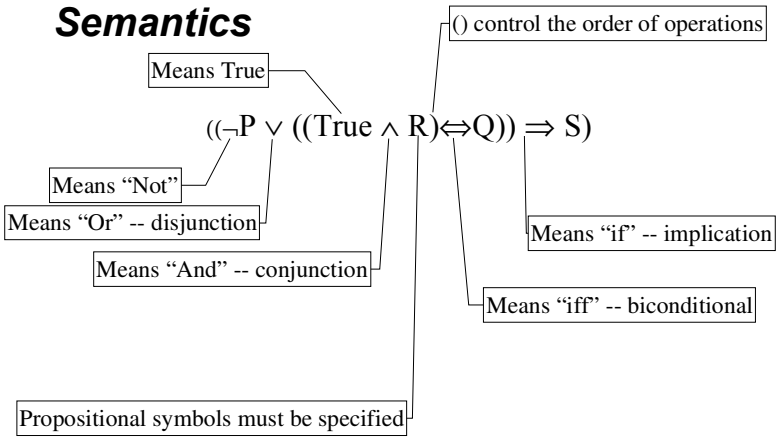
Syntax

Sentence	$\rightarrow$	*AtomicSentence* | *ComplexSentence*
AtomicSentence	$\rightarrow$	True | False | *Symbol*
Symbol	$\rightarrow$	P | Q | R | . . .
ComplexSentence	$\rightarrow$	$\neg$*Sentence*
	|	(*Sentence* $\wedge$ *Sentence*)
	|	(*Sentence* $\vee$ *Sentence*)
	|	(*Sentence* $\Rightarrow$ *Sentence*)
	|	(*Sentence* $\Leftrightarrow$ *Sentence*)

BNF grammar in propositional logic

$((\neg P \vee ((\text{True} \wedge R) \Leftrightarrow Q)) \Rightarrow S)$ well formed
$(\neg(P \vee Q) \wedge \Rightarrow S)$ not well formed

Semantics



Semantics—Order of Operations

- You can leave off parenthesis if the order of operations is clear
- Order precedence is (from highest to lowest):
  ¬, ∧, ∨, ⇒, ⇔

¬P ∨ True ∧ R⇔Q ⇒ S ok
P ⇒ Q ⇒ S not ok

Semantics – Propositional Symbols

- Propositional Symbols must be specified by a model

Calculated using the meaning of connectives

Model: specified directly

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Figuring out the meaning of Sentences

¬P ∨ Q∧ R ⇒ Q

P	Q	R	~P	Q^R	~PvQ^R	~PvQ^R->Q
0	0	0	1	0	1	0
0	0	1	1	0	1	0
0	1	0	1	0	1	1
0	1	1	1	1	1	1
1	0	0	0	0	0	1
1	0	1	0	0	0	1
1	1	0	0	0	0	1
1	1	1	0	1	1	1

n=number of symbols

Satisfiable: Sentences that are true under some model
Note:This is an NP-hard problem $O(2^n)$

Figuring out the meaning of Sentences

$$(P \wedge R \Rightarrow Q) \wedge P \wedge R \wedge \neg Q$$

P	Q	R	~Q	R^~Q	P^R^~Q	P^R	P^R->Q	final
0	0	0	1	0	0	0	1	0
0	0	1	1	1	0	0	1	0
0	1	0	0	0	0	0	1	0
0	1	1	0	0	0	0	1	0
1	0	0	1	0	0	0	1	0
1	0	1	1	1	1	1	0	0
1	1	0	0	0	0	0	1	0
1	1	1	0	0	0	1	1	0

unsatisfiable: Sentences that are false under all models

Figuring out the meaning of Sentences

$$(P \Rightarrow Q) \vee P \wedge \neg Q$$

P	Q	R	~Q	P->Q	P^~Q	(P->Q)vP^~Q
0	0	0	1	1	0	1
0	0	1	1	1	0	1
0	1	0	0	1	0	1
0	1	1	0	1	0	1
1	0	0	1	0	1	1
1	0	1	1	0	1	1
1	1	0	0	1	0	1
1	1	1	0	1	0	1

valid: Sentences that are true under all models (tautology)

Figuring out the meaning of Sentences

$$(P \Rightarrow Q) \vee P \wedge \neg Q$$

You Figure It Out

Logical Equivalences

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge$$
$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee$$
$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge$$
$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee$$
$$\neg(\neg\alpha) \equiv \alpha \quad \text{double-negation elimination}$$
$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition}$$
$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \quad \text{implication elimination}$$
$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{de Morgan}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{de Morgan}$$
$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee$$
$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge$$

Know and use these equivalences to modify sentences

Inference

- The act or process of deriving logical conclusions from premises known or assumed to be true.
(dictionary.com)

$$\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$$ Modus Ponens $$\frac{\alpha \wedge \beta}{\alpha}$$ And-Elimination

All Logical Equivalences can be used as inference mechanism

Proof

- Series of inference steps that lead from an initial state to the goal state (one method of proving)

given a: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
given b: $\neg B_{1,1}$

Knowledge Base (KB)

goal: $\neg P_{1,2}$ ← Query

Proof

- Series of inference steps that lead from an initial state to the goal state (one method of proving)

given a: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$	
given b: $\neg B_{1,1}$	
1) $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$	biconditional elimination to (a)
2) $((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$	and-elimination to (1)
3) $\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1})$	contraposition to (2)
4) $\neg(P_{1,2} \vee P_{2,1})$	modus ponens to (3 and b)
5) $\neg P_{1,2} \wedge \neg P_{2,1}$)	de Morgan's rule to (4)
goal: $\neg P_{1,2}$	and-elimination to (5)

How would a computer go about this? Can we simplify inference and still find the goal?

Maintaining Accuracy

- Sound – inference algorithm that derives only "true" sentences
- Complete – inference algorithm that can derive any "true" sentence

Resolution is all you need

- Resolution is another inference rule
- It alone is sound and complete*
  (but you have to get your sentences in the right form)
  (luckily all sentences can be converted to this form)

* -- refutation complete – it can answer any question you ask it, but it can't list out all things that are true.

Conjunctive Normal Form (CNF)

Here's how you convert any sentence to CNF:

1. Replace all $\Leftrightarrow$ using biconditional elimination
2. Replace all $\Rightarrow$ using implication elimination
3. Move all negations inward using
   -double-negation elimination
   -de Morgan's rule
4. Apply distributivity of $\vee$ over $\wedge$

Let's Try It!

Converting to CNF

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$	starting sentence
$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$	biconditional elimination
$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$	implication elimination
$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$	move negations inward
$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$	distribute $\vee$ over $\wedge$

clauses

This is in Conjunctive Normal Form
(now why did we do this again?)
To use resolution

Resolution

- Remember Resolution is *refutation* complete. You have to ask a question (*query*) of the given knowledge base.
- The *query* needs to be in CNF form
- Resolution works using proof by contradiction

KB ← knoweldge base
α ← query
KB $\models \alpha$ ← does the knowledge base entail α
show KB$\wedge\neg\alpha$ leads to a contradiction

Proof

- Series of inference steps that lead from an initial state to the goal state (one method of proving)

given a: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
given b: $\neg B_{1,1}$

goal: $\neg P_{1,2}$

Proof

- Series of resolution inference steps that lead from an initial state($KB \wedge \neg\alpha$) to the False state (proof by contradiction)

given a: $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$
given b: $\neg B_{1,1}$
given c: $P_{1,2}$

in CNF form

Write this as three seperate clauses

goal: $\square$ ← False

Proof

- Series of resolution inference steps that lead from an initial state($KB \wedge \neg\alpha$) to the False state (proof by contradiction)

given a1: $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1})$
given a2: $(\neg P_{1,2} \vee B_{1,1})$
given a3: $(\neg P_{2,1} \vee B_{1,1})$
given b: $\neg B_{1,1}$
given c: $P_{1,2}$

goal: $\square$

Setup Complete.

(Now all you need to know is resolution to fill in the missing lines.)

Resolution *(finally)*

- Take any two clauses that have a symbol and its complement (negative)

  $P \vee Q \vee R \qquad \neg Q \vee S \vee T$

- Combine the two clauses into one clause, omitting the symbol and its complement

  $P \vee R \vee S \vee T$

- If two clauses resolve and there are no symbols left then you have reached $\square$, False.

Proof

- Series of resolution inference steps that lead from an initial state($KB \wedge \neg\alpha$) to the False state (proof by contradiction)

Can we turn this into an algorithm a computer can do?

What if the KB doesn't entail α?

given a1: $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1})$
given a2: $(\neg P_{1,2} \vee B_{1,1})$
given a3: $(\neg P_{2,1} \vee B_{1,1})$
given b: $\neg B_{1,1}$
given c: $P_{1,2}$

1. $\neg P_{1,2}$ — resolved a2 with b

goal: $\square$ — resolved 1 with c

Resolution Algorithm

function resolution(KB,a) returns true or false
inputs: KB, the knowledge base
a, the query

clauses ← convert_to_CNF(KB ∧ ¬α)
new ← {}
loop do
  for each C_i, C_j **in** *clauses* **do**
    resolvents ← resolve(C_i, C_j)
    if *resolvents* contains the empty clause **then return** *true*
    new ← *new* ∪ *resolvents*
  if *new* ⊆ *clauses* **then return** *false*
  clauses ← *clauses* ∪ *new*

Evaluating Algorithm

- Run Time
  - In worst case is still exponential
  - Often much faster
- factoring: If a clause contains two of the same symbol, just delete one of them (keeps search tree finite)
  P∨R∨P∨T gets changed to P∨R∨T
- No need to resolve clauses that result in a symbol and its complement (these clauses will never help)

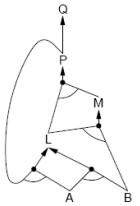
Horn Clauses

- Disjunction of literals with *at most one positive*
  - Not as general as full propositional logic
  - Still can represent many real world problems
- Reasoning is linear in the size of input
- Uses Forward chaining or Backward chaining algorithm to reason
  - Only can ask atomic queries

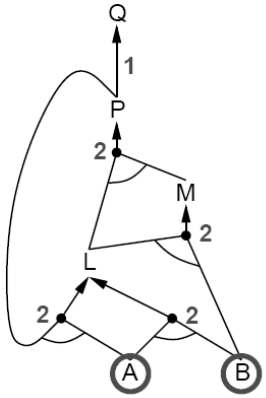
Forward Chaining

- Fire any rule whose premises are satisfied in the KB.
- Add its conclusion to the KB until query is found.

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

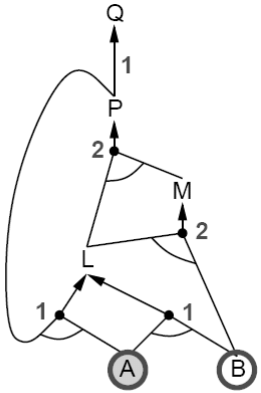


Forward Chaining Example



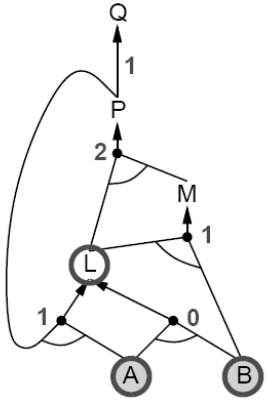
P ⇒ Q
L ∧ M ⇒ P
B ∧ L ⇒ M
A ∧ P ⇒ L
A ∧ B ⇒ L
A
B

Forward Chaining Example



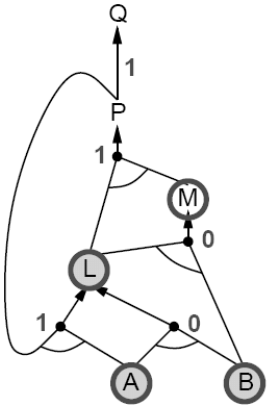
P ⇒ Q
L ∧ M ⇒ P
B ∧ L ⇒ M
A ∧ P ⇒ L
A ∧ B ⇒ L
A
B

Forward Chaining Example



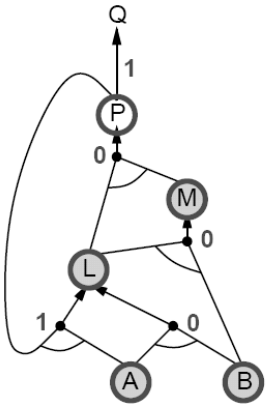
P ⇒ Q
L ∧ M ⇒ P
B ∧ L ⇒ M
A ∧ P ⇒ L
A ∧ B ⇒ L
A
B

Forward Chaining Example



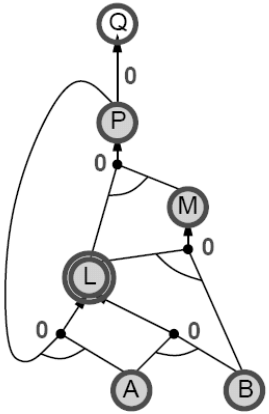
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Forward Chaining Example



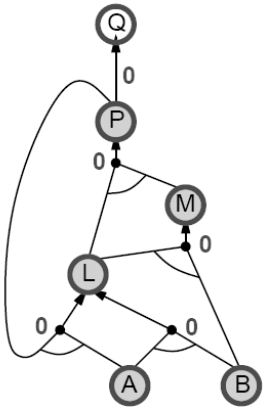
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Forward Chaining Example



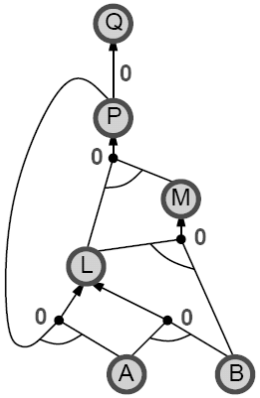
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Forward Chaining Example



$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Forward Chaining Example

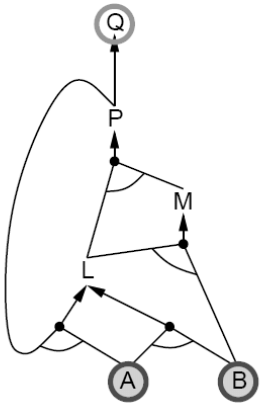


P ⇒ Q
L ∧ M ⇒ P
B ∧ L ⇒ M
A ∧ P ⇒ L
A ∧ B ⇒ L
A
B

Backward Chaining

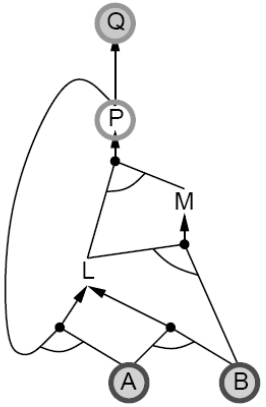
- Motivation: Need goal-directed reasoning in order to keep from getting overwhelmed with irrelevant consequences
- Main idea:
  - Work backwards from query q
  - To prove q:
    - Check if q is known already
    - Prove by backward chaining all premises of some rule concluding q

Backward Chaining Example



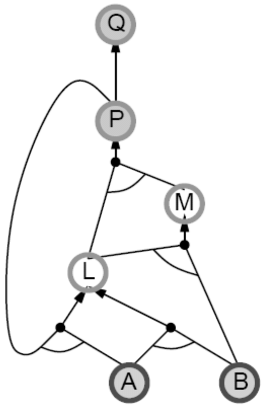
P ⇒ Q
L ∧ M ⇒ P
B ∧ L ⇒ M
A ∧ P ⇒ L
A ∧ B ⇒ L
A
B

Backward Chaining Example



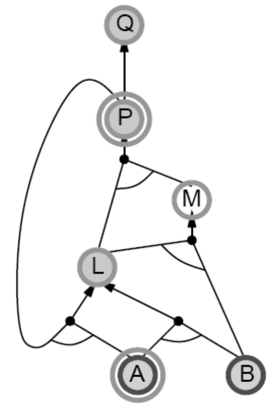
P ⇒ Q
L ∧ M ⇒ P
B ∧ L ⇒ M
A ∧ P ⇒ L
A ∧ B ⇒ L
A
B

Backward Chaining Example



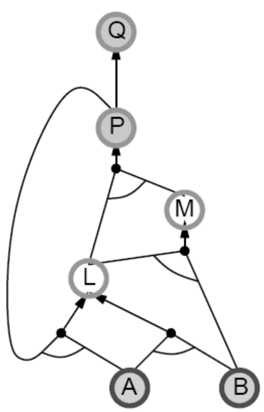
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Backward Chaining Example



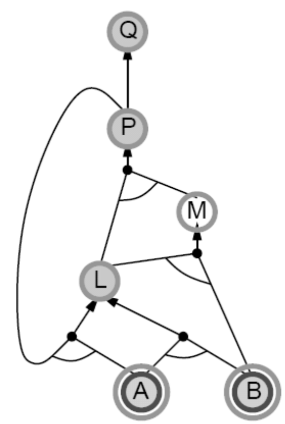
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Backward Chaining Example



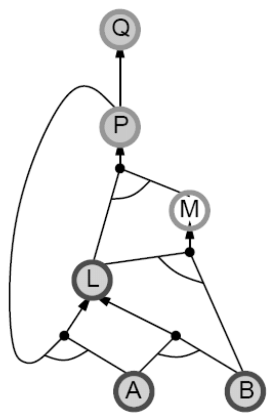
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Backward Chaining Example



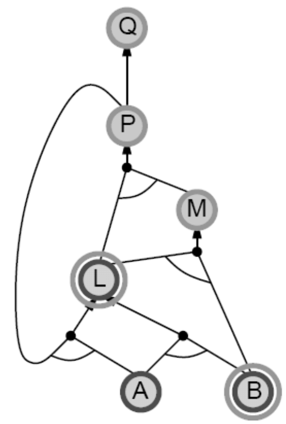
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Backward Chaining Example



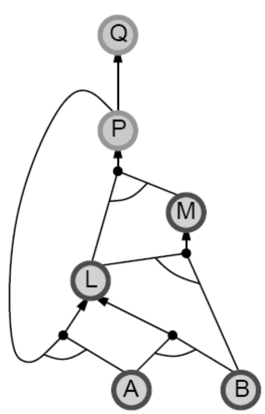
P ⇒ Q
L ∧ M ⇒ P
B ∧ L ⇒ M
A ∧ P ⇒ L
A ∧ B ⇒ L
A
B

Backward Chaining Example



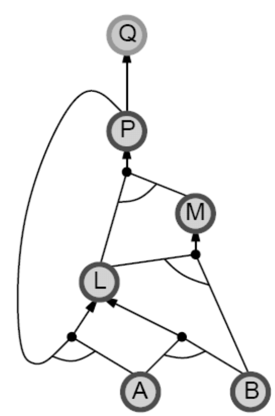
P ⇒ Q
L ∧ M ⇒ P
B ∧ L ⇒ M
A ∧ P ⇒ L
A ∧ B ⇒ L
A
B

Backward Chaining Example



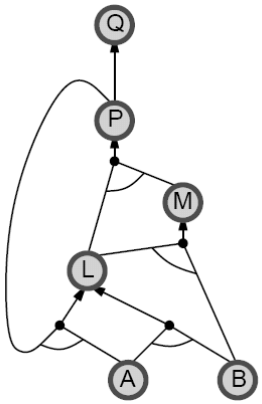
P ⇒ Q
L ∧ M ⇒ P
B ∧ L ⇒ M
A ∧ P ⇒ L
A ∧ B ⇒ L
A
B

Backward Chaining Example



P ⇒ Q
L ∧ M ⇒ P
B ∧ L ⇒ M
A ∧ P ⇒ L
A ∧ B ⇒ L
A
B

Backward Chaining Example



P ⇒ Q
L ∧ M ⇒ P
B ∧ L ⇒ M
A ∧ P ⇒ L
A ∧ B ⇒ L
A
B

Forward Chaining vs. Backward Chaining

- FC is data-driven—it may do lots of work irrelevant to the goal
- BC is goal-driven—appropriate for problem-solving

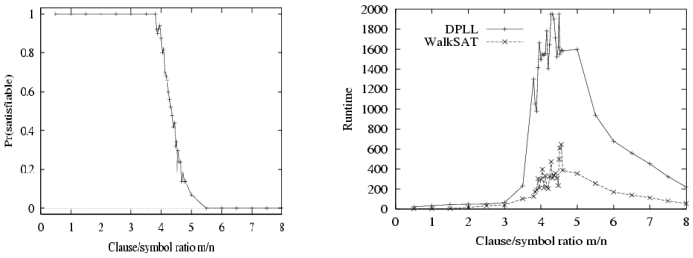
What You Should Know

- A ton of terms
- Truth tables
- Proofs
- Converting to Conjunctive Normal Form
- Proofs with Resolution (Proof by Contradiction)
- Horn Clauses
- Forward chaining algorithm
- Backward chaining algorithm

Proving Satisfiability

- DPLL (Davis-Putnam-Logemann, and Loveland)
  - Terminate search if you know it won't work
  - Fix symbols that only appear in one form
    - (all P or all ~P)
  - Fix symbols that appear alone in clauses
    - [P^(PvR)] -- Fix P to be true
- GSAT (local search)
  - Works with complete solutions
  - Flip variable that decreases number of unsatisfied clauses the most
- WalkSAT (local search)
  - Works with complete solutions
  - Picks clause that is not satisfied and flips a variable in clause (randomly or decreasing unsatisified clauses the most)

Comparing DPLL and WalkSAT (on hard problems)



Project Ideas

- Davis-Putnam with RRR, compare to DBLL, WalkSAT, and GSAT algorithms