

Propositional Logic

CSCE 420 – Fall 2023

read: Ch. 7

Knowledge-based programming

- As Feigenbaum said, one route to designing intelligent systems is to give them knowledge (expertise) to solve problems
- We need a general language for encoding knowledge.
- English is not sufficient (too ambiguous)
- The history of AI is tied to the search for higher-level languages that are “more expressive”.
 - AI drove advances in functional, object-oriented, and logic programming
 - LISP, Prolog, Smalltalk...

Logic

- Logic has become the standard for expressing knowledge in KBS
- Logic has advantages over procedural languages
 - context-independence: easier to judge correctness of a rule that 1 line buried in code (hence, easier to debug and maintain)
 - logic has a well-defined semantics (not subject to order, global variables, side-effects...)
 - rules can be used in many different ways (lots of different inferences)
- declarative vs. procedural programming: say “what”, not “how”
 - procedural languages require you to say HOW to do something
 - declarative languages let you describe the world, and the system can autonomously figure out the right thing to do as a consequence of the situation
- KBS: Knowledge-Based Systems
 - programming by writing “rule bases”

Example: Driving

- think about all the knowledge and inference you use while driving...
 - traffic laws (can you cross a yellow line?)
 - mechanics of vehicle operation
 - right of way, turn signal, yellow lights...
 - safety (speed, following distance , changing lanes, pedestrians)
 - slippery roads, fire trucks, school buses...
 - other drivers: do they see you? can you infer their intentions? are they displaying erratic behavior?
- it is better to put this all in a giant KB, rather than trying to program an enormous if-then-else to handle all possible situations

Inference Algorithms

- Of course, we need a way to extract conclusions and make decisions from a rule base
- synonyms: "inference", "automated deduction", "theorem-proving"
- Inference algorithms are a foundation for Expert Systems
- expert systems “shells” are the architecture/environment in which you:
 1. load your rule base
 2. describe current situation
 3. ask questions...or what to do...or whether something is a consequence...
 4. get an explanation of the information used to get the answer (i.e. “proof”)

Defining a “logic”

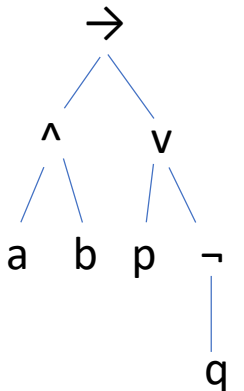
- There are actually many types of logics
 - propositional/Boolean logic
 - First-order logic (FOL), higher-order logics...
 - modal logics, epistemic logics (beliefs), temporal logics (used for program analysis)...
 - fuzzy logics, probabilistic logics...
 - non-sentential logics (like maps)
- These logics differ in *expressiveness* and *computational complexity*
 - First-order logic (FOL) is the *lingua franca* for most KR in AI
- Each of these logic has its own:
 1. **Syntax** – the rules defining what sentences are legal expressions
 2. **Semantics** – defines “truth” of sentences, and relationship of meaning between sentences
 3. **Proof theory** – a method for answering queries

Syntax of Propositional Logic

- well-defined sentences
- atomic sentences = propositional symbols (A, P, battery_low, lights_on_room124)
- complex sentences: generated using operators
 - binary ops: and (\wedge), or (\vee), xor (\oplus), implication (\rightarrow), biconditional (\leftrightarrow)
 - unary oper: negation (\neg)
 - parentheses

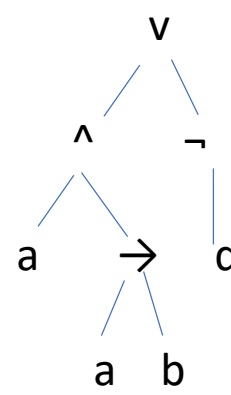
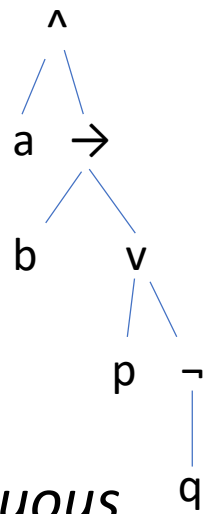
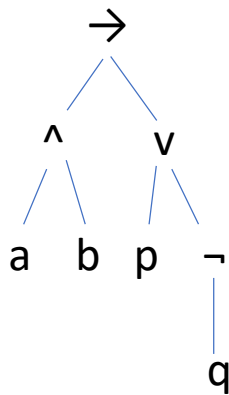
Syntax

- BNF grammar (Backus-Naur Form, production rules)
 - atomic ::= <prop> // tokens like “P”, “gas_tank_filled”...
 - binop ::= ^ | v | → | ↔ | ⊕
 - complex ::= <atomic> | ¬ complex | <complex> <binop> <complex> | (<complex>)
- examples:
 - legal: P, PvQ, ¬¬X^¬¬Y, ¬(¬X^(¬(¬Y))), Light ↔ Dark
 - not syntactically legal: "Win vv Lose", "(→Draw)"
- these can be used to derive the parse tree(s) for an expression
 - (I'm not going to give the algorithm for parsing this grammar here...)
 - a^b→pv¬q



Syntax

- of course, there are other possible parse trees...
 - $a \wedge b \rightarrow p \vee \neg q$



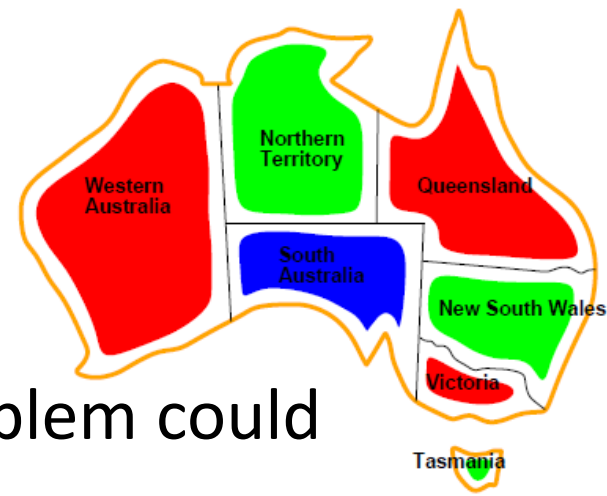
- the grammar is *ambiguous*
- one can always disambiguate an expression by adding parentheses
 - $(a \wedge b) \rightarrow (p \vee \neg q)$ vs $a \wedge (b \rightarrow p \vee \neg q)$ vs $a \wedge (b \rightarrow p) \vee \neg q$

Syntax

- ...or one can rely on rules of precedence among operators
 - \neg (highest)
 - \wedge
 - \vee, \oplus
 - $\rightarrow, \leftrightarrow$ (lowest)
- There can still be parsing ambiguity: $AvBvC = (AvB)vC$ or $Av(BvC)$?
- each operator is left-associative: $(AvB)vC$
- these syntax rules are similar to mathematics:
 - all operators are left associative, except \wedge (which is right associative)
 - $1+2*3/4/5+6^7^2 = (1+(((2*3)/4)/5))+6^7^2$
 - $1-2+3 = ?$ (2 or -4?)

- (unary minus) (highest)
\wedge (exponentiation)
*, /
+, -
=, <, > (lowest)

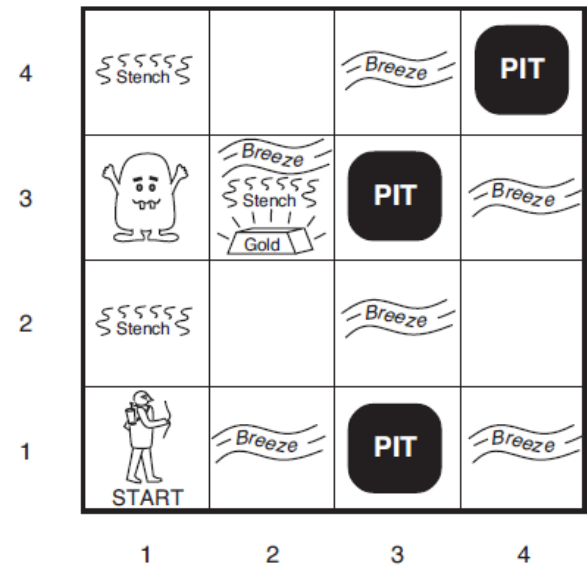
Example: Map-coloring



- A propositional encoding of the Australia map-color problem could look like this:
 - propositional symbols: WAR (Western Australia is Red), WAG, WAB, NTR, NTG, NTB (Northern Territories is Blue), QR, QG, QB... each can be True or False
 - KB:
 - $WAR \vee WAG \vee WAB, NTR \vee NTG \vee NTB, QR \vee QG \vee QB...$ // each state is 1 of 3 colors
 - $WAR \rightarrow \neg WAG \wedge \neg WAB, WAG \rightarrow \neg WAR \wedge \neg WAB, WAB \rightarrow \neg WAR \wedge \neg WAG, \dots$ // at most 1 color
 - // adjacent states must be different colors
 - $WAR \rightarrow \neg NTR \wedge \neg SAR, WAG \rightarrow \neg NTG \wedge \neg SAG, WAB \rightarrow \neg NTB \wedge \neg SAB...$
 - $NTR \rightarrow \neg WAR \wedge \neg SAR \wedge \neg QR...$
 - note: $KB \not\models WAR$ (does not entail)
 - however, $KB \models WAB \rightarrow VB$, and $KB \cup \{WAB\} \models VB$

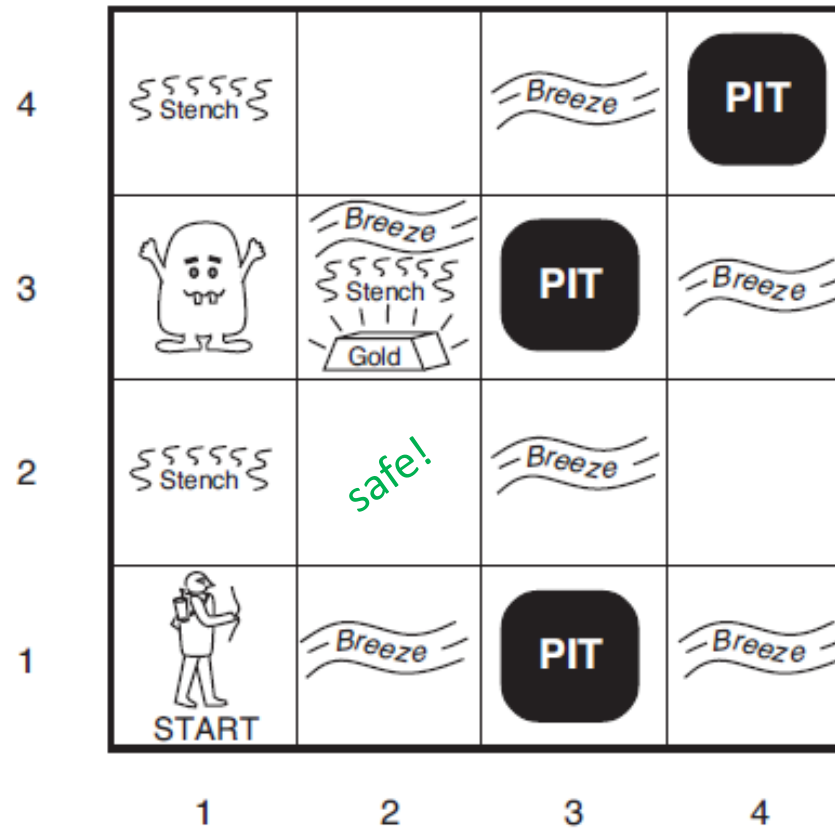
Example: Wumpus World

- the goal of the agent is to find the gold without falling in a pit or getting eaten by the wumpus (there is only 1, and it can't move)
- the agent does not know a priori where the pits or wumpus are located
- the agent can only sense breezes, stenches, and glitter
- a breeze is felt in rooms adjacent to pits, and a stench can be sensed in rooms adjacent to the wumpus
- a room is *safe* to explore if it is known not have the pit wumpus



- A = Agent
- B = Breeze
- G = Glitter, Gold
- OK = Safe square
- P = Pit
- S = Stench
- V = Visited
- W = Wumpus

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1



- after visiting rooms (1,1), (1,2), and (2,1), the agent should be able to infer that room (2,2) is safe
- $KB \cup \{S_{12}, B_{21}\} \models \text{safe}_{22}$

We can use propositions like:

- $W44$ = "the wumpus is in room 4,4"
- $B22$ = "there is a breeze in 2,2"
- $S13$ = "there is a stench in 1,3"
- $pit32$ = "there is a pit in room 3,2"
- $safe33$ = "room 3,3 is safe"

KB = {

1. $W11 \rightarrow S21 \wedge S12$	17. $P11 \rightarrow B21 \wedge B12$	33. $\neg W11 \wedge \neg P11 \rightarrow safe11$
2. $W12 \rightarrow S22 \wedge S11 \wedge S13$	18. $P12 \rightarrow B22 \wedge B11 \wedge B13$	34. $\neg W12 \wedge \neg P12 \rightarrow safe12$
3. $W13 \rightarrow S23 \wedge S12 \wedge S14$	19. $P13 \rightarrow B23 \wedge B12 \wedge B14$	35. $\neg W13 \wedge \neg P13 \rightarrow safe13$
4. $W14 \rightarrow S24 \wedge S13$	20. $P14 \rightarrow B24 \wedge B13$	36. $\neg W14 \wedge \neg P14 \rightarrow safe14$
5. $W21 \rightarrow S11 \wedge S31 \wedge S22$	21. $P21 \rightarrow B11 \wedge B31 \wedge B22$	37. $\neg W21 \wedge \neg P21 \rightarrow safe21$
6. $W22 \rightarrow S12 \wedge S32 \wedge S21 \wedge S23$	22. $P22 \rightarrow B12 \wedge B32 \wedge B21 \wedge B23$	38. $\neg W22 \wedge \neg P22 \rightarrow safe22$
7. $W23 \rightarrow S13 \wedge S33 \wedge S22 \wedge S24$	23. $P23 \rightarrow B13 \wedge B33 \wedge B22 \wedge B24$	39. $\neg W23 \wedge \neg P23 \rightarrow safe23$
8. $W24 \rightarrow S14 \wedge S34 \wedge S23$	24. $P24 \rightarrow B14 \wedge B34 \wedge B23$	40. $\neg W24 \wedge \neg P24 \rightarrow safe24$
9. $W31 \rightarrow S21 \wedge S41 \wedge S32$	25. $P31 \rightarrow B21 \wedge B41 \wedge B32$	41. $\neg W31 \wedge \neg P31 \rightarrow safe31$
10. $W32 \rightarrow S22 \wedge S42 \wedge S31 \wedge S33$	26. $P32 \rightarrow B22 \wedge B42 \wedge B31 \wedge B33$	42. $\neg W32 \wedge \neg P32 \rightarrow safe32$
11. $W33 \rightarrow S23 \wedge S43 \wedge S32 \wedge S34$	27. $P33 \rightarrow B23 \wedge B43 \wedge B32 \wedge B34$	43. $\neg W33 \wedge \neg P33 \rightarrow safe33$
12. $W34 \rightarrow S24 \wedge S44 \wedge S33$	28. $P34 \rightarrow B24 \wedge B44 \wedge B33$	44. $\neg W34 \wedge \neg P34 \rightarrow safe34$
13. $W41 \rightarrow S31 \wedge S42$	29. $P41 \rightarrow B31 \wedge B42$	45. $\neg W41 \wedge \neg P41 \rightarrow safe41$
14. $W42 \rightarrow S32 \wedge S41 \wedge S43$	30. $P42 \rightarrow B32 \wedge B41 \wedge B43$	46. $\neg W42 \wedge \neg P42 \rightarrow safe42$
15. $W43 \rightarrow S33 \wedge S42 \wedge S44$	31. $P43 \rightarrow B33 \wedge B42 \wedge B44$	47. $\neg W43 \wedge \neg P43 \rightarrow safe43$
16. $W44 \rightarrow S34 \wedge S43$	32. $P44 \rightarrow B34 \wedge B43$	48. $\neg W44 \wedge \neg P44 \rightarrow safe44$

}

Semantics

- semantics refers to “meaning” of sentences, and relationships among them
 - this is defined using Model Theory
 - models describe states of the world, and are used to give “interpretations” of sentences
- Truth-functional semantics
 - each sentence is assumed to be either True or False in the world
 - (Law of the Excluded Middle) – there is no in-between
 - propositions correspond to “facts” about the state of the world, which can only be True or False
 - good example: plutoCold, mercuryCold (in our universe, the first is T, the second is F)
 - bad example: surface_temp_of_pluto (value can only be T or F)
- in Propositional Logic, *models* are *truth assignments* over all propositional symbols (that appear in the KB)
 - {A=F, B=F, C=T ... P=T, Q=F, R=F}
 - {mercuryCold=F, mercuryWarm=F, mercuryHot=T...earthCold=F, earthWarm=T, earthHot=F... plutoCold=T, plutoWarm=F, plutoHot=F}

also, uncertainty will be accounted for via multiple models...

Semantics

- Compositionality

- a model defines the truth value for all atomic sentences
- given a model, the truth value of ANY sentence can be computed by combining truth values of sub-sentences using truth tables
- these are pretty straightforward in PropLog (except for \rightarrow)

A	B	$\neg A$	$A \vee B$	$A \wedge B$	$A \oplus B$	$A \rightarrow B$	$A \leftrightarrow B$
T	T	F	T	T	F	T	T
T	F	F	T	F	T	F	F
F	T	T	T	F	T	T	F
F	F	T	F	F	F	T	T

in a model $M = \{p=F, q=F, r=T\}$:

$$\neg p \vee (q \wedge r) \quad (\neg F) \vee (F \wedge T) = T \vee F = T$$

$$p \rightarrow (q \rightarrow p) \quad F \rightarrow (F \rightarrow F) = F \rightarrow T = T$$

$$p \wedge \neg p \quad F \wedge (\neg F) = F \wedge T = F$$

here's how you figure these out...

$P \rightarrow Q$: if the LHS (antecedents) are F, it doesn't matter what the RHS (consequent) is; only $T \rightarrow F$ is disallowed

Semantics

- We say a model M *satisfies* a sentence s iff the interpretation (or truth value) of s in M is True
- a sentence s is *satisfiable* if there is at least 1 model that satisfies it
- a sentence s is *unsatisfiable* if no model that satisfies it
- a sentence s is a *tautology* (or *valid*) if it is satisfied by ALL models
- examples:
 - satisfiable: $X, X \vee \neg Y \vee Z, (X \wedge \neg Y) \rightarrow Z$ (*what models make each of these True?*)
 - unsatisfiable: $X \wedge \neg X, P \oplus P, \neg Q \leftrightarrow Q$ (*convince yourself there are no models*)
 - tautologies: $A \vee \neg A, P \rightarrow (Q \rightarrow P)$ (*will be True in any model*)

Semantics

- semantic relationship between 2 sentences (or sets of sentences)
 - examples: $\{P\}$ and $\{\neg P\}$ can't both be True (i.e. satisfied by same models)
 - if $\{P \vee Q, P \rightarrow R, Q \rightarrow S, \neg P\}$ is True, then $\{S\}$ must be True
- two sentences are *semantically equivalent* if they are satisfied by exactly the same models: $\alpha \equiv \beta$ iff $M(\alpha) = M(\beta)$
 - example: $A \rightarrow B \equiv \neg A \vee B$
- note: a set of sentences is True (or satisfied in a model) iff each sentence is True (equivalent to an implicit conjunction)

$$\{P \vee Q \wedge P \rightarrow R \wedge Q \rightarrow S \wedge \neg P\}$$

- **Entailment**

- $\alpha \models \beta$ iff all models that satisfy α also satisfy β
- captures the notion of “logical consequence”

Semantics

- show that $\{P \vee Q, Q \rightarrow R, \neg P\} \models \{R\}$
 - models that satisfy the premises (as a conjunction): $\{M3\}$
 - models that satisfy the consequents ($\{R\}$): $\{M1, M3, M5, M7\}$
 - $\{M3\} \subseteq \{M1, M3, M5, M7\}$

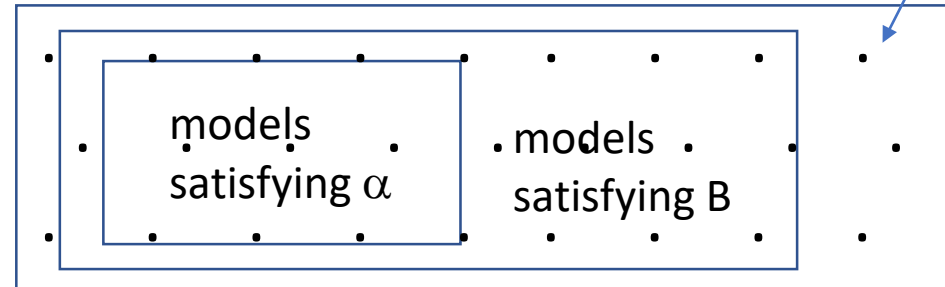
	P	Q	R	premises			premises as conjunction
				$P \vee Q$	$Q \rightarrow R$	$\neg P$	$\{P \vee Q, Q \rightarrow R, \neg P\}$
M0	0	0	0	0	1	1	0
M1	0	0	1	0	1	1	0
M2	0	1	0	1	0	1	0
M3	0	1	1	1	1	1	1
M4	1	0	0	1	1	0	0
M5	1	0	1	1	1	0	0
M6	1	1	0	1	0	0	0
M7	1	1	1	1	1	0	0

Semantics

- Deduction Theorem

- note that $P \rightarrow Q$ is valid looks like $P \models Q$, and they seem similar
- but they are different:
 - $P \rightarrow Q$ is a sentence (defined by syntax)
 - $P \models Q$ means P entails Q (defined by semantics)
- the Deduction Theorem shows that they are related:

- $\alpha \models \beta$ iff $\alpha \rightarrow \beta$ is valid
- (proof \Rightarrow) see Venn diagram



a model $M(i)$:
 $\{A=F, B=T, C=F$
 $D=T, E=F...\}$

in all models,
either α is false or
 β is true,
hence $\alpha \rightarrow \beta$ is
valid

- (proof \Leftarrow) if $\alpha \rightarrow \beta$ is valid, then all models satisfy it, so all models either make α false or β true; hence those model that satisfy α also satisfy β ; hence $\alpha \models \beta$

Inference

- is there a procedure to determine if $\alpha \models \beta$? (or $\text{KB} \models \text{query}$)
- model-checking (truth tables)
 - of course, we can just enumerate all models and check if those satisfy α also satisfy β
 - how many models are there? 2^n (for n propositional symbols)
 - it is *finite*, so the procedure will halt and return yes (entailed) or no

Inference

- model-checking is inefficient
 - we need a more practical procedure to determine whether $\alpha \models \beta$
- Proof Procedures: methods to determine whether $\alpha \models \beta$ *purely by syntactic manipulation*
 - aka “Inference Methods”, “Theorem-Proving”, “Automated Deduction”...
- Propositional Rules of Inference (ROI)
 - rules for generating new sentences from old sentences
 - a *sound* ROI only generates new sentences that are entailed
 - in this context, ‘ \vdash ’ means ‘derives’ by a ROI, i.e. ‘ $\alpha \vdash \beta$ ’ means ‘ β is derived from α ’
 - hence a rule R is **sound** iff for all sentences α, β , if $\alpha \vdash \beta$, then $\alpha \models \beta$
 - an ROI $\alpha \vdash \beta$ is *truth preserving* if the derived sentence β is semantically equivalent to α (satisfied by exactly the same models)

Rules of Inference

- example: Modus Ponens

- from P and $P \rightarrow Q$, we can derive Q
- $\{P, P \rightarrow Q\} \vdash Q$
- is MP sound?
- all the models that satisfy the premises (conjunction of P and $P \rightarrow Q$) also satisfy the derived sentence Q , so Q is entailed, so MP is sound

P	Q	$P \rightarrow Q$	premises (conj)	derived (Q)
0	0	1	0	0
0	1	1	0	1
1	0	0	0	0
1	1	1	1	1

Rules of Inference

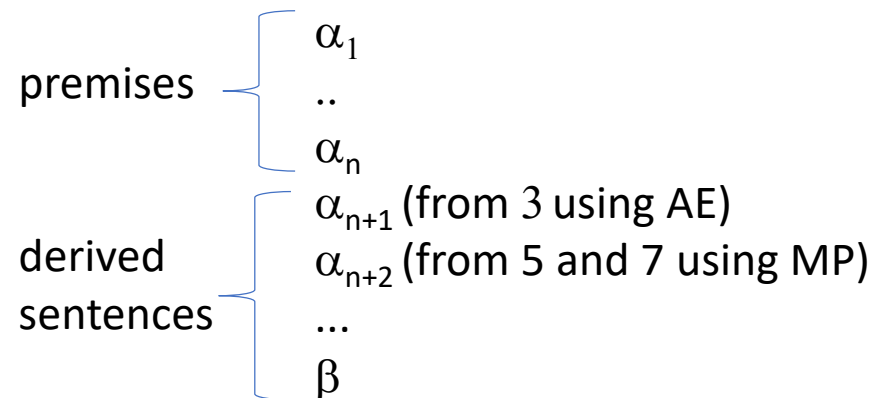
*These are inference 'schemas'.
A, B, and C are patterns
representing sub-sentences.*

*Can you prove that each of
these is sound?
Make truth tables.
AE and MP are easy.
Resolution is worth doing...*

	from this...	derive this...	comments
AndElimination (AE)	$A \wedge B$	A	
AndIntroduction (AI)	A, B	$A \wedge B$	
OrIntroduction	A, B	$A \vee B$	no such things as OrElimination!
Commutativity	$A \wedge B$	$B \wedge A$	truth-preserving
Distributivity	$A \vee (B \wedge C)$ $A \wedge (B \vee C)$	$(A \vee B) \wedge (A \vee C)$ $(A \wedge B) \vee (A \wedge C)$	
DoubleNegationElim (DN)	$\neg\neg A$	A	
DeMorgan's Laws (DM)	$\neg(A \vee B)$ $\neg(A \wedge B)$	$\neg A \wedge \neg B$ $\neg A \vee \neg B$	flip the operator
ImplicationElimination (IE)	$A \rightarrow B$	$\neg A \vee B$	truth-preserving
Modus Ponens (MP)	$A, A \rightarrow B$	B	pattern-matching, if LHS is matched, can derive RHS
Modus Tolens	$A \rightarrow B, \neg B$	$\neg A$	
contraposition	$A \rightarrow B$	$\neg B \rightarrow \neg A$	
Resolution	$A \vee B, \neg A \vee C$	$B \vee C$	requires 2 clauses with opposite literals

Natural Deduction

- Proof procedure to show that $\alpha \vDash \beta$
 - start by listing sentences in premise α
 - derive additional sentences using sound ROI
 - must be a *finite* sequence of steps ending in β
- number your sentences
- label each new sentence with ROI and sentences it was derived from



Example of Nat Ded (1)

from this...

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

$$B \wedge L \Rightarrow M$$

$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

A

B

...show that Q is entailed

premises

1. $P \rightarrow Q$
2. $L \wedge M \rightarrow P$
3. $B \wedge L \rightarrow M$
4. $A \wedge P \rightarrow L$
5. $A \wedge B \rightarrow L$
6. A
7. B

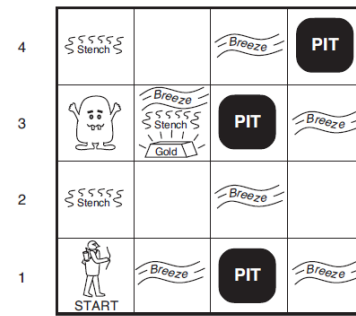
derivations

8. $A \wedge B$ [AndIntr, 6,7]
9. L [MP, 5,8]
10. $B \wedge L$ [AndIntr, 7,9]
11. M [MP, 10,3]
12. $L \wedge M$ [AndIntr, 9,11]
13. P [MP, 12,2]
14. $\neg P \vee Q$ [ImplElim, 1]
15. Q [Reso, 13,14]

Natural Deduction

- Why does Natural Deduction work?
 - it is not enough to show that $KB \cup \{\alpha_{n+1}, \dots, \alpha_{n+m}\} \models Q$ (for last step, assuming sound ROI)
 - we have to show that if Q is derived after a sequence of steps $\alpha_{n+1}, \dots, \alpha_{n+m}$ using sound ROI, then $KB \models Q$ (i.e. the original KB entails Q)
 - whenever we derive a new sentence by a sound ROI and add it to the premises, the set of entailments stays the same
 - suppose $KB \models Q$, and $M(KB) \subseteq M(Q)$
 - so if $KB \vdash \alpha_{n+1}$ (an intermediate sentence in proving Q) by a sound ROI, then $KB \models \alpha_{n+1}$, so $M(KB) \subseteq M(\alpha_{n+1})$, $M(KB \cup \alpha_{n+1}) = M(KB) \cap M(\alpha_{n+1}) = M(KB)$, so $KB \cup \{\alpha_{n+1}\} \models Q$
 - this property is known as “*monotonicity*” (i.e. adding entailed intermediate conclusions doesn’t affect what else is entailed)
 - similarly, if $KB \cup \alpha_{n+1} \vdash \alpha_{n+2}$, then $KB \cup \{\alpha_{n+1}\} \models \alpha_{n+2}$ and $M(KB \cup \{\alpha_{n+1}, \alpha_{n+2}\}) \subseteq M(Q)$, so $KB \cup \{\alpha_{n+1}, \alpha_{n+2}\} \models Q$
 - if proof has $m+1$ steps $KB, \alpha_{n+1}, \alpha_{n+m}, Q$, then $KB \cup \{\alpha_{n+1}, \alpha_{n+m}\} \models Q$ (by induction)

Example of Nat Ded (2): Wumpus World



after visiting rooms (1,1), (1,2), and (2,1), the agent should be able to infer that room (2,2) is safe

KB = {

- | | | |
|--------------------------|--------------------------|-----------------------|
| 1. W11->S21^S12 | 17. P11->B21^B12 | 33. -W11^-P11->safe11 |
| 2. W12->S22^S11^S13 | 18. P12->B22^B11^B13 | 34. -W12^-P12->safe12 |
| 3. W13->S23^S12^S14 | 19. P13->B23^B12^B14 | 35. -W13^-P13->safe13 |
| 4. W14->S24^S13 | 20. P14->B24^B13 | 36. -W14^-P14->safe14 |
| 5. W21->S11^S31^S22 | 21. P21->B11^B31^B22 | 37. -W21^-P21->safe21 |
| 6. W22->S12^S32^S21^S23 | 22. P22->B12^B32^B21^B23 | 38. -W22^-P22->safe22 |
| 7. W23->S13^S33^S22^S24 | 23. P23->B13^B33^B22^B24 | 39. -W23^-P23->safe23 |
| 8. W24->S14^S34^S23 | 24. P24->B14^B34^B23 | 40. -W24^-P24->safe24 |
| 9. W31->S21^S41^S32 | 25. P31->B21^B41^B32 | 41. -W31^-P31->safe31 |
| 10. W32->S22^S42^S31^S33 | 26. P32->B22^B42^B31^B33 | 42. -W32^-P32->safe32 |
| 11. W33->S23^S43^S32^S34 | 27. P33->B23^B43^B32^B34 | 43. -W33^-P33->safe33 |
| 12. W34->S24^S44^S33 | 28. P34->B24^B44^B33 | 44. -W34^-P34->safe34 |
| 13. W41->S31^S42 | 29. P41->B31^B42 | 45. -W41^-P41->safe41 |
| 14. W42->S32^S41^S43 | 30. P42->B32^B41^B43 | 46. -W42^-P42->safe42 |
| 15. W43->S33^S42^S44 | 31. P43->B33^B42^B44 | 47. -W43^-P43->safe43 |
| 16. W44->S34^S43 | 32. P44->B34^B43 | 48. -W44^-P44->safe44 |

}

Natural Deduction Proof of $KB \wedge \text{Facts} \models \text{safe22}$:

Facts = { 49. $\neg B_{11}$, 50. $\neg S_{11}$, 51. $\neg B_{12}$, 52. S_{12} , 53. B_{21} , 54. $\neg S_{21}$ }

55. $\neg W_{22} \vee (S_{12} \wedge S_{32} \wedge S_{21} \wedge S_{23})$ [Impl Elim, 6]

56. $(\neg W_{22} \vee S_{12}) \wedge (\neg W_{22} \vee S_{32}) \wedge (\neg W_{22} \vee S_{21}) \wedge (\neg W_{22} \vee S_{23})$ [Distrib, 55]

57. $\neg W_{22} \vee S_{21}$ [And Elim, 56]

58. $S_{21} \vee \neg W_{22}$ [Commut, 57]

59. $\neg S_{21} \rightarrow \neg W_{22}$ [Impl Intro, 58]

60. $\neg W_{22}$ [MP, 59, 54]

61. $\neg P_{22} \vee (B_{12} \wedge B_{32} \wedge B_{21} \wedge B_{23})$ [Impl Elim, 22]

62. $(\neg P_{22} \vee B_{12}) \wedge (\neg P_{22} \vee B_{32}) \wedge (\neg P_{22} \vee B_{21}) \wedge (\neg P_{22} \vee B_{23})$ [Distrib, 61]

63. $\neg P_{22} \vee B_{12}$ [And Elim, 62]

64. $B_{12} \vee \neg P_{22}$ [Commut, 63]

65. $\neg B_{12} \rightarrow \neg P_{22}$ [Impl Intro, 64]

66. $\neg P_{22}$ [MP, 65, 51]

67. $\neg W_{22} \wedge \neg P_{22}$ [And Intro, 60, 66]

68. safe22 [MP, 38, 67]

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

Natural Deduction

- limitations
 - it can be difficult (but not impossible) to find the right sequence of derivations automatically
 - you could use a Search and try applying all ROI to all combinations of sentences till you generate the query (i.e. as the goal)
 - in theory, is Nat Ded a *complete* proof procedure???
 - can every query that is entailed by a KB be proved in a finite number of steps?
 - one can show that certain combinations of ROI are sufficient to guarantee that a proof always exists for entailed sentences (in Propositional Logic)

Forward Chaining

- let's explore more practical theorem-proving methods
- Forward-Chaining (FC) is super-easy: you only need Modus Ponens!
- however, FC only works on definite-clause KBs
 - a *clause* is a disjunction of literals, e.g. $A \vee \neg B \vee C \vee \neg D$
 - a *Horn clause* is a clause with at most one positive literal, e.g. $\neg A \vee B \vee \neg C$
 - a *definite clause* is a clause with exactly one positive literal
- where do definite clauses come from? facts and conjunctive rules
 - facts: A, B (note – negations are not allowed!)
 - rules with conjunct. of pos. lits as antecedents and 1 consequent
 - $A \wedge B \wedge C \rightarrow D$ (conjunctive rule)
 - $\neg A \vee \neg B \vee \neg C \vee D$ (definite clause, by Implication Elimination)

```

function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
           q, the query, a proposition symbol
  count ← a table, where count[c] is initially the number of symbols in clause c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  queue ← a queue of symbols, initially symbols known to be true in KB

  while queue is not empty do
    p ← POP(queue)
    if p = q then return true
    if inferred[p] = false then
      inferred[p] ← true
      for each clause c in KB where p is in c.PREMISE do
        decrement count[c]
        if count[c] = 0 then add c.CONCLUSION to queue
  return false

```

in this context,
'clause' means
'rule' (i.e. definite
clause)

initial facts

- the key idea in FC is to use a queue (sometimes called an 'agenda') to keep track of facts that have been inferred (initially, just the given facts)
- with each new fact inferred, we check which rules can be triggered (i.e. when all their antecedents have been satisfied), and then we put the consequents in the queue
- there are ways to make this efficient for large KBs by indexing on which rules have which propositions as antecedents (to quickly figure out which rules are triggered by new facts)

Example of Forward Chaining

	<u>count</u>
1. $P \rightarrow Q$	1
2. $L \wedge M \rightarrow P$	2
3. $B \wedge L \rightarrow M$	2
4. $A \wedge P \rightarrow L$	2
5. $A \wedge B \rightarrow L$	2
6. A	0
7. B	0

- agenda:

- 1. A, B
- 2. ~~A, B~~, L // rule 5 is triggered
- 3. ~~A, B, L~~, M // rule 3
- 4. ~~A, B, L, M~~, P // rule 2
- 5. ~~A, B, L, M, P~~, Q // rule 1 fires
- stop since query was generated

*note, in this illustration,
I am not popping things out
of the queue*

Forward Chaining

- so using FC requires the KB to be formulated as definite clauses (i.e. a rule base)
- in theory, this is not always possible
 - for example, if $\{\neg P\}$ or $\{P \vee Q\}$ or $\{P \wedge Q \rightarrow R \vee S\}$ is in the KB, it can't be transformed into definite clauses
 - these examples represent *uncertainty*, which isn't permitted
 - hence this requirement limits expressiveness (not full Propositional Logic)
- in practice, it is often possible to express KBs for real problems in definite clause form, with judicious choice of propositions

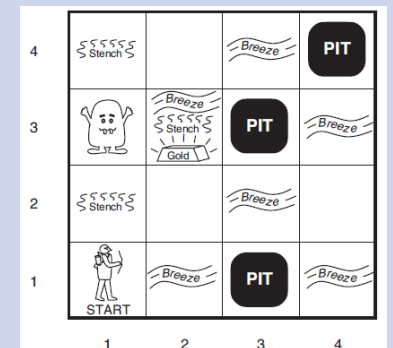
To show that *safe22* in the Wumpus World, we have to re-write the KB as definite clauses, which can be achieved by using new propositions...

PropSyms:

- **WF = wumpus-free**
- **PF = pit-free**
- **C = calm**
- **US = unstenchy**

note that we chose new propositional symbols representing information in a *negative* way

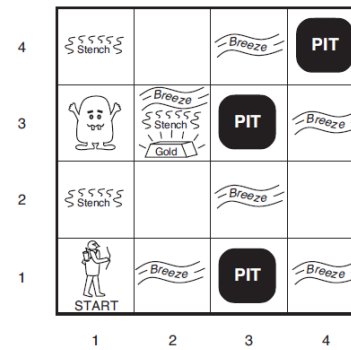
1. C11->PF21	25. C31->PF21	49. US11->WF21	73. US31->WF21	97. WF11^PF11->safe11
2. C11->PF12	26. C31->PF41	50. US11->WF12	74. US31->WF41	98. WF12^PF12->safe12
3. C12->PF22	27. C31->PF32	51. US12->WF22	75. US31->WF32	99. WF13^PF13->safe13
4. C12->PF11	28. C32->PF22	52. US12->WF11	76. US32->WF22	100. WF14^PF14->safe14
5. C12->PF13	29. C32->PF42	53. US12->WF13	77. US32->WF42	101. WF21^PF21->safe21
6. C13->PF23	30. C32->PF31	54. US13->WF23	78. US32->WF31	102. WF22^PF22->safe22
7. C13->PF12	31. C32->PF33	55. US13->WF12	79. US32->WF33	103. WF23^PF23->safe23
8. C13->PF14	32. C33->PF23	56. US13->WF14	80. US33->WF23	104. WF24^PF24->safe24
9. C14->PF24	33. C33->PF43	57. US14->WF24	81. US33->WF43	105. WF31^PF31->safe31
10. C14->PF13	34. C33->PF32	58. US14->WF13	82. US33->WF32	106. WF32^PF32->safe32
11. C21->PF11	35. C33->PF34	59. US21->WF11	83. US33->WF34	107. WF33^PF33->safe33
12. C21->PF31	36. C34->PF24	60. US21->WF31	84. US34->WF24	108. WF34^PF34->safe34
13. C21->PF22	37. C34->PF44	61. US21->WF22	85. US34->WF44	109. WF41^PF41->safe41
14. C22->PF12	38. C34->PF33	62. US22->WF12	86. US34->WF33	110. WF42^PF42->safe42
15. C22->PF32	39. C41->PF31	63. US22->WF32	87. US41->WF31	111. WF43^PF43->safe43
16. C22->PF21	40. C41->PF42	64. US22->WF21	88. US41->WF42	112. WF44^PF44->safe44
17. C22->PF23	41. C42->PF32	65. US22->WF23	89. US42->WF32	
18. C23->PF13	42. C42->PF41	66. US23->WF13	90. US42->WF41	
19. C23->PF33	43. C42->PF43	67. US23->WF33	91. US42->WF43	
20. C23->PF22	44. C43->PF33	68. US23->WF22	92. US43->WF33	
21. C23->PF24	45. C43->PF42	69. US23->WF24	93. US43->WF42	
22. C24->PF14	46. C43->PF44	70. US24->WF14	94. US43->WF44	
23. C24->PF34	47. C44->PF34	71. US24->WF34	95. US44->WF34	
24. C24->PF23	48. C44->PF43	72. US24->WF23	96. US44->WF43	



FC proof of $KB^{\wedge}Facts \models safe22$

Facts:

- 113. C11 // room 1,1 is calm (no breeze)
- 114. US11 // room 1,1 is unstenchy
- 115. C12 // room 1,2 is calm
- 116. US21 // room 2,1 is unstenchy



inferred	agenda
	C11 US11 C12 US21 // initialize by pushing facts
C11	US11 C12 US21 <u>PF12 PF21</u> // C11 causes 2 new facts to be pushed onto agenda from rules 1&2
US11	C12 US21 PF12 PF21 <u>WF12 WF21</u>
C12	US21 PF12 PF21 WF12 WF21 <u>PF11 PF22 PF13</u> // C12 causes rules 3-5 to fire
US21	PF12 PF21 WF12 WF21 PF11 PF22 PF13 <u>WF11 WF31 WF22</u> // rules 59-61
PF12 PF21 WF12 WF21 PF11 PF22 PF13	// these just pop off without pushing anything new
WF11	WF31 WF22 safe11 // since WF11 and PF11 have been inferred, rule 97 fires
WF31	WF22 safe11
WF22	safe11 safe22 // since WF22 and PF22 have been inferred, rule 102 fires
safe11	safe22
safe22	// found what we were looking for, showing the query is entailed; also, agenda becomes empty

Back-Chaining

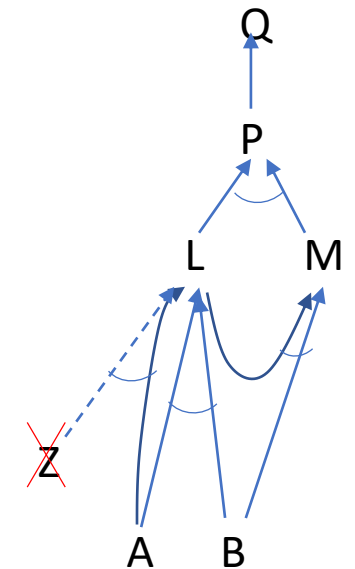
- one of the problems with FC is that it can waste time generating a lot of unnecessary inferences that are irrelevant to the query
- back-chaining (BC) also works on definite-clause KBs, but it works *backwards* from the goal to find supporting facts
- hence BC is more efficient because it is goal-directed
- BC is in fact the basis of PROLOG (as we will see later when we cover FOL)

Back-Chaining

- BC uses a *goal stack* (initialized by pushing the query)
- with each iteration:
 - pop the goal on the top of the stack
 - check to see if it is a known fact
 - otherwise, find a rule that has the goal as consequent, and push the antecedents onto the stack as subgoals
 - the algorithm terminates when the stack becomes empty (success, showing the query is entailed, because it has been reduced to known facts)
- important: back-tracking
 - if some subgoals cannot be proved, BC must back-track and try another rule to prove goal

Example of Backward Chaining

- | | | |
|----|----------------------------|---|
| 1. | $P \rightarrow Q$ | • Q // initialize with query |
| 2. | $L \wedge M \rightarrow P$ | • P // pop Q, replace with antecedent of rule 1 |
| 3. | $B \wedge L \rightarrow M$ | • L, M // replace P with ants. of rule 2 |
| 4. | $A \wedge Z \rightarrow L$ | • A, Z, M // pop L, push A, P from rule 4 |
| 5. | $A \wedge B \rightarrow L$ | • Z, M // pop A (known fact) |
| 6. | A | • // since Z is not provable, back-track to other rule for L |
| 7. | B | • A, B, M // pop A (fact) |
| | | • M // pop B (fact) |
| | | • B, L // pop M, try rule 3, push B, L |
| | | • L // pop B (fact) |
| | | • A, Z // pop L, try rule 4 |
| | | • Z // pop A (fact) |
| | | • // since Z is not provable, back-track to other rule for L |
| | | • A, B |
| | | • B // pop A (fact) |
| | | • \emptyset // pop B (fact); stack becomes empty; return success! |



visualizing the proof tree as an “and-or” graph

(note: This example is modified from Fig 7.16 in the book to simplify for illustration purposes. The P in $A \wedge P \rightarrow L$ was replaced with Z, to avoid the complication of checking for repeated subgoals, which would succeed implicitly, representing a loop. In this context, however, that technical detail is an unnecessary distraction.)

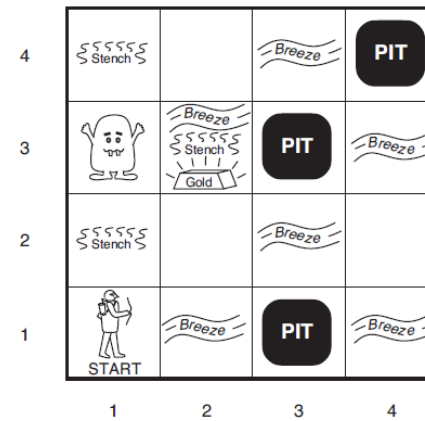
Back-chaining using Propositional Logic (Recursive stack-based version)

```
Backchain(KB, query)
  stack.push(query) // initialize
  return BC(KB, stack)
```

```
BC(KB, stack)
  if stack empty, return True
  subgoal ← stack.pop()
  if subgoal ∈ KB, return BC(KB, stack) // a known fact
  for each rule  $a_1..a_n \rightarrow$  subgoal in KB: // choice point for back-tracking
    stack.push( $a_1..a_n$ )
    result ← BC(KB, stack)
    if result = True, return True
    else remove  $a_1..a_n$  from stack
  return False
```


BC proof of KB^Facts |= safe22
 (using the definite-clause KB from slide 34)

- Facts:
 113. C11
 114. US11
 115. C12
 116. US21



goal stack	
safe22	push query
<u>WF22</u> PF22	rule102
<u>US12</u> PF22	try rule 51 for WF22
<u>US21</u> PF22	fail, back-track; try rule 61 for WF22
PF22	succeed (116); US21 pops off
<u>C21</u>	try rule 21 for PF22
C12	fail; back-track; try rule 61
∅	C12 is a known fact (115); pop off stack becomes empty; proof succeeds

Resolution Refutation

- FC and BC are effective proof procedures, but they are limited because they are not complete (not all KBs are in definite-clause form)
- Is there a complete proof procedure that is simpler than Nat. Ded.?
- Resolution Refutation proofs – you can prove any entailed sentence, and all you need is one ROI: *resolution*

$$\frac{A \vee B \vee \dots, \neg A \vee C \vee \dots}{B \vee \dots \vee C \vee \dots}$$

- prerequisite: you have to convert your KB into CNF (Conjunctive-Normal Form, i.e. clauses), which you can always do

simple example: $A \wedge B \wedge \neg C \rightarrow D \wedge E$ can be transformed into 2 clauses (not necessarily Horn) that are equivalent:
 $(\neg A \vee \neg B \vee C \vee D), (\neg A \vee \neg B \vee C \vee E)$

Conversion to CNF

clause = disjunction, e.g. $(A \vee B)$

CNF = conjunction of clauses, e.g. $(A \vee B) \wedge (C \vee D)$

these can be treated like multiple sentences, $\{(A \vee B), (C \vee D)\}$

- procedure for converting any propositional sentence to CNF (p. 227)
 1. eliminate implications (and biconditionals)
 2. push negations inward (using DoubleNegElim and DeMorgan's)
 3. distribute Or's over And's (till expression is 2-level Boolean CNF)
 4. break final conjunction into multiple clauses
- example: $A \wedge B \wedge \neg C \rightarrow D \wedge E$
 1. $\neg(A \wedge B \wedge \neg C) \vee D \wedge E$ // implication elimination
 2. $(\neg A \vee \neg B \vee \neg \neg C) \vee (D \wedge E)$ // push negations inward
 3. $(\neg A \vee \neg B \vee C \vee D) \wedge (\neg A \vee \neg B \vee C \vee E)$ // distribution
 - 4a. $(\neg A \vee \neg B \vee C \vee D)$
 - 4b. $(\neg A \vee \neg B \vee C \vee E)$

Refutation Proofs

- negate the query and add it to the KB
- if the query was entailed, this creates an inconsistency (unsatisfiable),
 $M(KB \cup \{\neg q\}) = \emptyset$
- thus we should be able to derive the empty clause (which means “false” or “inconsistent”)

simple example:

suppose $KB = \{A, A \rightarrow B\}$ and $q = B$

negate query and append it: $\{A, A \rightarrow B, \neg B\}$

convert to CNF $\{A, \neg A \vee B, \neg B\}$

1. A

2. $\neg A \vee B$

3. $\neg B$

4. $\neg A$ // resolve 2 and 3

5. \emptyset // resolve 1 and 4, empty clause

this means we proved $KB \models B$

Refutation Proofs

- Why do refutation proofs work?
- like “proof by contradiction”
- no models satisfy both KB and $\neg q$ (empty intersection)
 - if $KB \models q$, then $M(KB \cup \{\neg q\}) = \emptyset$, hence unsatisfiable



Example of Resolution Refutation Proof

KB:

1. $P \rightarrow Q$
2. $L \wedge M \rightarrow P$
3. $B \wedge L \rightarrow M$
4. $A \wedge P \rightarrow L$
5. $A \wedge B \rightarrow L$
6. A
7. B

query: Q

CNF:

1. $\neg P \vee Q$
2. $\neg L \vee \neg M \vee P$
3. $\neg B \vee \neg L \vee M$
4. $\neg A \vee \neg P \vee L$
5. $\neg A \vee \neg B \vee L$
6. A
7. B
8. $\neg Q$

// negated query

9. $\neg P$ // reso on 1 and 8 (eliminate Q)
10. $\neg L \vee \neg M$ // reso 2,9
11. $\neg A \vee \neg B \vee \neg M$ // reso 5,10 (eliminate L)
12. $\neg A \vee \neg B \vee \neg B \vee \neg L$ // reso 11,3 (eliminate M)
13. $\neg A \vee \neg B \vee \neg L$ // (factoring, combine $\neg B$ s)
14. $\neg A \vee \neg B \vee \neg A \vee \neg B$ // reso 13,5
15. $\neg A \vee \neg B$ // factoring
16. $\neg B$ // reso 15,7
17. \emptyset // reso 16,8; empty clause!

Resolution Proof Procedure

```
function PL-RESOLUTION( $KB, \alpha$ ) returns true or false  
  inputs:  $KB$ , the knowledge base, a sentence in propositional logic  
            $\alpha$ , the query, a sentence in propositional logic  
  
   $clauses \leftarrow$  the set of clauses in the CNF representation of  $KB \wedge \neg\alpha$   
   $new \leftarrow \{\}$   
  loop do  
    for each pair of clauses  $C_i, C_j$  in  $clauses$  do  
       $resolvents \leftarrow$  PL-RESOLVE( $C_i, C_j$ )  
      if  $resolvents$  contains the empty clause then return true  
       $new \leftarrow new \cup resolvents$   
  if  $new \subseteq clauses$  then return false  
   $clauses \leftarrow clauses \cup new$ 
```

Wumpus World
Clauses (CNF) for
Resolution

1. -W11vS21	25. -W31vS21	49. -P11vB21	73. -P31vB21	97. W11 v P11 v safe11
2. -W11vS12	26. -W31vS41	50. -P11vB12	74. -P31vB41	98. W12 v P12 v safe12
3. -W12vS22	27. -W31vS32	51. -P12vB22	75. -P31vB32	99. W13 v P13 v safe13
4. -W12vS11	28. -W32vS22	52. -P12vB11	76. -P32vB22	100. W14 v P14 v safe14
5. -W12vS13	29. -W32vS42	53. -P12vB13	77. -P32vB42	101. W21 v P21 v safe21
6. -W13vS23	30. -W32vS31	54. -P13vB23	78. -P32vB31	102. W22 v P22 v safe22
7. -W13vS12	31. -W32vS33	55. -P13vB12	79. -P32vB33	103. W23 v P23 v safe23
8. -W13vS14	32. -W33vS23	56. -P13vB14	80. -P33vB23	104. W24 v P24 v safe24
9. -W14vS24	33. -W33vS43	57. -P14vB24	81. -P33vB43	105. W31 v P31 v safe31
10. -W14vS13	34. -W33vS32	58. -P14vB13	82. -P33vB32	106. W32 v P32 v safe32
11. -W21vS11	35. -W33vS34	59. -P21vB11	83. -P33vB34	107. W33 v P33 v safe33
12. -W21vS31	36. -W34vS24	60. -P21vB31	84. -P34vB24	108. W34 v P34 v safe34
13. -W21vS22	37. -W34vS44	61. -P21vB22	85. -P34vB44	109. W41 v P41 v safe41
14. -W22vS12	38. -W34vS33	62. -P22vB12	86. -P34vB33	110. W42 v P42 v safe42
15. -W22vS32	39. -W41vS31	63. -P22vB32	87. -P41vB31	111. W43 v P43 v safe43
16. -W22vS21	40. -W41vS42	64. -P22vB21	88. -P41vB42	112. W44 v P44 v safe44
17. -W22vS23	41. -W42vS32	65. -P22vB23	89. -P42vB32	
18. -W23vS13	42. -W42vS41	66. -P23vB13	90. -P42vB41	
19. -W23vS33	43. -W42vS43	67. -P23vB33	91. -P42vB43	
20. -W23vS22	44. -W43vS33	68. -P23vB22	92. -P43vB33	
21. -W23vS24	45. -W43vS42	69. -P23vB24	93. -P43vB42	
22. -W24vS14	46. -W43vS44	70. -P24vB14	94. -P43vB44	
23. -W24vS34	47. -W44vS34	71. -P24vB34	95. -P44vB34	
24. -W24vS23	48. -W44vS43	72. -P24vB23	96. -P44vB43	

ResoRef proof of $KB^{\wedge}Facts \models safe22$

Facts:

113. -B11

114. -S11

115. -B12

116. S12

117. B21

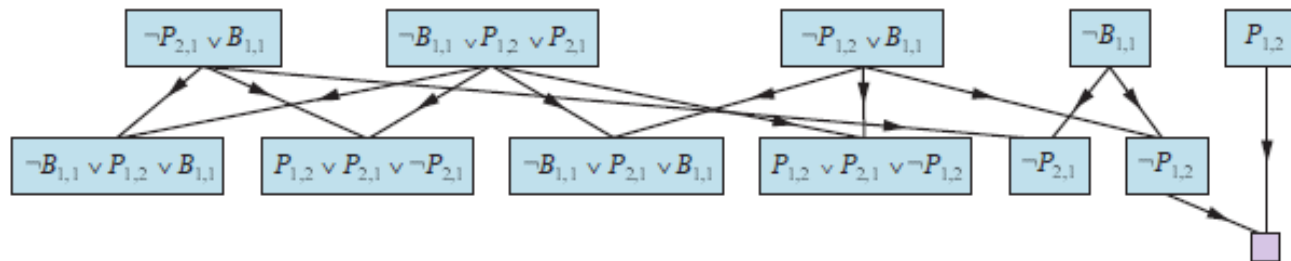
118. -S21

119. -safe22 // negation of query

new clauses	annotation
120. -P22	Reso on 115 & 62 (-P22vB12)
121. -W22	Reso on 118 & 16 (-W22vS21)
122. W22 v safe22	Reso on 120 & 102 (W22 v P22 v safe22)
123. safe22	Reso on 121 & 122
124. \emptyset derived empty clause; proof succeeds	Reso on 119 & 123

Resolution Refutation

- Resolution as a Search for the empty clause
- nodes in Search Tree = clauses, but they have multiple parents
- there are often many clauses that can be resolved (most are irrelevant)



- heuristics to make the resolution search more efficient:
 - unit-clause heuristic:
 - choose pairs of clauses that can resolve, where at least one clause is just a single literal
 - rationale: size of resolvent of clauses of size n and m is $n+m-2$, so if one is a unit clause, the resolvent shrinks in size to $n-1$ (closer to the goal of size 0 for the empty clause)
 - this is effective, but incomplete (there are some proofs you can't do if you always use the unit-clause heuristic)

Resolution Refutation

- other resolution heuristics (resolution strategies)
 - input resolution: always choose one of the clauses from the input set (premise clauses)
 - linear resolution: always choose the previously resolved clause
- we will discuss these heuristics in more detail Ch. 9 (Inference in FOL)

Resolution Refutation

- Is it complete proof procedure? can it determine whether *any* sentence is entailed?
- Ground Resolution Theorem:
 - if a set of sentences S is unsatisfiable, then there exists a finite sequence of resolution steps that will generate the empty clause.
 - the textbook restates this as “the empty clause will be contained in the resolution closure”
 - the proof involves showing that: suppose S is unsat but $RC(S)$ does not contain the empty clause; then we can construct a model out of the clauses in $RC(S)$ - contradiction
- Theorem: **Resolution refutation is a complete proof procedure.**
 - if $\alpha \not\models \beta$, then there exists a finite sequence of resolution steps (starting from the CNF of $\{\alpha \wedge \neg\beta\}$) that will generate the empty clause.
- generally, we do not try to show the converse, i.e. that if b is not entailed, resolution should stop and say so, e.g. when it runs out of clauses that can be resolved
 - theoretically you could do it in Prop Log, but it depend on the $RC(S)$ being finite (requires factoring)

Satisfiability

- another propositional theorem-proving strategy
- Sat methods can test if a set of sentences is unsatisfiable (like in a Refutation proof)
- more commonly, Sat methods are used on satisfiable KBs – the goal is to generate a model (where the truth values are the solution to a problem)
- this is a (slightly) more efficient form of model-checking

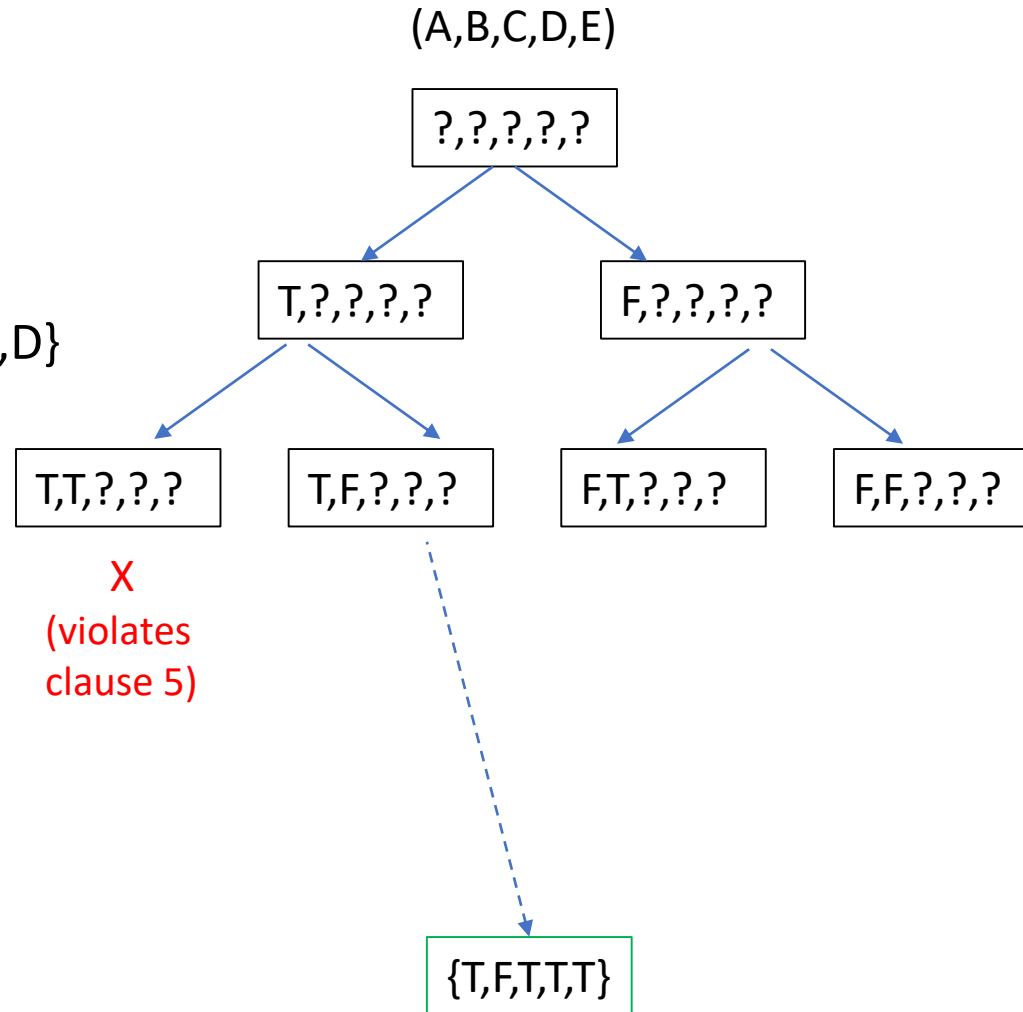
DPLL

- a truth assignment (as a model) is a specification of truth values (T,F,?) for all propositional symbols in a KB
 - examples: {F,F,F,F,F}, {T,F,?,?,?}
- Search for complete truth assignment (like CSP)
 - $KB = \{-D \vee C, A \wedge B \rightarrow E, -D \rightarrow -E, C \rightarrow E, -B \wedge D\}$, $CNF = \{-D \vee C, -A \vee -B \vee E, D \vee -E, -C \vee E, -B, D\}$
 - props are {A,B,C,D,E}
 - initial state={?, ?, ?, ?, ?}
 - goal states={F,F,T,T,T} and {T,F,T,T,T}
- “Davis-Putnam-Logemann-Loveland” (DPLL) procedure
 - convert propositional KB into CNF
 - start with an empty truth assignment {?, ?, ?, ..., ?}
 - try binding one more variable at a time
 - back-track whenever a clause is violated
 - quit when a complete assignment is found that satisfies all clauses

DPLL

find a model for:

$\{\neg D \vee C, \neg A \vee \neg B \vee E, D \vee \neg E, \neg C \vee E, \neg B, D\}$



(this tree assigns the props in alphabetical order by default, but DPLL could choose a different prop at each node using the unit-clause or pure symbol heuristics discussed on the next slide)

function DPLL-SATISFIABLE?(*s*) **returns** *true* or *false*

inputs: *s*, a sentence in propositional logic

clauses \leftarrow the set of clauses in the CNF representation of *s*

symbols \leftarrow a list of the proposition symbols in *s*

return DPLL(*clauses*, *symbols*, { })

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

if every clause in *clauses* is true in *model* **then return** *true*

if some clause in *clauses* is false in *model* **then return** *false*

P, *value* \leftarrow FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)

if *P* is non-null **then return** DPLL(*clauses*, *symbols* - *P*, *model* \cup {*P*=*value*})

P, *value* \leftarrow FIND-UNIT-CLAUSE(*clauses*, *model*)

if *P* is non-null **then return** DPLL(*clauses*, *symbols* - *P*, *model* \cup {*P*=*value*})

P \leftarrow FIRST(*symbols*); *rest* \leftarrow REST(*symbols*)

return DPLL(*clauses*, *rest*, *model* \cup {*P*=*true*}) **or**

DPLL(*clauses*, *rest*, *model* \cup {*P*=*false*})

heuristics

the essence of DPLL is *guessing* a truth value for each proposition, and *backtracking* when a conflict is discovered

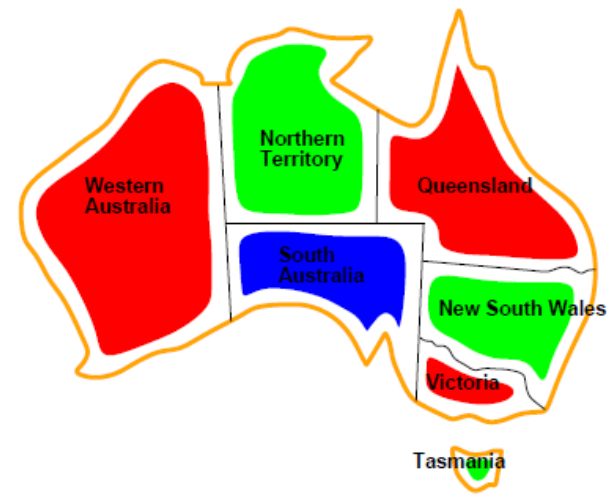
DPLL

- DPLL systematically explores the space of models (which can be slow)
- heuristics to speed up DPLL
 - we can bias the choice of which proposition to assign next
 - Unit Clause heuristic – given a partial assignment, if there is a clause where **all but one literal is False and the last is unknown (?)**, then add the appropriate truth value to the model
 - example: $\{\neg B, D, \neg D \vee C, \neg A \vee \neg B \vee E, D \vee \neg E, \neg C \vee E\}$
 - Pure Symbol heuristic - given a partial assignment, if proposition $X=?$ and X appears only as positive literal (X) in all unsatisfied clauses remaining, bind $X=T$
 - if it appears only as neg. lit. ($\neg X$) in all *unsatisfied* clauses remaining, bind $X=F$
 - it doesn't mean X *has* to have that truth value, only it *can* (if there is a model of the KB, then there is a model in which $X=T$)
 - important: apply these incrementally; as the model gets more vars bound:
 - ignore clauses that are satisfied by the partial model (have at least 1 var that is true)
 - mark off vars in a clause that evaluate to false; non-Unit clauses might *become* Unit clauses

Example of DPLL heuristics

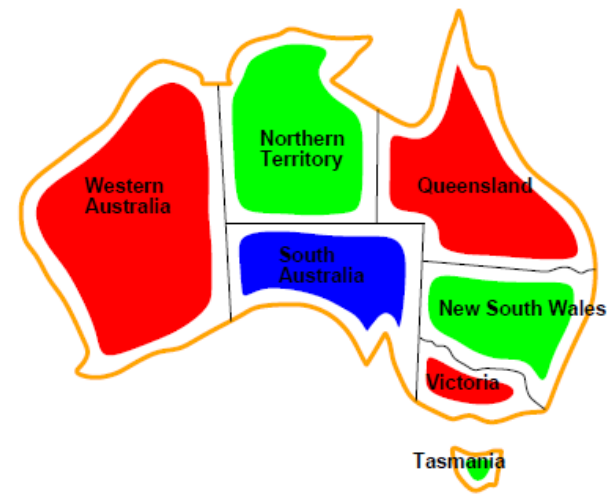
- $CNF = \{-B, D, \neg D \vee C, \neg A \vee \neg B \vee E, D \vee \neg E, \neg C \vee E\}$
- init: $m_0 = \{?, ?, ?, ?, ?\}$
- step 1: clause 1 is Unit, bind $B = F$: $m_1 = \{?, F, ?, ?, ?\}$
- step 2: clause 2 is Unit, bind $D = T$: $m_2 = \{?, F, ?, T, ?\}$
- step 3: are there any new unit clauses?
 - clauses 1, 2, 4, 5 are satisfied (at least 1 var is true in each), so these can be ignored
 - clause 3: $\neg D \vee C$ evaluates to “Fv?” in current model m_2 , so C must be true (since all vars are false except 1): $m_3 = \{?, F, T, T, ?\}$
- step 4: now clause 6 becomes Unit because $\neg C \vee E = \text{“Fv?”}$ in m_3 , so E must be true: $m_4 = \{?, F, T, T, T\}$
 - could halt, since this model now satisfies all clauses (2 solutions: $\{T, F, T, T, T\}$ or $\{F, F, T, T, T\}$)
- by the way, A only appears as a negative literal in all clauses, so the Pure Symbol heuristic would bind it to false: $m_5 = \{F, F, T, T, T\}$

Solving Problems via Satisfiability



- example: map-coloring
 - convert KB (slide 11) to CNF
 - there are 21 propositions (7 states X 3 colors)
 - clauses={WAR v WAG v WAB, ¬WAR v ¬WAB, ¬WAR v ¬NTR, ...} (100-200 CNF sentences)
 - DPLL(<?,?,?,?.....?>,clauses) returns a complete truth assignment
 - <WAR=T, WAG=F, WAB=F, NTR=F, NTG=T, NTB=F, SAR=F...>
 - 7 T's and 14 F's
 - the DPLL algorithm can be modified to return additional models
 - how many times does back-tracking occur?
 - when does the unit-clause heuristic get invoked?
 - how much back-tracking would there be without the UC or PS heuristics?
 - size of search space?

Solving Problems via Satisfiability



- using DPLL to find other solutions
 - find a coloring of the map where Queensland is green
 - $\text{DPLL}(\langle ?, ?, ?, ? \dots ? \rangle, \text{clauses} \cup \{QG\})$ returns
 - $\langle \text{WAR}=\text{F}, \text{WAG}=\text{T}, \text{WAB}=\text{F}, \text{NTR}=\text{T}, \text{NTG}=\text{G}, \text{NTB}=\text{F}, \text{SAR}=\text{F}, \text{SAG}=\text{F}, \text{SAB}=\text{T}, \text{QR}=\text{F}, \text{QG}=\text{T}, \text{QB}=\text{F} \dots \rangle$
- using DPLL to show something is entailed
 - show that if WA is red, then V has to be red: $\text{WAR} \rightarrow \text{VR}$
 - negate the sentence and add to clauses: $\neg(\text{WAR} \rightarrow \text{VR}) = \text{WAR} \wedge \neg \text{VR}$ (as CNF)
 - $\text{DPLL}(\langle ?, ?, ?, ? \dots ? \rangle, \text{clauses} \cup \{\text{WAR}, \neg \text{VR}\})$ returns *unsatisfiable*

DPLL

- many other problems can be solved by encoding them as Sat problems
 - **CSPs**
 - Sammy's sport shop, Wumpus world
 - planning (SatPlan), scheduling,
 - multi-agent coordination,...
 - vertex cover, knapsack,...
 - program verification (write a Boolean expression describing the steps in a piece of code, and an invariant or property it is supposed to maintain)

Complexity of Propositional Inference

- Cook's Theorem: Boolean SAT is NP-complete.
 - proof involves showing that you can describe or "encode" a Turing machine that simulates any non-det. computation in the form of a Boolean expression with at size at most a polynomial in the number of states, tape symbols, etc
- Hence, *complete* proof procedures can't be guaranteed to halt and return an answer in polynomial time (unless $P=NP$)
 - so we could wait a *long* time for a resolution proof to finish
 - however, restricted methods, like FC and BC can potentially run in poly time

WalkSAT - a stochastic approach to satisfiability

- not guaranteed to be complete, but it is fast and often effective at finding models of a set of clauses

function WALKSAT(*clauses*, *p*, *max_flips*) **returns** a satisfying model or *failure*

inputs: *clauses*, a set of clauses in propositional logic

p, the probability of choosing to do a “random walk” move, typically around 0.5

max_flips, number of flips allowed before giving up

model ← a random assignment of *true/false* to the symbols in *clauses*

for *i* = 1 **to** *max_flips* **do**

if *model* satisfies *clauses* **then return** *model*

clause ← a randomly selected clause from *clauses* that is false in *model*

with probability *p* **flip** the value in *model* of a randomly selected symbol from *clause*

else flip whichever symbol in *clause* maximizes the number of satisfied clauses

return *failure*

*this is what makes the search efficient;
kind of like the MinConflicts alg. for CSPs*

Is this set of clauses satisfiable?

How hard would it be to find a model?

$$\bar{a} \vee \bar{f} \vee g$$

$$\bar{a} \vee \bar{b} \vee \bar{h}$$

$$a \vee c$$

$$a \vee \bar{i} \vee \bar{l}$$

$$a \vee \bar{k} \vee \bar{j}$$

$$b \vee d$$

$$b \vee g \vee \bar{n}$$

$$b \vee \bar{f} \vee n \vee k$$

$$\bar{c} \vee k$$

$$\bar{c} \vee \bar{k} \vee \bar{i} \vee l$$

$$c \vee h \vee n \vee \bar{m}$$

$$c \vee l$$

$$d \vee \bar{k} \vee l$$

$$d \vee \bar{g} \vee l$$

$$\bar{g} \vee n \vee o$$

$$h \vee \bar{o} \vee \bar{j} \vee n$$

$$\bar{i} \vee j$$

$$\bar{d} \vee \bar{l} \vee \bar{m}$$

$$\bar{e} \vee m \vee \bar{n}$$

$$\bar{f} \vee h \vee i$$

here is a solution:

a = True

b = True

c = True

d = True

e = False

f = False

g = False

h = False

i = False

j = False

k = True

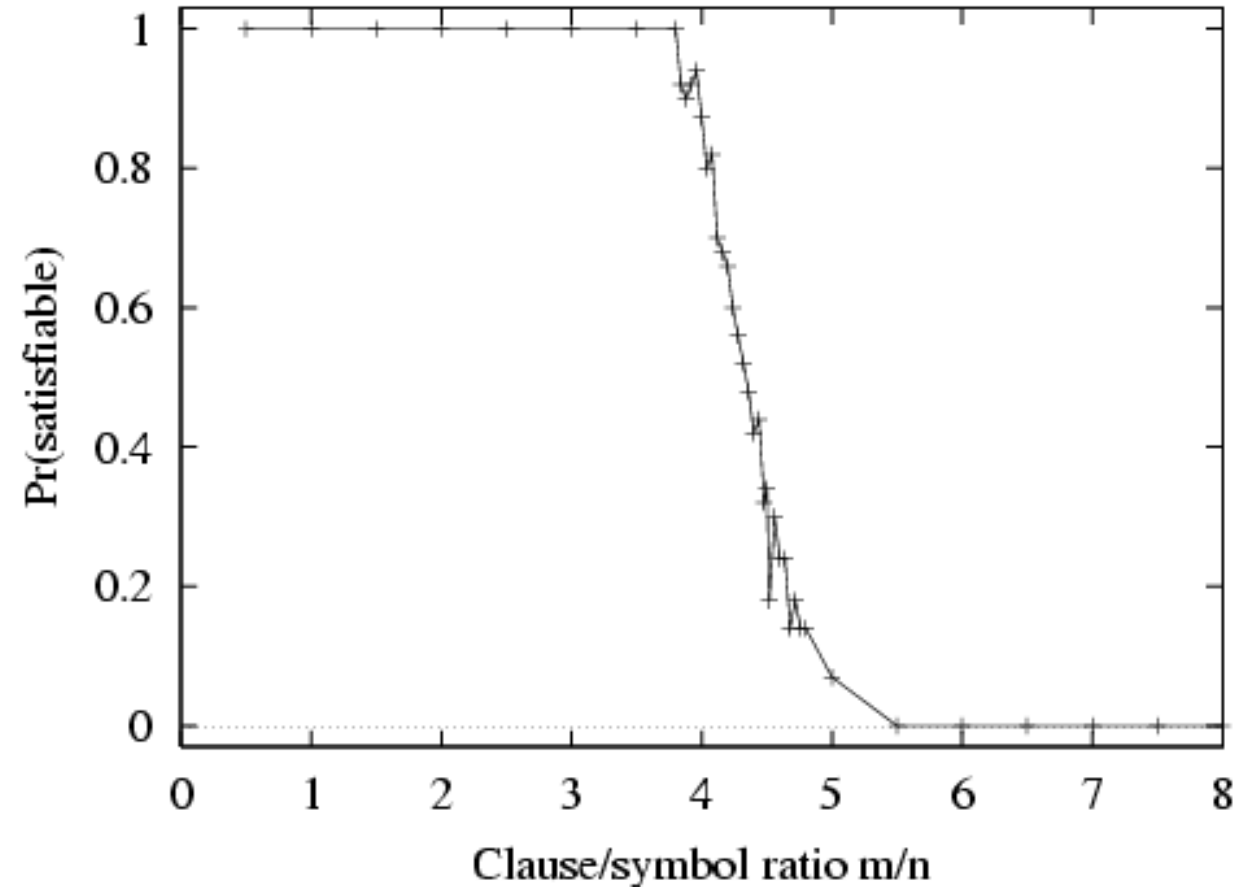
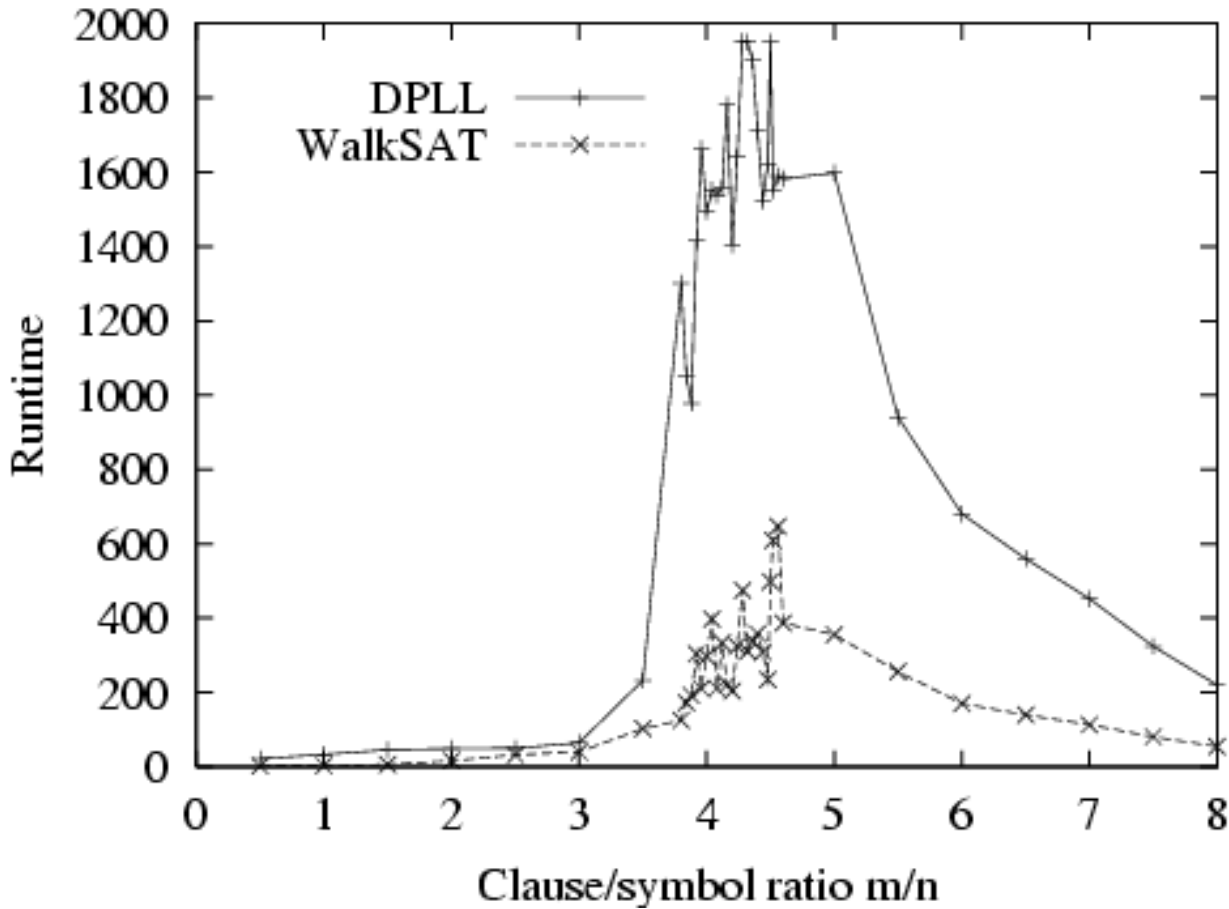
l = True

m = False

n = True

o = True

Hard Satisfiability Problems - The "Computational Cliff"



- experiments with *randomly generated* Sat problems (1000s of Boolean clauses)
- "computational cliff" at ~ 4.3 clauses per symbol

Results of the SAT 2009 Competition

