15-780: Graduate AI
Lecture 3. Logic, SAT, and CSPs

Geoff Gordon (this lecture)
Ziv Bar-Joseph
TAs Michael Benisch, Yang Gu

Admin

- *HW1 out today!*
  - *On course website*
  - *Due Wed 10/4*
- *Reminder: Matlab tutorial today*
  - *WeH 5409, 4:30PM*

Last episode, on *Grad AI*

Topics covered

- *C-space*
- *Ways of splitting up C-space*
  - *Visibility graph*
  - *Voronoi*
  - *Exact, approximate cell decomposition*
  - *Adaptive cells (quadtree, parti-game)*
- *Potential fields*
- *RRTs*

More on search

- *I defined an "admissible" heuristic as one that underestimates cost-to-goal*
- *Better definition: admissible heuristic is* ***optimistic*** *about future*
- *Covers rewards as well as costs*
- *Map still must have absorbing goals and no negative-cost (positive-reward) cycles*

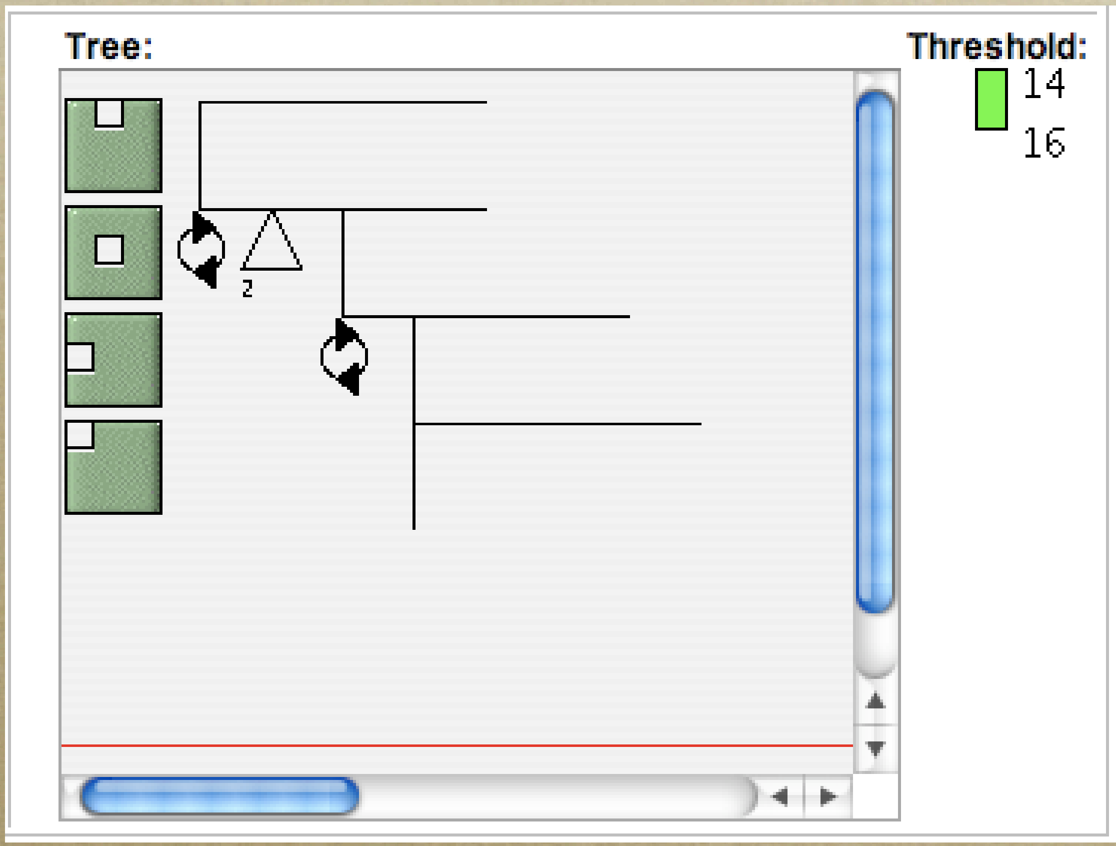
More on search

- *You might think from last lecture that search = path planning*
- *Other apps: logical problems (robotic grad student, HW1, today's lecture), planning (coming up soon)*

IDA*

- *Do a DFS of all nodes with f(node) < k*
- *If no solution, increment k and try again*
- *Just like DFID, except that instead of a depth bound, bounds f = g + h*

8/15 puzzle applet



http://www.cs.ualberta.ca/~aixplore/search/IDA/Applet/SearchApplet.html

Project ideas

Poker

Poker

- *Have code which learns a provably near-minimax RI Hold'Em strategy in 40 min*
- *Code is easily parallelizable and works on abstractions of larger games*
- *Can we beat world's best computer Texas Hold'Em players?*

More on Poker

- *Minimax strategy for heads-up poker = solving linear program*
- *1-card hands, 13-card deck: 52 vars, instantaneous*
- *RI Hold'Em: ~1,000,000 vars, 2 weeks / 30GB (exact sol) on CPLEX, 40 min / 1.5GB (approx sol) w/ our algorithm*
- *TX Hold'Em: ??? (up to 10^{17} vars or so)*

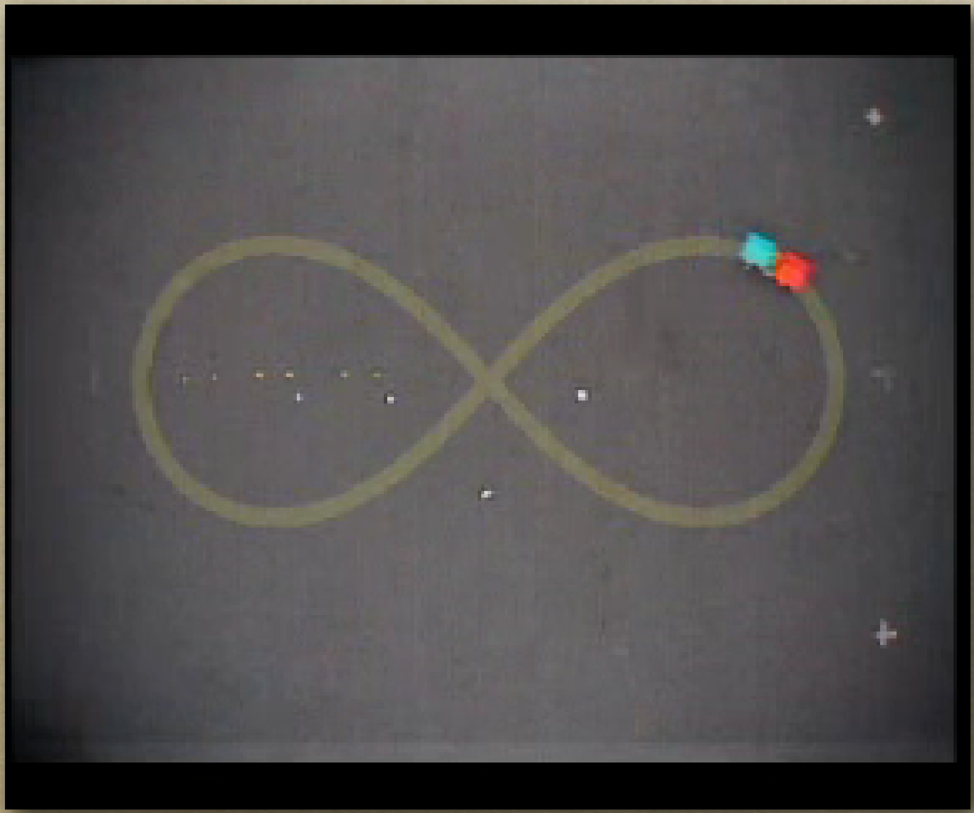
Scrabble™

- *Can buy a hand-tweaked, very good computer Scrabble player for $30 or so*
- *Can we learn to beat it?*
- *Easy: enumerate legal moves*
- *Hard: which should we take?*

Learning models for control

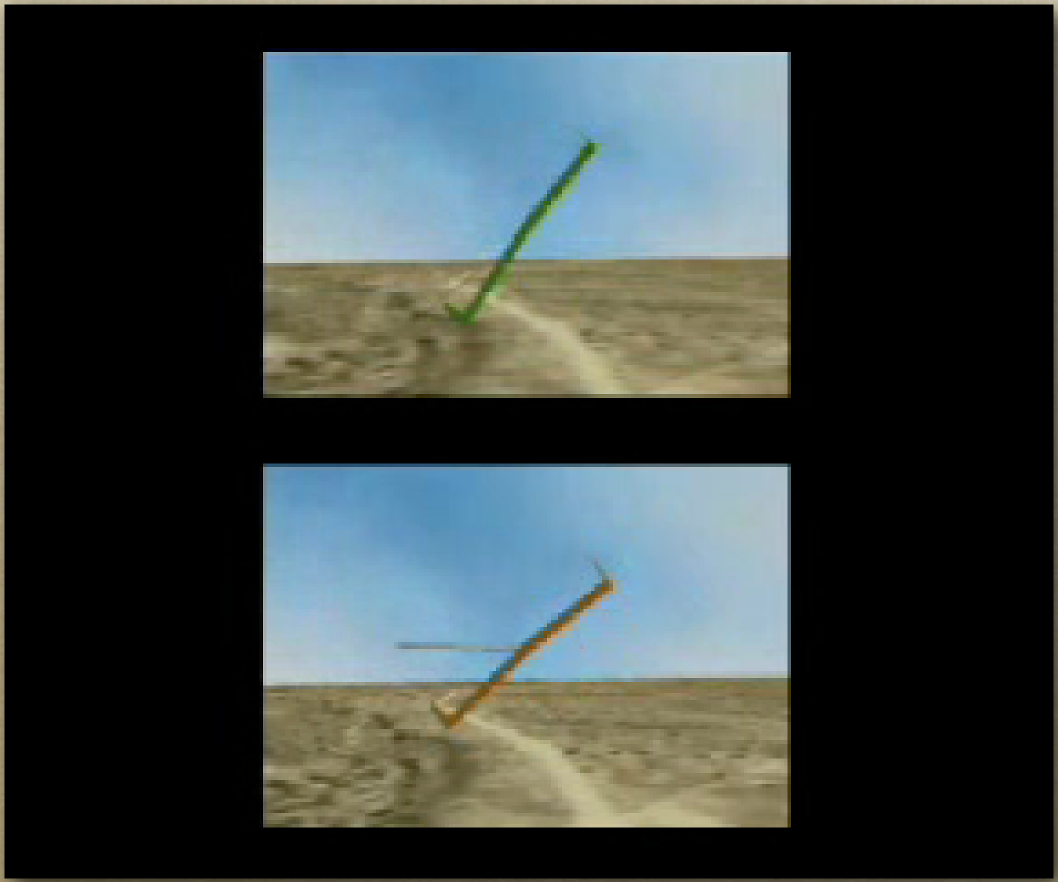
- *Most of this course, we'll assume we have a good model of the world when we're trying to plan*
- *Usually not true in practice—must learn it*
- *Project: learn a model for an interesting system, write a planner for learned model, make planner work on original system*

Learning models for control



- *R/C car*

Learning models for control



- *Model airplane*

Logic

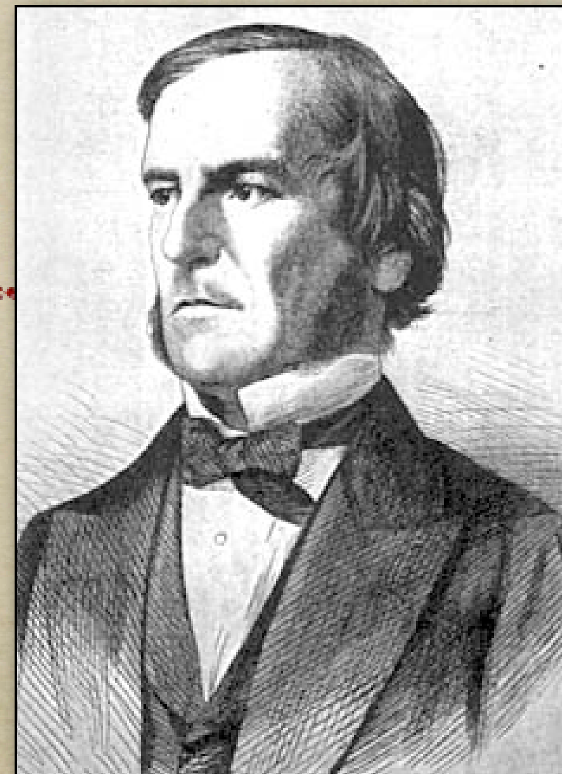
Why logic?

- *Problem-solving: want to find solutions for problems like 8-puzzle*
- *Reasoning: intelligent agents need knowledge about world to reach good decisions, want them to figure out consequences of their knowledge*
- *Also, logical inference is a special case of probabilistic inference (Part II)*

Propositional logic

- *Constants & variables: T or F*
- *Connectives:* ∧, ∨, ¬
  - *Can get by w/ just NAND*
  - *Sometimes also add others:*

    ⊕, ⇒, ⇔, …
- *Terminology: variable or constant with or w/o negation =* ***literal***
- *Whole thing =* ***formula*** *or* ***sentence***



George Boole
1815–1864

Propositional logic

- *Precedence:*
  - *unary binds tighter than binary*
  - $\wedge$ *has higher precedence than* $\vee$
  - *parens:* $\neg(a \wedge b)$ *vs.* $\neg a \wedge b$
- *Quiz:*

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

Truth tables

x	y	$x \wedge y$
T	T	T
T	F	F
F	T	F
F	F	F

x	y	$x \vee y$
T	T	T
T	F	T
F	T	T
F	F	F

A note on implication

- *$(a \Rightarrow b)$ is logically equivalent to $(\neg a \vee b)$*
- *If a is True, b must be True too*
- *If a False, no requirement on b*
- *E.g., "if I go to the movie I will have popcorn": if no movie, may or may not have popcorn*

a	b	$a \Rightarrow b$
T	*T*	*T*
T	*F*	*F*
F	*T*	*T*
F	*F*	*T*

Truth tables of all sizes

x	$\neg x$
T	F
T	T

x	y	z	$(x \vee y) \Rightarrow z$
T	T	T	
T	T	F	
T	F	T	
T	F	F	
F	T	T	
F	T	F	
F	F	T	
F	F	F	

Expressive variable names

- *Rather than names like a, b, x, y, we may use names like "rains" or "happy(John)"*
- *For now, "happy(John)" is just a string with no internal structure*
  - *propositional logic doesn't know about John, just about variable happy(John)*
- *Later we will assign semantics*

What can we do with a sentence?

- *Assign values to variables and evaluate it*
- *Ask whether it's true for zero, some, or all assignments to variables*
- *Simplify it to get another, equivalent formula (which might have side effect of helping us test its value)*

Models

- *An assignment to all variables is called a* ***model****: e.g.*

$$M = (a: T, b: T, c: F)$$

- *A sentence has a truth value in each model*
- *Finding this truth value takes linear time*
- *Ex:* $((a \wedge \neg b) \vee c)$ *in M is ???*

Definitions

- *A sentence is **satisfiable** if it is True in some model*
- *If not satisfiable, it is a **contradiction***
- *A sentence is **valid** if it is True in every model (a valid sentence is a **tautology**)*
- *A is a contradiction* $\Leftrightarrow$ $\neg$*A is valid*

SAT

Important search problem: SAT

- *SAT is the problem of determining whether a given propositional logic sentence is satisfiable*
- *SAT is a search problem: search nodes are (full or partial) models, neighbors differ in assignment for a single variable*

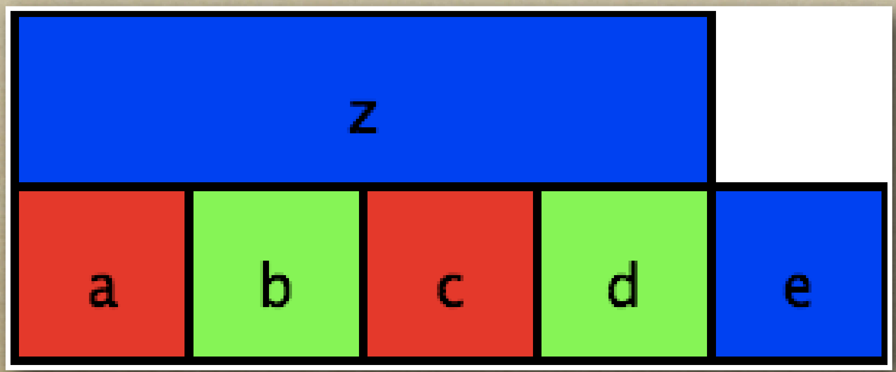
Why is SAT important?

- *Any ordinary* search problem reduces* to SAT*
- *So a good SAT solver is a good AI building block*

Ordinary search problem

- *OS problem = search graph + start node + solution test function*
  - *search graph = next-neighbor function*
  - *problem: decide whether a solution node is reachable from start*
- *Limit: efficient (poly-time) functions*
- *Limit: polynomial depth, branching factor (exponentially many vertices)*

Example (ordinary) search problem

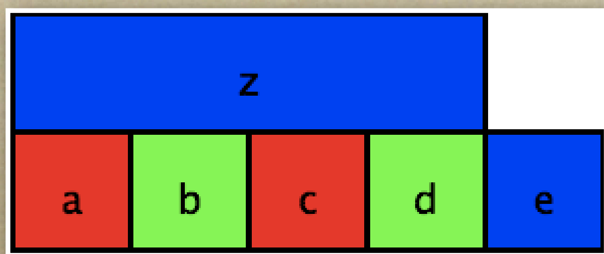


- *3-coloring: can we color a map using only 3 colors in a way that keeps neighboring regions from being the same color?*

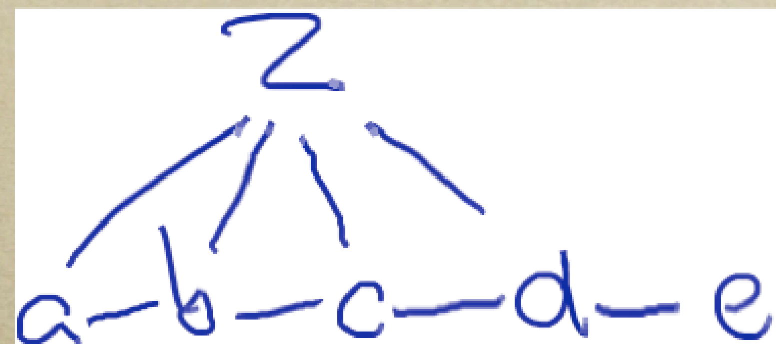
Reduction

- *Loosely, "problem A reduces to problem B" means that if we can solve B then we can solve A*
- *More formally, A, B are decision problems (instances $\mapsto$ truth values)*
- $\exists$ *a poly-time function f so that: given an instance a of A, f(a) is an instance of B, and* $A(a) = B(f(a))$

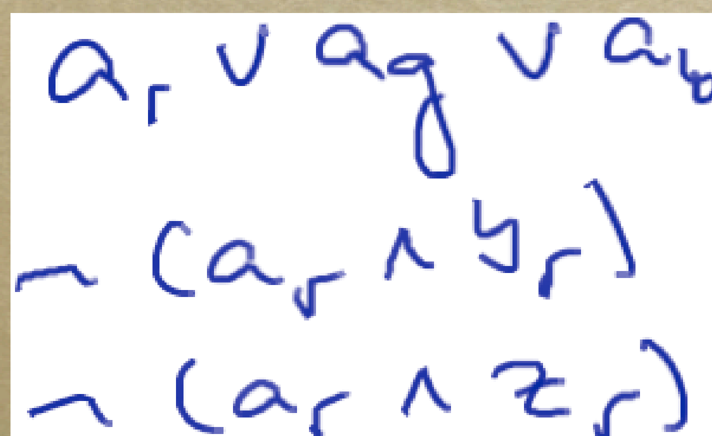
Example reduction



=



$\Longrightarrow$



Search and reduction

- *Ordinary search problems reduce to SAT and (usually) vice versa*
- *Equivalently, SAT is as hard (in theory at least) as (most) other search problems*
- *Proven by S. A. Cook in 1971*
  - *showed how to simulate poly-size-memory computer w/ (very complicated, but still poly-size) SAT problem*

Cost of reduction

- *Complexity theorists often ignore little things like constant factors (or even polynomial factors!)*
- *So, is it a good idea to reduce your search problem to SAT?*
- *Answer: sometimes…*

Cost of reduction

- *SAT is well studied ⇒ fast solvers*
- *So, if there is an efficient reduction, ability to use fast SAT solvers can be a win*
  - *e.g., 3-coloring*
  - *another example later (SATplan)*
- *Other times, cost of reduction is too high*
  - *usu. because instance gets bigger*
  - *will also see example later (MILP)*

Working with formulas

Simplifying formulas

- *Searching for a model might get a lot easier if we simplify the formula first*
- *Best case: could prove a sentence valid or contradictory without testing any models by simplifying until we get just T/F*
- *Catch: in general, about as hard as SAT*
  - $[A \in SAT] = \neg[\text{is } (\neg A) \text{ valid}]$

Equivalence

- *Two sentences are* ***equivalent****, $A \equiv B$, if they have same truth value in every model*
  - *(rains ⇒ pours) ≡ (¬rains ∨ pours)*
  - *reflexive, transitive, commutative*
- *Simplifying = searching for a simpler*, equivalent formula*

Simplification as search

- *Search node = formula*
- *Neighborhood = equivalence rules*
- *Simplify formula by finding a sequence*

  $A \equiv B \equiv C \equiv \ldots \equiv Z$

  where Z is "simple"
- *Equivalently, a path in search graph from A to goal node*

Example

- *"simple" = "literally True or False"*
- *Prove a formula valid or contradictory by showing a sequence*

$$A \equiv B \equiv C \equiv \ldots \equiv True$$

or

$$A \equiv B \equiv C \equiv \ldots \equiv False$$

Simplification as search

- *Contrast w/ SAT search*
  - *search node = partial model*
- *Or hybrid*
  - *search node = partial model + formula*

Transformation rules

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge$$
$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee$$
$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge$$
$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee$$
$$\neg(\neg\alpha) \equiv \alpha \quad \text{double-negation elimination}$$

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee$$
$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge$$

α, β, γ are arbitrary formulas

More rules

$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$$ contraposition

$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$$ implication elimination

$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$$ biconditional elimination

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$$ de Morgan

$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$$ de Morgan

α, β are arbitrary formulas

Example

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

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

Normal Forms

CNF

- *To make their lives easier, programs often require their inputs in **conjunctive normal form** (CNF)*
- *CNF = conjunction of disjunctions of literals*
- *Each disjunct called a **clause***

CNF cont'd

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

- *Often used as a form for storage of knowledge databases*
- *Can add new clauses as we find them out*
- *Formula implies each individual clause*

Another normal form: DNF

- *DNF = disjunctive normal form = disjunction of conjunctions of literals*
- *Doesn't compose the way CNF does: can't just add new conjuncts w/o changing meaning of KB*
- *Example:*

$$(rains \lor \neg pours) \land fishing$$
$$\equiv$$
$$(rains \land fishing) \lor (\neg pours \land fishing)$$

Transforming to normal form

- *Naive algorithm:*
  - *replace all connectives with* $\wedge \vee \neg$
  - *move all negations inward using De Morgan's laws and double-negation*
  - *repeatedly distribute over* $\wedge$ *over* $\vee$ *for DNF (*$\vee$ *over* $\wedge$ *for CNF)*

Example

- *Put the following formula in CNF*
- *Start to try DNF*

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

Discussion

- *Problem with naive algorithm: it's exponential! (Both space and time, as well as size of result.)*
- *Each use of distributivity can almost double the size of a subformula*

A smarter transformation

- *Can we avoid exponential blowup in CNF?*
- *Yes, if we're willing to introduce new variables*
- *D. Plaisted and S. Greenbaum. A structure-preserving clause form translation. Journal of Symbolic Computation, 2:293—304, 1986.*

Inference rules

Inference rules

- *De Morgan's laws are nice, but not really enough for all the proofs we need*
- *Need a way to generate additional consequences of our formula*

Modus ponens

$$\frac{(a \wedge b \wedge c \Rightarrow d) \quad a \quad b \quad c}{d}$$

- *Probably most famous inference rule: all men are mortal, Socrates is a man, therefore Socrates is mortal*
- *Quantifier-free version:*

$man(Socrates) \wedge$

$(man(Socrates) \Rightarrow mortal(Socrates))$

Inference example

- *When it rains, it pours*
- *It's never the case that it's pouring and we're not rusted*
- *When we rust, we break*
- *It's raining*

$$(rains \Rightarrow pours) \wedge$$
$$\neg(pours \wedge \neg rusted) \wedge$$
$$(rusted \Rightarrow broken) \wedge$$
$$rains$$

Are we broken?

$$(rains \Rightarrow pours) \land$$
$$\neg(pours \land \neg rusted) \land$$
$$(rusted \Rightarrow broken) \land$$
$$rains$$

Getting organized

- *Can we organize our search for a proof better?*
- *Mechanical system for deciding which transformations to apply*

Theorem provers

Typical theorem prover

- ***Knowledge Base*** *(KB): a set of assertions about the world*
    - *we "know" conjunction of all sentences in KB*
- *Query sentence B: want to know, B or ¬B?*

Typical theorem prover

- *Apply **inference rules** to sets of sentences (subsets of KB) to generate new assertions*
- *Add (conjoin) new assertions to KB*
- *Result is a larger KB which is (hopefully) equivalent to original KB (more later)*
- ***Search control heuristics** decide which consequence to add next*

When are we done?

- *One approach: done if we add B to our KB*
- *More common: proof by contradiction*
- *Add ¬B to KB, finish when we add False*

When can we give up?

- *For propositional logic, there are finitely many possible conclusions from a finite set of assertions*
- *So, if we run out of things to try, we can conclude B does not follow from KB*

Backtracking

- *Never **have** to backtrack: inference rules only add valid consequences to KB*
- *But irrelevant assertions can pile up*
  - *might want to get rid of old conclusions*
  - *e.g., DFS for results whose proof adds ≤ k new assertions to KB*
  - *then increase k (DFID on proof size)*

Inference rules

- *What inference rules should we use in an automated theorem prover?*
- *Requirement: if rule can take sentence S and produce sentence A, want A to be true whenever S is*
  - *S might be conjunction of several elements of KB*
- *This is called* ***entailment***

Entailment

- *Sentence A* ***entails*** *sentence B, A $\models$ B, if B is True in every model where A is*
    - *same as saying that (A $\Rightarrow$ B) is valid*
- *If (subset of KB $\models$ A) then KB $\models$ A (**monotonicity**)*
- *If KB $\models$ A then KB $\equiv$ (KB $\wedge$ A)*
    - *so it's safe to add A to KB*

Special cases

- *If A* $\models$ *False, then A must be a contradiction*
- *A* $\models$ *True for any A*

Proof using entailment

- *Suppose we have an inference rule that generates entailed sentences*
- *Definition: **proof tree***
  - *Leaves: sentences from KB*
  - *Children* $\vDash$ *parent*
  - *Root = consequence*
- *If ∃ a proof tree with root False, KB is contradictory*

Proof tree

KB

rains ⇒ pours

pours ∧ outside ⇒ rusty

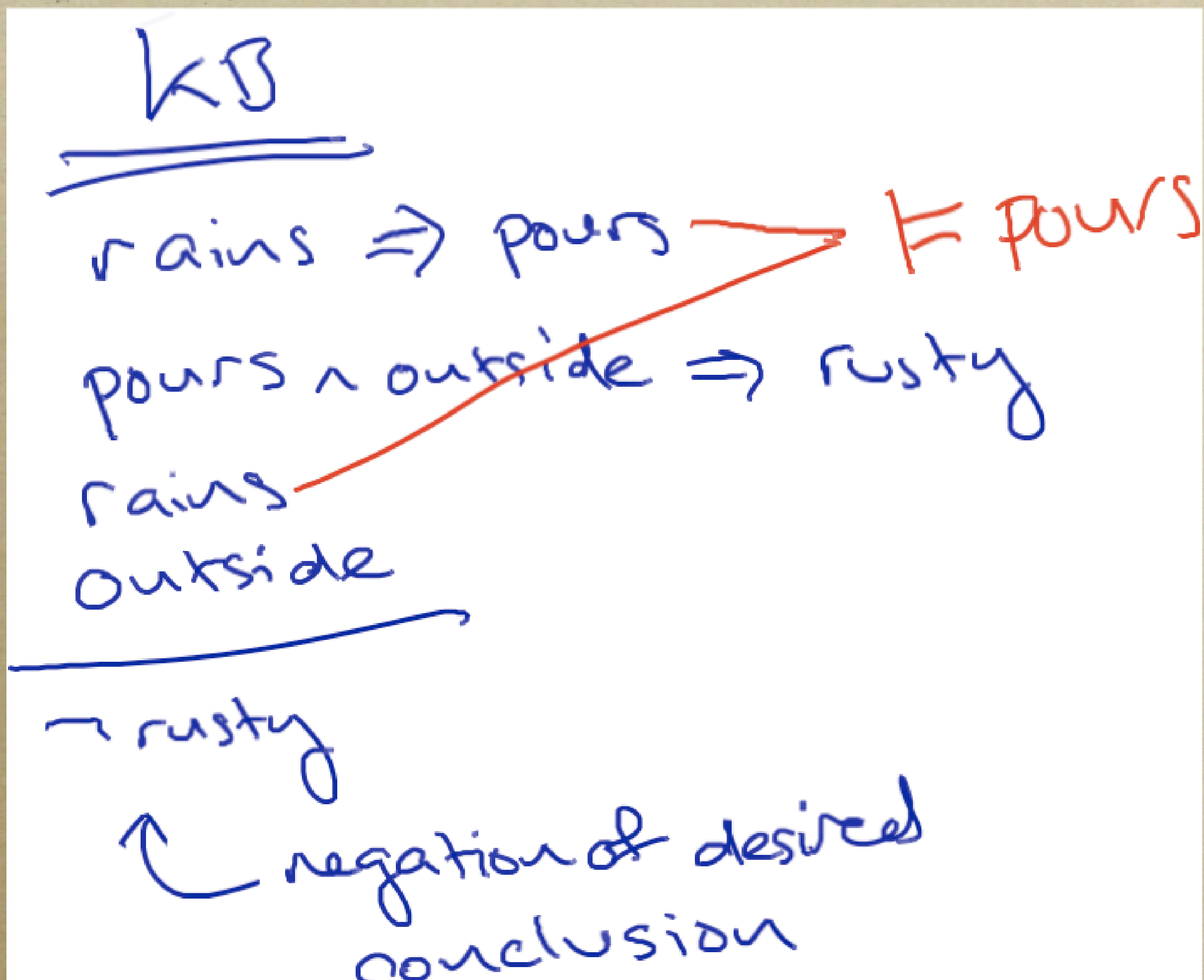
rains

outside

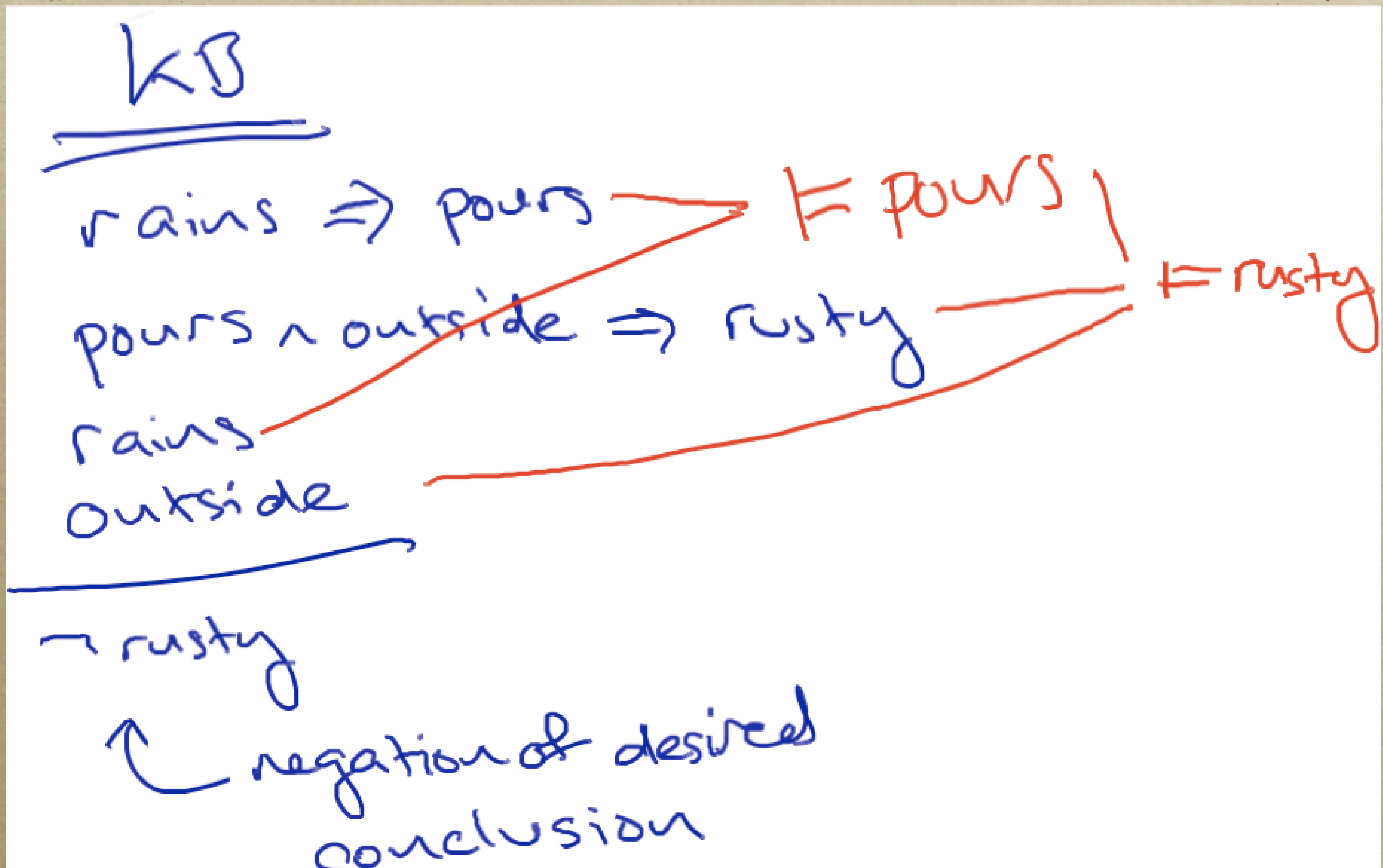
¬ rusty

↑ negation of desired conclusion

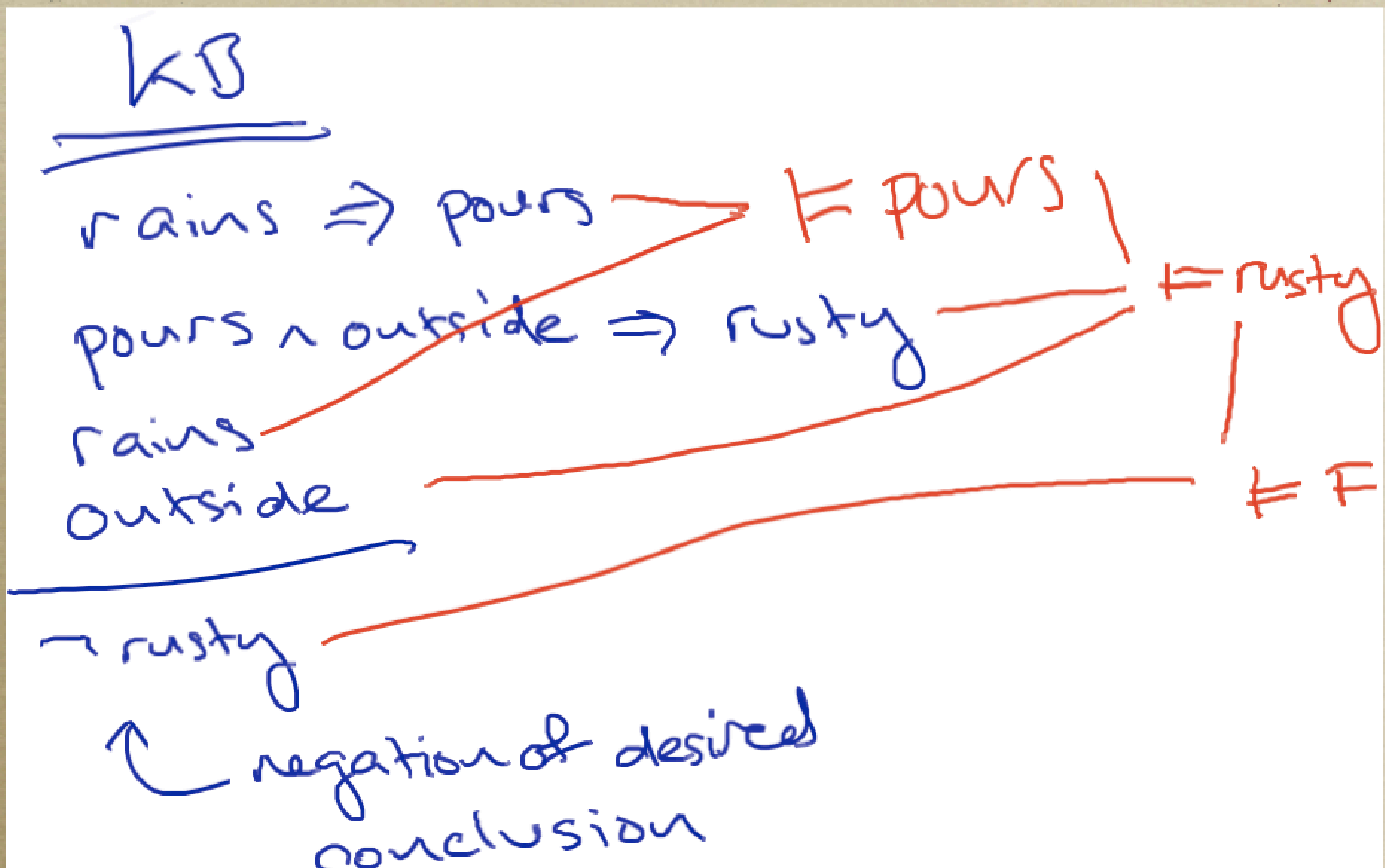
Proof tree


KB
rains ⇒ pours
pours ∧ outside ⇒ rusty
rains
outside
¬ rusty
⊨ pours
negation of desired conclusion

Proof tree


KB
rains ⇒ pours
⊨ pours
pours ∧ outside ⇒ rusty
⊨ rusty
rains
outside
¬ rusty
negation of desired conclusion

Proof tree


KB
rains ⇒ pours
pours ∧ outside ⇒ rusty
rains
outside
¬ rusty
⊨ pours
⊨ rusty
⊨ F
negation of desired conclusion

Entailment v. equivalence proofs

- *A proof with a proof tree and* $\models$ *is the same as one with a sequence of* $\equiv$ *relations*
- *Proof tree on KB with nodes x, y, z, … yields sequence*

  $$KB \equiv KB \wedge x \equiv KB \wedge x \wedge y \equiv \ldots$$

  if x, y, z, … topological sort of proof tree

Inference rules

$$\frac{(a \wedge b \wedge c \Rightarrow d) \quad a \quad b \quad c}{d}$$

- *Above proof tree used* ***modus ponens***
- *Already saw this rule above*

Another inference rule

$$\frac{(a \Rightarrow b) \quad \neg b}{\neg a}$$

- ***Modus tollens***
- *If it's raining the grass is wet; the grass is not wet, so it's not raining*

One more…

$$\frac{(a \vee b \vee c) \quad (\neg c \vee d \vee e)}{a \vee b \vee d \vee e}$$

- ***Resolution***
- *Not as commonly known as modus ponens / tollens*

Resolution

- *Combines two sentences that contain a literal and its negation*
- *Modus ponens / tollens are special cases*
- *Modus tollens:*

  $(\neg raining \lor grass\text{-}wet) \land \neg grass\text{-}wet \models \neg raining$

Resolution

$$\frac{(a \lor b \lor c) \quad (\neg c \lor d \lor e)}{a \lor b \lor d \lor e}$$

- *Simple proof by case analysis*
- *Consider separately cases where we assign c: True and c: False*

Resolution

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

- *Case c: True*

$$(a \lor b \lor T) \land (F \lor d \lor e)$$

$$= (T) \land (d \lor e)$$

$$= (d \lor e)$$

Resolution

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

- *Case c: False*

$$(a \lor b \lor F) \land (T \lor d \lor e)$$

$$= (a \lor b) \land (T)$$

$$= (a \lor b)$$

Resolution

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

- *Since c must be True or False, conclude*

  $$(d \lor e) \lor (a \lor b)$$

  as desired