CS 161 Intro. To Artificial Intelligence

Week 5, Discussion 1A

Thanks again previous TAs

Inference

What's **inference**?

- Given a KB Δ and β, we want to derive β from Δ with algorithm i
- Denote as: $KB \vdash_i \alpha$

  Does KB Δ entail α: Δ⊨α

Inference methods:

- Proof by enumeration – **Model Checking** $M(\Delta) \subseteq M(\alpha)$
  - List all the models where Δ is True, check whether β is also True
  - E.g. Use truth table
- **Proof by refutation (resolution)** or other rules $$\frac{\alpha,\ \alpha \Rightarrow \beta}{\beta}$$
  - Use resolution rule
  - Often use proof by contradiction
- Search - SAT solver → state-of-art
- Converting sentences to tractable forms (NNF circuits) → state-of-art

Inference Methods

Proof by enumeration - Model Checking:

- Enumerate all possible worlds (models), check one by one
  - E.g. Truth table enumeration
- Sound and complete
- $O(2^n)$ for n symbols

$\Delta : \{A, A \vee B \rightarrow C\}$

$\alpha : c$

Determine if $\Delta \models \alpha$

A	B	C	A	A∧B⇒ C	Δ	α
T	T	T				
T	T	F				
T	F	T				
T	F	F				
F	T	T				
F	T	F				
F	F	T				
F	F	F				

Inference Methods

Proof by enumeration - Model Checking:

- Enumerate all possible worlds (models), check one by one
  - E.g. Truth table enumeration
- Sound and complete
- $O(2^n)$ for n symbols

$\Delta : \{A, A \vee B \rightarrow C\}$

$\alpha : c$

Determine if $\Delta \models \alpha$

$\Delta \models \alpha$ iff $M(\Delta) \subseteq M(\alpha)$

- Visible in the table!

A	B	C	A	A∧B⇒ C	Δ	α
T	T	T	T	T	T	T
T	T	F	T	F	F	F
T	F	T	T	T	T	T
T	F	F	T	F	F	F
F	T	T	F	T	F	T
F	T	F	F	F	F	F
F	F	T	F	T	F	T
F	F	F	F	T	F	F

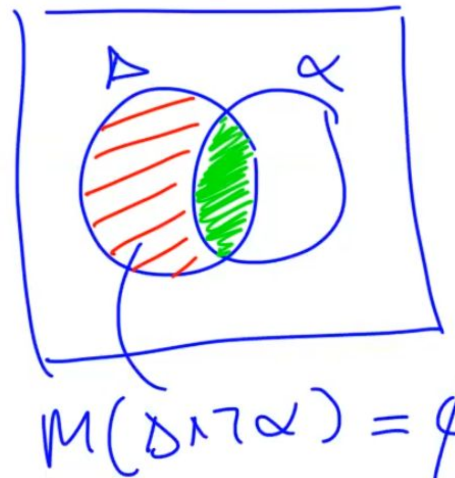
Inference Methods

Proof by refutation (resolution):

How do we determine whether $\Delta \models \alpha$?
Proof by refutation: $\Delta \models \alpha$ if and only if the sentence $(\Delta \wedge \neg\alpha)$ is unsatisfiable.

How do we determine whether $(\Delta \wedge \neg\alpha)$ is unsatisfiable?
Proof by Resolution (a.k.a. a resolution-based algorithm): Use the resolution inference rule. This algorithm is sound and complete. It applies to any kind of Δ and α.

$M(\Delta) \subseteq M(\alpha)$



$M(\Delta \wedge \neg\alpha) = \phi$

- Typically require translation of sentences into normal forms
- Sound and complete
- Still $O(2^n)$ in worst case

Inference Rules

- Modus Ponen: $$\frac{\alpha, \alpha \rightarrow \beta}{\therefore \beta}$$
    - Example: $\Delta = \{A, B, B \vee C, B \rightarrow D\}$
- And-Elimination $$\frac{\alpha \wedge \beta}{\therefore \alpha}$$
- Resolution $$\frac{\alpha \vee \beta, \neg\beta \vee \delta}{\therefore \alpha \vee \delta} \longleftarrow (\alpha \vee \beta) \wedge (\neg\beta \vee \delta)$$

OR introduction

$$\frac{\alpha, \beta}{\alpha \vee \beta}$$

AND introduction

$$\frac{\alpha, \beta}{\alpha \wedge \beta}$$

Inference

Properties:

- **Soundness (correctness)**:
  - Is this inference rule correct in all cases?
  - Rule i is **sound** if whenever we can derive α from KB with i (KB $\vdash_i$ α), we know KB |= α

    If Δ⊢α then Δ⇒α
- **Completeness**:
  - Can this inference rule determine entailment for <u>any</u> $\Delta \models \beta$?
  - Rule i is **complete** if whenever KB |= α, we can derive α from KB with i (KB $\vdash_i$ α)

    If Δ⇒α then Δ⊢α

    Modus ponens is NOT complete!

$$\frac{\alpha, \alpha \rightarrow \beta}{\beta}$$

$$\Delta = A \vee B$$
$$\alpha = A \vee B \vee C$$

Inference

What's the difference between: implication($\Rightarrow$), entailment($|=$), and inference($\vdash$)

- Implication A $\Rightarrow$ B:
  - Statement in the object language
  - If make it a statement in the meta-language (asserting that the implication is provable) then you get something equivalent to $\vdash$
- Entailment A $|=$ B:
  - Statement in the meta language
  - Asserts that every B holds in every model of A (semantic consequence)
- Inference A $\vdash$ B:
  - Statement in the meta language
  - Asserts the existence of a proof of B from A (syntactic consequence)

If the logical system is sound, then we can conclude $A \vDash B$ from $A \vdash B$.

If the logical system is complete, then we can conclude $A \vdash B$ from $A \vDash B$.

Resolution

Two CLAUSES

- disjunctions of literals

A literal in one and it’s negation in the other.

Derive a new clause, the “resolvent”

$$\frac{\alpha \lor \beta \,, \quad \neg\beta \lor \gamma}{\alpha \lor \gamma}$$

An equivalent and comparable derivation rule

using implication - very intuitive.

$$\frac{\neg\alpha \Rightarrow \beta \,, \quad \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

Remember you can have any number of other

literals in the respective clauses

Resolution

Resolution is sound but not complete for propositional logic.

But it is **Refutation Complete for CNF.**

- Resolution **will** derive a contradiction (FALSE)

  if CNF is <u>unsatisfiable</u>

$\Delta = A \cup B$

$\alpha = A \cup B \vee C$

$\Delta \nvdash_{\text{Resolutn}} \alpha$

$\Delta \models \alpha$

How to **proof KB |=α with resolution + refutation**?

- As introduced before, **it is equivalent to proof KB ∧ ¬α <u>unsatisfiable</u>**
- **First turn KB into CNF (may explode in size)**
- **Then apply resolution rule until an empty clause appears (false/unsat).**
  - **If no more resolution can be applied and no empty clause appear (no contradiction is found), then KB ∧ ¬α is satisfiable**
- **UNIT resolution (one of the clauses is unit/one literal) is linear time**
  - **Not refutation complete though**

Resolution - Example

$$\Delta : (A \vee \neg B) \rightarrow C$$
$$C \rightarrow (D \vee \neg E)$$
$$E \vee D$$

$$\alpha : A \rightarrow D$$

Determine if $\Delta \models \alpha$

Remember: For KB which is a set of sentences $\{\alpha 1, \alpha 2, \ldots\}$, we can consider the whole it as a single long sentence $\alpha 1 \wedge \alpha 2 \wedge \cdots$

Resolution - Example

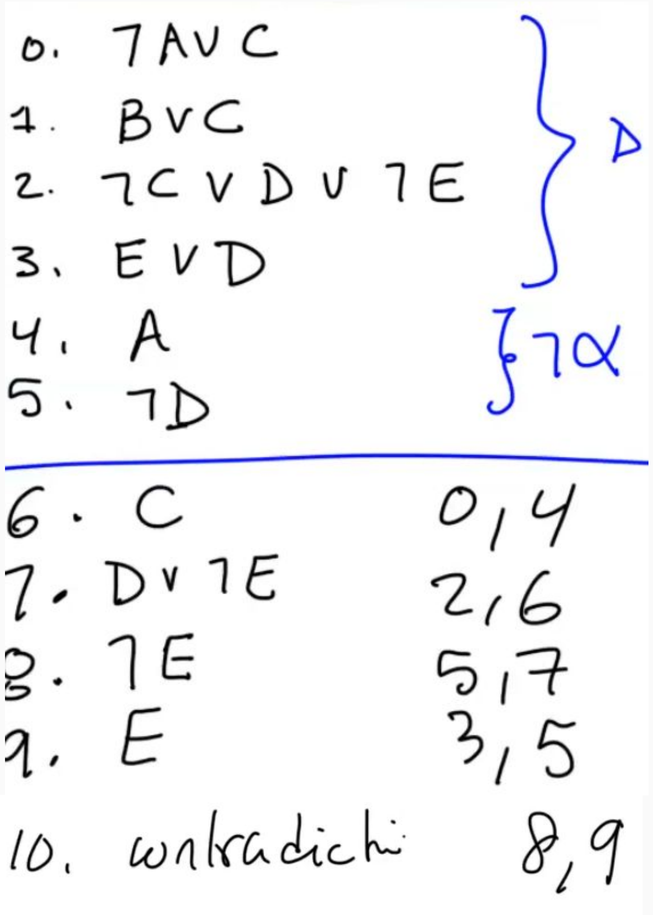
$\Delta : (A \vee \neg B) \rightarrow C$

$C \rightarrow (D \vee \neg E)$

$E \vee D$

$\alpha : A \rightarrow D$

Determine if $\Delta \models \alpha$



$$\left.\begin{array}{ll} 0. & \neg A \vee C \\ 1. & B \vee C \\ 2. & \neg C \vee D \vee \neg E \\ 3. & E \vee D \end{array}\right\} \Delta$$

$$\left.\begin{array}{ll} 4. & A \\ 5. & \neg D \end{array}\right\} \neg\alpha$$

6.	C	0, 4
7.	$D \vee \neg E$	2, 6
8.	$\neg E$	5, 7
9.	E	3, 5
10.	contradiction	8, 9

UNSAT!

Resolution - Exercise

$$\Delta : P \vee Q, P \rightarrow R, Q \rightarrow R$$

$$\alpha : R$$

Determine if $\Delta \models \alpha$

Resolution - Exercise

$$\Delta : P \vee Q, P \rightarrow R, Q \rightarrow R$$

$$\alpha : R$$

Determine if $\Delta \models \alpha$

1.	P ∨ Q	KB
2.	¬P ∨ R	
3.	¬Q ∨ R	
4.	¬R	Not Alpha
5.	¬Q	3,4
6.	¬P	2,4
7.	P	1,5
8.	Contradiction	6,7

CNF Conversion

1. Only AND, OR, NOT
   a. $\alpha \Rightarrow \beta \rightarrow \neg \alpha \wedge \beta$
   b. $\alpha \Leftrightarrow \beta \rightarrow (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
2. deMorgan's Laws
   a. $\neg(\alpha \wedge \beta) \rightarrow \neg \alpha \vee \neg \beta$
   b. $\neg(\alpha \vee \beta) \rightarrow \neg \alpha \wedge \neg \beta$
3. Distribute OR over AND
   a. $(\alpha \wedge \beta) \vee \gamma \rightarrow (\alpha \wedge \gamma) \wedge (\beta \vee \gamma)$

SAT Solver (SEARCH!)

Key idea: reducing queries to SAT (a special case of CSP)

CNF: $\Delta = \underbrace{(A \lor B \lor \neg C)}_{\text{Clause 1}} \land \underbrace{(\neg A \lor C)}_{\text{Clause 2}} \land \underbrace{(A \lor C \lor \neg D)}_{\text{Clause 3}}$

Consider each clause in a CNF as constraint

Say if we assign: A ← F, B ← T, C← F, D ← F ➔ this is a <u>world</u> w

Clause 1: F ∨ T ∨ T = T

Clause 2: T ∨ F = T

Clause 3: F ∨ F ∨ T = T

Thus the world w ⊨ Δ, it is proven that Δ is satisfiable.

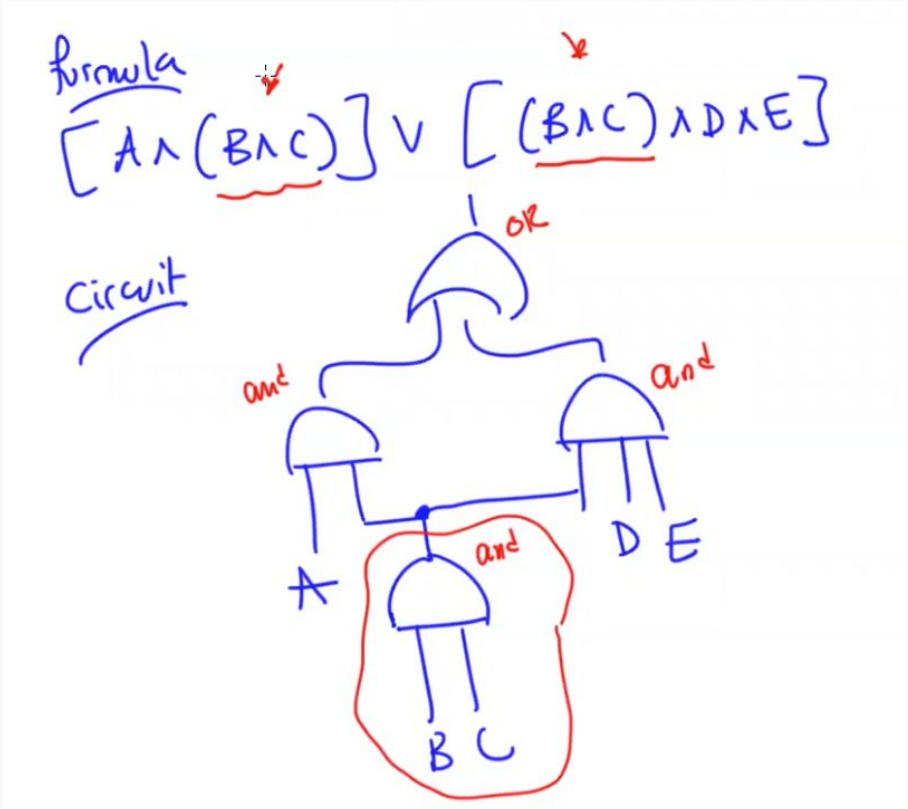
SAT Solver

How do we solve SAT?

- Backtrack search: DFS + detecting failures early (arc consistency or forward checking)
  - Called **DPLL** (initials of four authors) in the context of SAT
    - Uses unit resolution as an aid to detect failures early and increase efficiency (do resolution in linear time)
    - Use special variable/value ordering
    - Modern SAT solver can “learn” clauses → empower unit resolution
- Local search:
  - Can be **very fast**, but **NOT complete**
  - Steps:
    - Guess a truth assignment (world w)
    - Check whether $w \models \Delta$: If yes → done. If no → find another w to try
  - Method: Pick a clause that is violated (unsatisfied), then flip a variable that either maximize # of satisfied clauses or minimize # of unsatisfied clauses → can go back and forth, thus not complete

Compile Sentences into Tractable Circuits

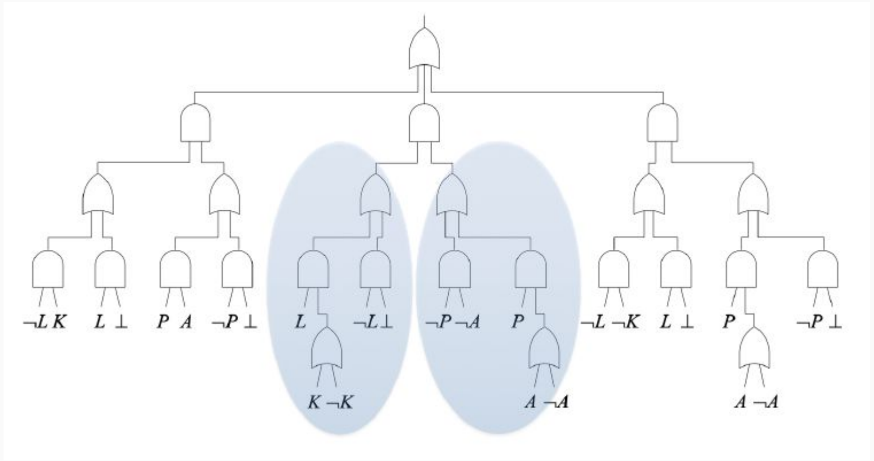
Formula/sentence vs **circuit**: circuit is more compact (no repeat portion) than formula



Compile Sentences into Tractable Circuits

Methods:

- Decomposable NNF (DNNF) circuit
  - **Decomposability**: subcircuits feeding into an **and-gate** share no variables
- Deterministic decomposable NNF (d-DNNF)
  - **Determinism**: at most one high input for each **or-gate** under any circuit input



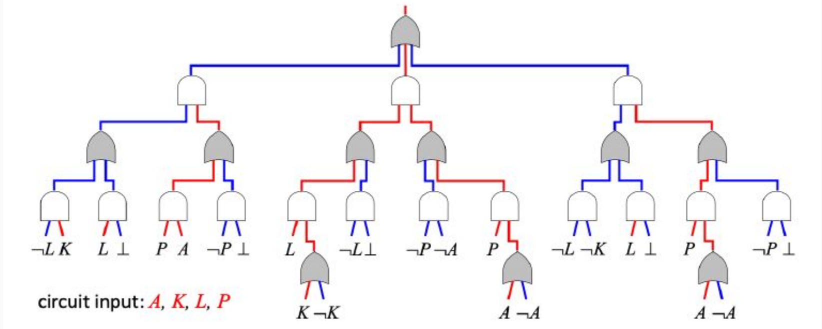


Figure 7: Illustrating the *determinism* property of NNF circuits. Red wires are high and blue ones are low.

Midterm Review – Study Guide

Midterm will be **in person on Monday. Closed book unless stated otherwise**

The midterm in previous years has had some of the following questions. This year's is likely to be similar.

1. Mark nodes according to their order of expansion by different search strategies.
2. Properties of various search algorithms.
3. Minimax and alpha-beta pruning (evaluate nodes according to minimax and prune nodes according to alpha-beta pruning). May not get into details on this.
4. Formulating a problem as A* search — admissible heuristics.
5. Converting propositional sentences to CNF.
6. Answering queries using the method of enumerating models.
7. Answering implication questions using resolution.
8. Write a lisp function.
9. Various true/false questions.

The midterm may slightly deviate from the above list, but the list should give you a very good idea of what to expect.

Midterm Review

LISP

- quote or ‘ : everything under it is kept symbolic
- nil or (): empty list
- car: first element of the list, cdr: the rest of the list (always a list)
- (**cons** arg1 arg2): reverse of car+cdr
- (**list** arg1 … argn): construct a list ‘(arg1 … argn)
- (**append** ‘(l11 l12 …) ‘(l21 l22 …) … ‘(ln1 ln2 …)): ‘(l11 l12 … l21 l22 … ln1 ln2 ...)
- predicates: atom, listp, null, equal
- (cond (cond1 value1) (cond2 value2) … (condn valuen))
- (let ((var1 value1) … (varn valuem)) (expression))
    - let: parallel assignment, let*: sequential assignment

Midterm Review

LISP

- (defun functionName (arg1 … argn) (expression))
- Calling a function: (functionName arg1 … argn)
- General form for a lisp recursion function

```
(defun functionName (arg1 … argn)
    (cond
        (baseCase someValue)
        (Case1 someValue/recursiveCall)
        …
        (t someValue/recursiveCall)
    )
)
```

Midterm Review

SEARCH

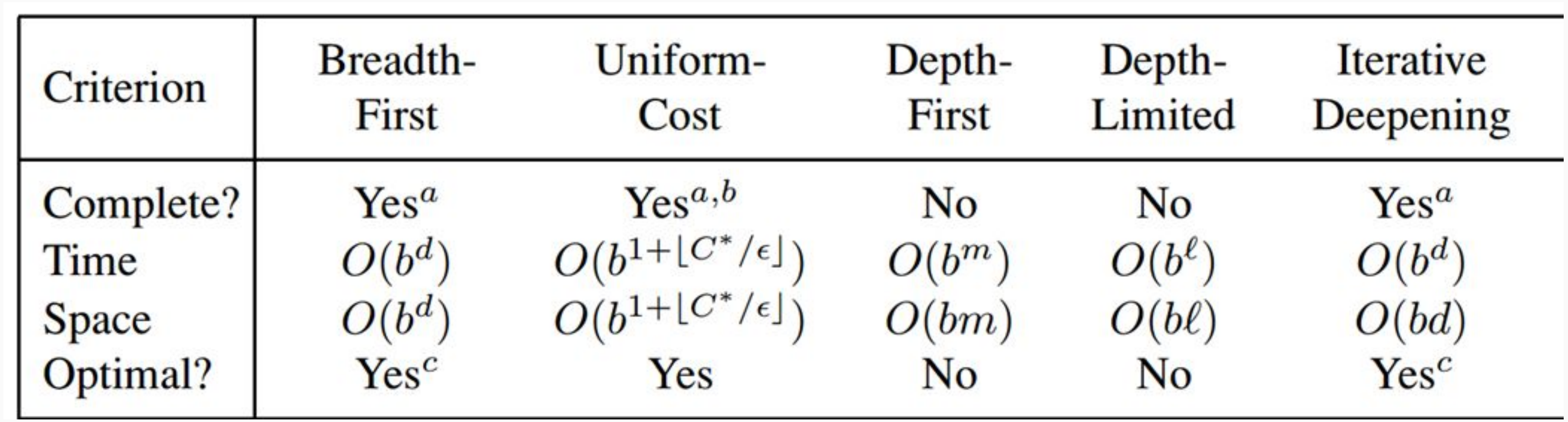
- Search Problem Formulation
  - Initial state, State space, Successor Function, Goal Test/Final State
  - 8 queens - complete formulation and incremental formulation
- State space and search tree - how big are they?
- Solution
  - A path from initial state to goal state
- Node expansion: Goal check + generate children
- Fringe
  - Nodes to expand. Keep in memory. Defines space complexity.

Evaluation of search strategies

- Completeness, Optimality, Time complexity, Space complexity

Midterm Review

Uninformed Search: Properties



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$
Optimal?	Yes[c]	Yes	No	No	Yes[c]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

Midterm Review

Uninformed Search:

- **Breadth-first search**: expands the shallowest nodes first
  - Complete, optimal for unit step costs, exponential space complexity.
- **Uniform-cost search**: expands the node with lowest path cost (root to node)
  - Complete, optimal
- **Depth-first search**: expands the deepest unexpanded node first.
  - Neither complete nor optimal, but has linear space complexity.
- **Depth-limited search:** adds a depth bound to DFS
- **Iterative deepening search**: calls depth-first search with increasing depth limits until a goal is found.
  - Complete, optimal for unit step costs, time complexity comparable to breadth-first search, linear space complexity.

Midterm Review

Informed Search:

- Informed search methods have access to heuristic function h(n)
    - Estimate cost from node n to goal
    - **Admissible: Never overestimates actual cost to get to goal**
- **Greedy search** expands nodes with minimal h(n)
    - Not always optimal but efficient
- **A-star search** expands nodes with minimal g(n) + h(n)
    - Complete and optimal
    - **when h is admissible**

Midterm Review

Constraint Satisfaction:

- Constraint satisfaction problem formulation
    - Variables, Domains, Constraints
- **Backtracking DFS**
    - Fail and backtrack when a consistent assignment is not possible
- **Heuristics**: increase the efficiency of backtracking DFS
    - Variable selection
        - **Most constrained variable**

          Choose unassigned variable with fewest legal values
    - Value selection
        - **Least constraining value**

          Choose value that rules out fewest legal values of remaining variables

Midterm Review

Constraint Satisfaction:

- **Constraint propagation**
    - Node consistency and arc consistency
    - Propagating updates forward along arcs

        O(n^2d^3) - n = variables, d = largest domain
    - Forward checking (variable-level arc consistency)

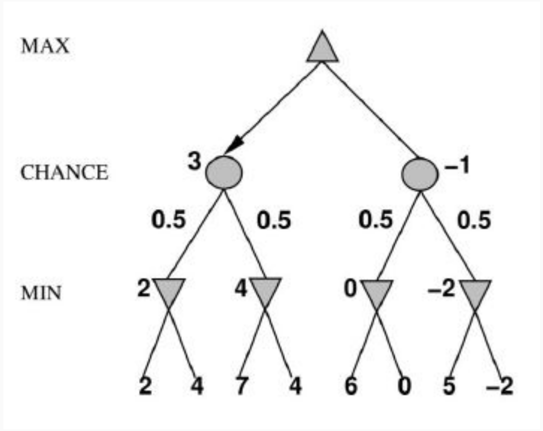
Midterm Review

Game Playing: Basics

- **Minimax**:
    - a utility value for all goal states (leaves)
    - max player: value is max of its children
    - min player: value is min of its children
    - value of the root: the value of the game outcome
- **Expected-minimax**:
    - calculate the expectation over children

      No change in decision making.

      The chance-nature of the game changes tree structure



Midterm Review

Game Playing: Alpha-beta pruning

- Motivation: skip branches that won't matter to improve efficiency
- **A generic algorithm**:
    - During the DFS search, each node carries an lower bound α and an upper bound β.
    - Pushing bound upward: when a child returns, it pushes its value onto the parent (**always tighten the bound**). Min-child pushes onto max-parent's α. Max-child pushes onto min- parent's β. (How to remember: Max player will modify its lower bound, and min player will modify its upper bound.)
    - Pushing bound downward: right before analyzing a child, parent pushes its bound (**Both α and β**) onto that child.
    - Prune all unsearched children when parent has α >= β
- A website for alpha-beta pruning practice:

  http://inst.eecs.berkeley.edu/~cs61b/fa14/ta-materials/apps/ab_tree_practice/

Midterm Review

Propositional Logic:

- Syntax and semantics
- Important terms: model, satisfiability, validity, entailment, etc.
- **Syntactic forms**: CNF, DNF, Horn clauses, NNF, DNNF
    - All but horn clauses are <u>universal</u>. DNF, horn and DNNF are <u>tractable</u>
- **Propositional Inference**:
    - Inference rules
    - Method 1: Proof by enumeration - model checking: E.g. Using a truth table
    - Method 2: Proof by refutation (resolution):
        - Step 1: Convert KB to CNF
        - Step 2: Keep applying inference rules until an empty clause appear

          → Showing $\Delta \land \neg\alpha$ is unsatisfiable!