

TDT4136 Introduction to Artificial Intelligence

Lecture 1: Introduction (Chapter 1 and 2 in the textbook)

Pinar Öztürk

Norwegian University of Science and Technology
Fall 2024

TDT4136 Teaching Team

Coordinator: Pinar Öztürk

Lecturers:

Ole Christian Eidheim

Xavier Fernando Cuauhtémoc Sánchez Díaz

Keith Downing

Ahmed Abouzeid

Pinar Öztürk

Teaching assistants:

Espen Berntzen Høgstedt

Xavier Fernando Cuauhtémoc Sánchez Díaz

Student assistants: 11 assistants.

Syllabus

- Text book: Artificial Intelligence - A Modern Approach, Russel & Norvig, 4th edition (Global)
- Lecture slides
- See blackboard ("Syllabus") for more information.

Lecture slides and streams/videos: Published weekly on Blackboard.

Exam

- The final exam counts 100 %.
- The *mandatory assignments* shall be delivered and passed in order to take the exam. .
- Exam date and form: Written exam. The date is/will be announced centrally by NTNU . The date will also be announced on BB later.
- Exam language: Questions only in English, you can answer in English or Norwegian

Assignments

- There will be 3 mandatory assignments that must be delivered and "passed".
- You may deliver the assignments individually or may work with one another student.
 - You both will have to write your names when delivering your assignment
 - You may not copy work from anyone else
 - You may not give others your solution
- Assignment lectures: There will be 3 assignment lectures during the semester.

Lab hours

- Labs start on 02.09.2024
- Labs on 3 days a week, 2-hours each.
- 2 weeks of lab for each assignment.
- You attend if you need, and as much as you need.

Main information medium and material source

During the semester, you'll find the most updated information on the Blackboard:

- Announcements about practicalities
- The schedules
- Slides for the main and assignment lectures
- Streams/videos for the main and assignment lectures
- The assignments
- Link to discussion forum

You are responsible to make yourself aware of the announcements on the blackboard

More on *mandatory* assignments

Overview

- Assignments 1 (Informed and uninformed search) requires to solve and write the answers by hand, i.e., not programming
- Assignment 2 (constraint satisfaction problem) and assignment 3 (adversarial search) are **is programming assignments**
 - Consider that the student/lærings assistants know Python better!
 - Submit only the necessary files and read instructions carefully!
 - Submit **digital** solutions only: typeset PDF (Word/L^AT_EX) or tablet handwriting
 - Absolutely **no scanned paper nor photos**

Frequently Asked Questions

Assignments

Are assignments mandatory?

Yes, you need to pass 3 mandatory assignments to take the exam.

I am re-taking the course. Do I need the assignments as well?

If you had approved assignments in 2023, then you do **not** need to pass the assignments in 2024.

I have a question about an assignment

Go to the forum¹ and see if the question has been asked before. If not, post your question.

¹See <https://tdt4136.idi.ntnu.no/>

Frequently Asked Questions

Assignments

I signed up for the wrong group!

Send an email to **Espen**²

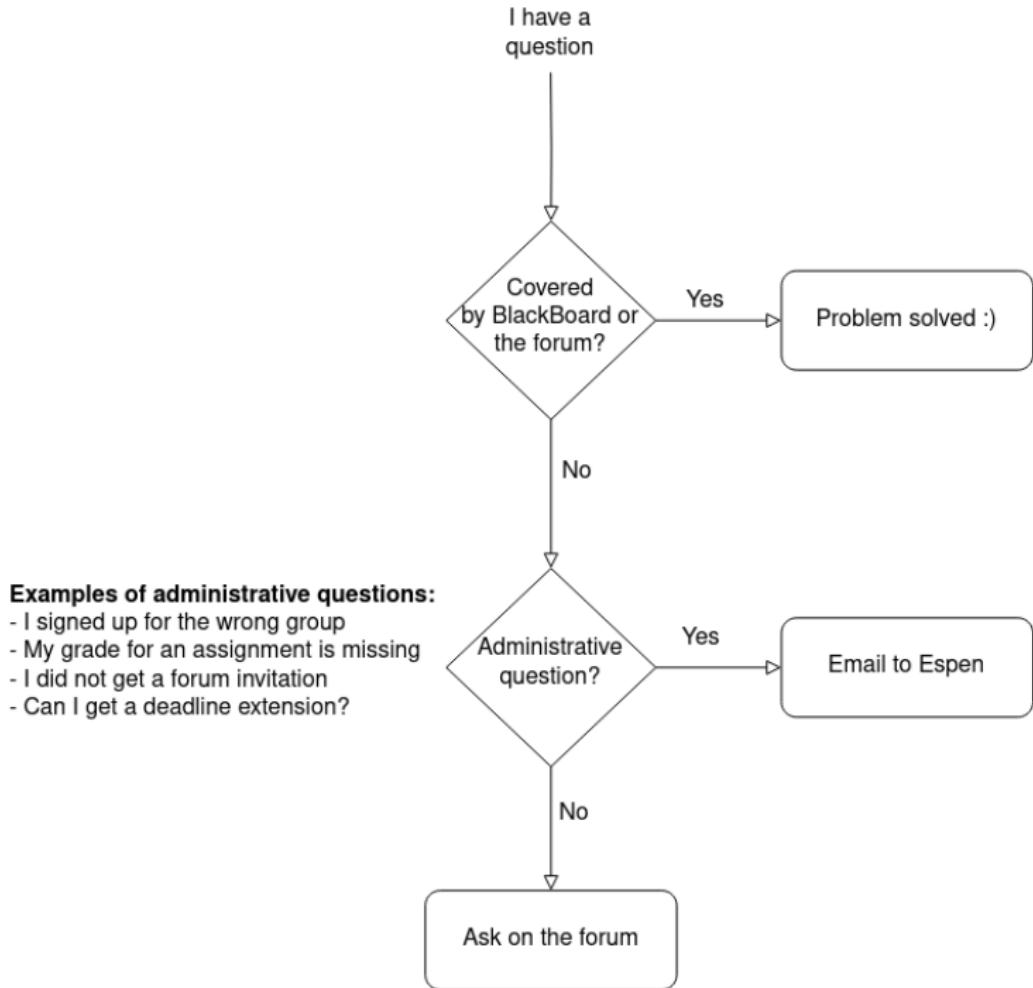
I didn't get a grade for an assignment!

Check that the grading deadline has passed. If the grade is still overdue, send an email to **Espen**

I need an extension to the deadline!

Under certain circumstances you can get an extension (e.g. w/ doctor's notice). Explain why in an email to **Espen**

²espen.b.hogstedt@ntnu.no



Please prefix the topic of any email with
TDT4136

For example

TDT4136 - Incorrect grade on Assignment 2

Frequently Asked Questions

Assignment Lectures

How many assignment lectures will there be?

Three, one for each mandatory assignment.

Are assignment lectures mandatory?

No, they are not mandatory although they could be helpful.

Will assignment lectures be recorded?

Yes, they will be recorded like regular lecture.

Check the detailed FAQ in
Blackboard!

Course topic overview

Intelligent agents

Problem solving by Searching

Adversarial Search

Constraint Satisfaction Problems

Logical systems

Knowledge representation

Ethical issues in AI

Planning

Game Theory

Natural Language processing

Birth of AI field

A Summer Research Project in the Dartmouth College, in 1956 was the birth of the AI research field.



(From left: Trenchard More, John McCarthy, Marvin Minsky, Oliver Selfridge, and Ray Solomonoff)

From the project proposal: "...to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it."

What is AI?

There are no crisp definitions. Here is one from [John McCarthy](#), (father of the phrase *Artificial Intelligence*)

Question: What is artificial intelligence?

Answer: It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.

Question: Yes, but what is intelligence?

Answer: Intelligence is the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people, many animals and some machines.

see <http://www.formal.Stanford.EDU/jmc/whatisai/>

What is AI? - cont

"AI is the science of making machines to do things that would require intelligence if done by men." [Marvin Minsky](#)

"The study of mental faculties through the use of computational models"
[Eugen Charniak](#)

In short: There is no formal definition covering all aspects of intelligence

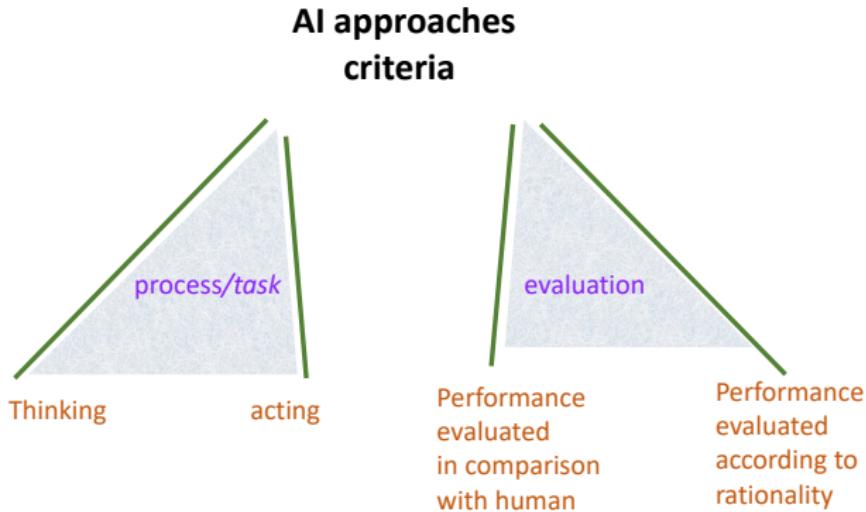
Can machines be Intelligent?

- *Symbolic system hypothesis* (Newell and Simon)
 - Intelligence is substrate neutral
 - A *physical symbol system* has necessary and sufficient means for general intelligent action.
- Biological substrate only (John Searle, philosopher)
 - Intelligence is substrate dependent.
 - The material humans are made of is fundamental for our intelligence.
 - Thinking is possible only in special machines - living ones made of proteins.

What is AI?

Russell and Norvig's definition of AI

Two dimensions:



What is AI?

How R&N define AI

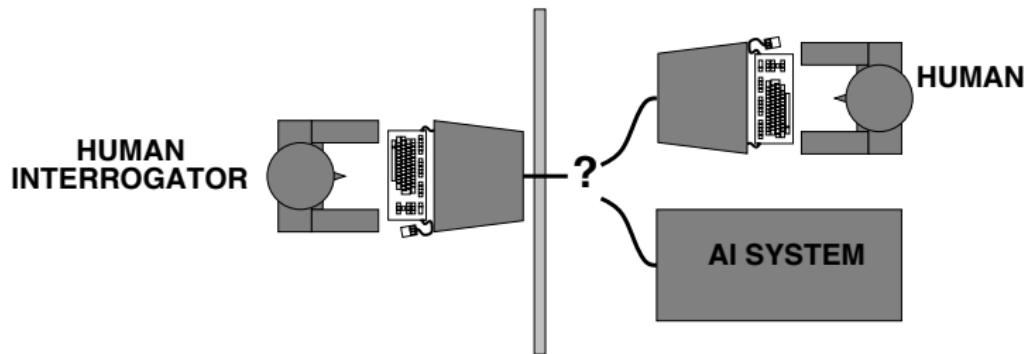
humanly vs. rationally

thinking
vs.
acting

	Systems that think like humans	Systems that think rationally
	Systems that act like humans	Systems that act rationally <i>Rational Agents</i>

Acting humanly

Alan Turing designed an operational test for intelligence -1950



- Computer would need:
 - natural language processing
 - knowledge representation
 - automated reasoning
 - machine learning
- Total Turing test requires interaction with the external world, would require:
 - computer vision and speech recognition
 - robotics

Acting Humanly - Eliza

- One of the most famous early AI programs: Eliza, the computer psychotherapist, created by Joseph Weizenbaum in 1966 at MIT.
- Eliza functions by "twisting the statements of her 'patients' back at them in the classic manner of a non-directive psychotherapist."
- The fact that it understands the subject's statements is an illusion.
- Surprisingly, many users were taking its performance quite seriously.

Acting Humanly - Eliza 2

young woman: Men are all alike.

eliza: In what way?

young woman: They're always bugging us about something specific or other.

eliza: Can you think of a specific example?

young woman: Well, my boyfriend made me come here.

eliza: Your boyfriend made you come here?

young woman: He says I'm depressed much of the time.

eliza: I am sorry to hear you are depressed.

young woman: It's true. I'm unhappy.

eliza: Do you think coming here will help you not to be unhappy?

young woman: I need some help, that much seems certain.

eliza: What would it mean to you if you got some help?

young woman: Perhaps I could learn to get along with my mother.

eliza: Tell me more about your family.

young woman: My mother takes care of me.

eliza: Who else in your family takes care of you?

young woman: My father.

eliza: Your father.

young woman: You are like my father in some ways.

eliza: What resemblance do you see?

young woman: You are not very aggressive but I think you don't want me to notice that.

eliza: What makes you think I am not very aggressive?

young woman: You don't argue with me.

eliza: Why do you think I don't argue with you?

young woman: You are afraid of me.

eliza: Does it please you to believe I am afraid of you?

young woman: My father is afraid of everybody.

eliza: What else comes to mind when you think of your father?

young woman: Bullies.

eliza: Does that have anything to do with the fact that your boyfriend made you come here?

> Hello. My name is Eliza. How may I help you?

Eliza Chat bot

Is she a Rogerian psychotherapist, a semi intelligent chat bot, or just a toy?



Eliza, one of the original is now ready to talk to you. She is programmed to be a Rogerian psychotherapist. #1 interesting example of limitations of early artificial intelligence programs.

If Eliza (or you, or your web browser) gets refresh the page. To start a new session with therapist reload the page.

Thinking Humanly

- Cognitive modelling approach
- Learn about how humans think
 - introspection
 - psychological experiments
 - brain imaging
- Cognitive Science promising

Thinking rationally

- Aristotle: what are correct thought processes - Laws of Thought?
 - Formalize “correct” reasoning using a mathematical model
 - Syllogism (as a law of thought) codifies the patterns for argument structures.
- Theory of probability

Thinking rationally



Thinking rationally



Thinking rationally

Problems:

- General deductive inference is computationally intractable
- It does not generate intelligent *behaviour*

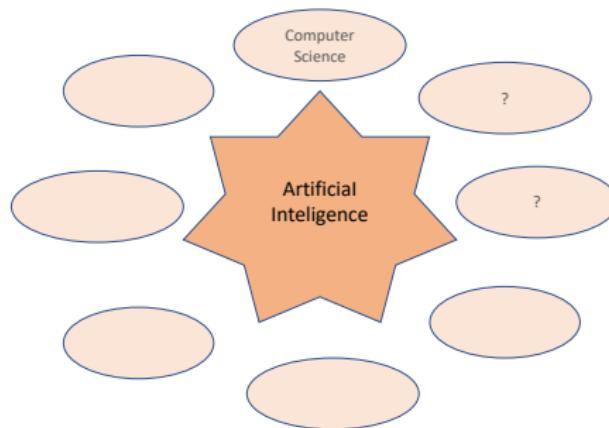
Acting Rationally: Rational Agents approach

- Rational behavior: doing the right thing
- The right thing: that which is expected to maximize goal achievement, given the available information and computational abilities
- Doesn't necessarily involve thinking—e.g., blinking reflex—but thinking should be in the service of rational action
- Two advantages:
 - Not limited to "laws of thought" in order to achieve rationality
 - Rationality is mathematically well-defined - more operational

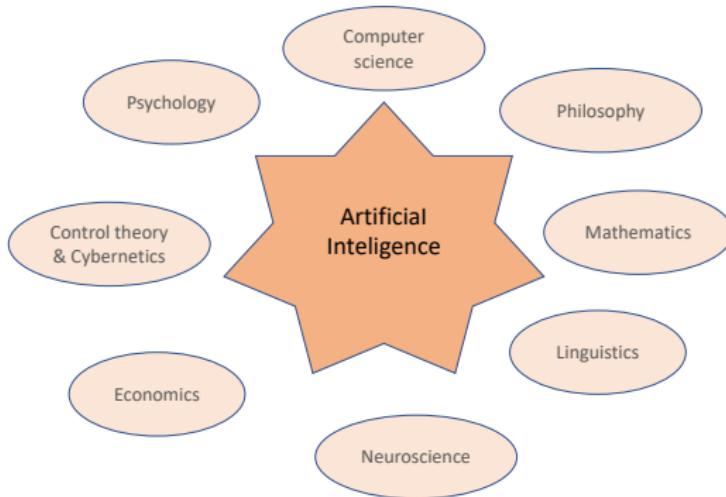
Foundations of AI

Computer Science is the main discipline underlying AI

Can we think which other disciplines AI is grounded on?



Foundations of AI



Brief history of AI

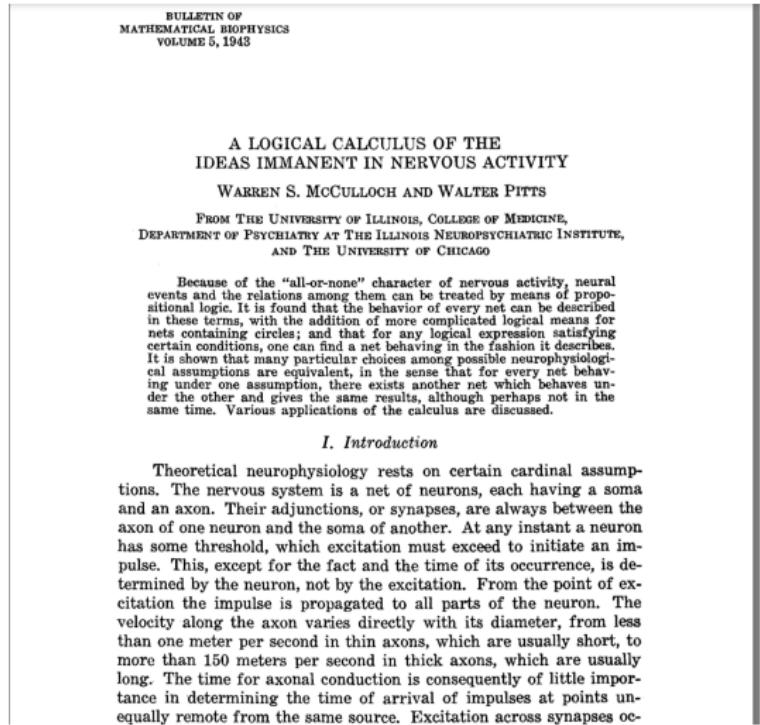
- 1943 McCulloch & Pitts: Boolean circuit model of brain
- 1950 Turing's "Computing Machinery and Intelligence"
- 1952-69 Early AI programs, including Samuel's checkers program, Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
- 1956 Dartmouth meeting: "Artificial Intelligence" adopted
- 1962 Rosenblatt's Perceptron for training simple neural networks
- 1965 Robinson's complete algorithm for logical reasoning
- 1972 The logic programming language PROLOG is created.
- 1966–74 Disappointment: AI discovers computational complexity
Neural network research almost disappears

Brief history of AI -cont.

- 1969–79 Early development of knowledge-based systems
- 1980–88 Expert systems industry booms
- 1988–93 Expert systems industry busts: “AI Winter”
- 1985–95 Backpropagation learning returns neural networks to popularity
- 1988– Resurgence of probability; general increase in technical depth
“Nouvelle AI”: ALife, GA
- 1995– Agents, agents, everywhere ...
- 2003– Human-level AI back on the agenda
- 2005–2010 AI disappoints again, AI is not much appreciated
- 2012 - Deep learning and a very hot AI summer since then

Example from Early History of AI

1943 McCulloch & Pitts: Boolean circuit model of brain



Some killer apps in AI history

- 1991** During the Gulf War, US forces deployed an AI logistics planning and scheduling program that involved up to 50,000 vehicles, cargo, and people. Saved the US more money than spent on all AI research since 1950
- 1997** Deep Blue (IBM) defeated world chess champion Gerry Kasparov
- 2011** Watson (IBM) beat human champions on “Jeopardy”
- 2012** Google car obtains driver's license in Nevada, US.
- 2017** DeepMind's Alphago AND Elon Musk's A.I. Destroys Champion Gamer!
- 2022** Open AI's. ChapGPT, language model/text generation.

How humanly is Sofia (Hong Kong firm Hanson Robotics)?

<https://www.youtube.com/watch?v=suRuQbDXcrc>

More Videos

Computers with common sense(Doug Lenat):

https://www.youtube.com/watch?v=2w_ekB08ohU

Artificial Intelligence: The Common Sense Problem (Hubert Dreyfus)

<https://www.youtube.com/watch?v=SUZUbYCBtGI>

AI Boom (Rodney Brooks):

<https://www.youtube.com/watch?v=ovbzjLobiDo>

IBM Watson : https://www.youtube.com/watch?v=_Xcmh1LQB9I

Jeopardy : https://www.youtube.com/watch?v=WFR3l0m_xhE

State of the art AI (Minsky):

https://www.youtube.com/watch?v=aODnFdU_hds

John McCarthy talks about AI:

Chapter 2 Outline

What is an agent?

Rationality

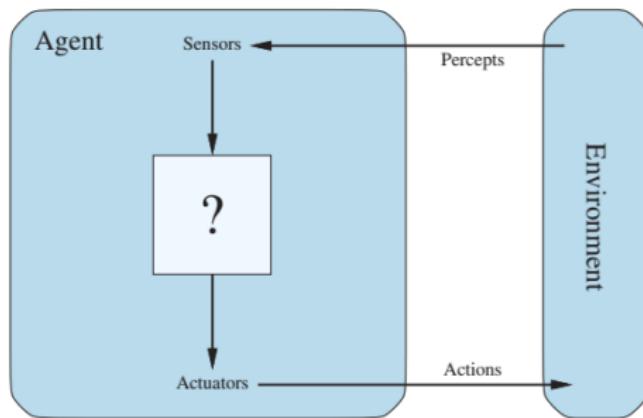
PEAS (Performance measure, Environment, Actuators, Sensors)

Environment types

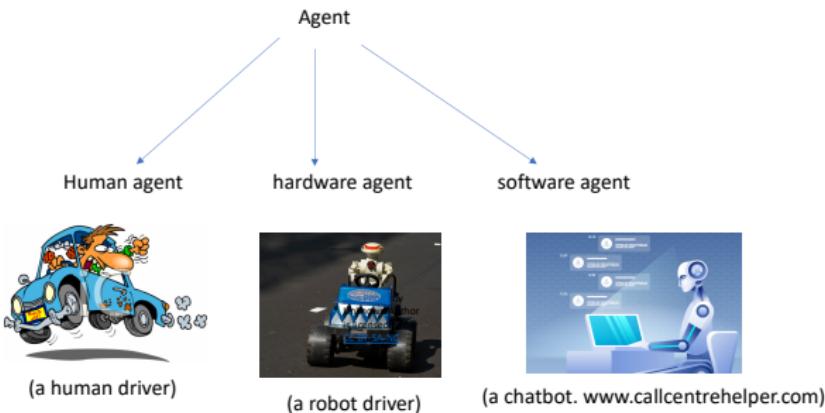
Agent types

What is an agent

An agent is anything that can be viewed as *perceiving* its *environment* through *sensors* and acting upon that environment through *actuators*



What is an agent



robot:

perceives: with video-cameras, infra-red sensors, radar, ...
acts: with wheels, motors

softbot:

perceives: receiving keystrokes, files, network packets, ...
acts: displaying on the screen, writing files, sending network packets

....

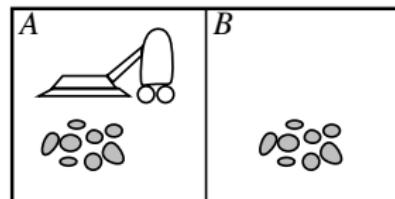
Agents behaviour

An agent's choice of action at any given instant can depend on the entire percept sequence observed to date

Hence, agent's behaviour is described by the **agent function** that maps any percept sequence to an action.

A simple agent function

- Example: vacuum cleaner world. If the current square is dirty, then suck; otherwise, move to the other square.
- Tabulating the vacuum's agent function:



Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
:	:

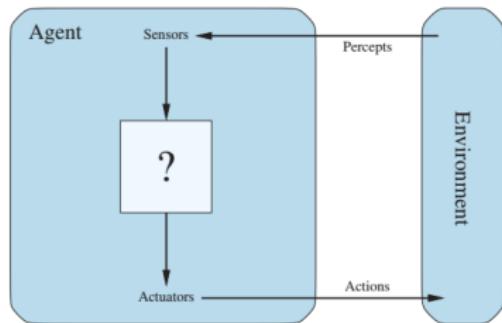
What is the right way to fill this table?

Agent Behaviour

Agent function - math description of behaviour, maps from percept sequences to actions:

$$f : \mathcal{P} \rightarrow \mathcal{A}$$

Agent program - concrete implementation of agent function, runs on the physical *architecture* to produce f



Rational agent

- A rational agent is one that "does the right thing" in context
 - but what does this really mean?
- The right action should cause the agent to be most successful
 - but how and when should the success of the agent be evaluated?
- Success needs to be evaluated with respect to an objective *performance measure*, which depends on what the agent is designed to achieve

Performance measure

Performance measure evaluates the environment sequence - *consequentialism*.

- For example, in case of the vacuum cleaner, the designer could focus only on collecting as much dirt as possible in time T
- Or, take also other factors into account, e.g :
 - amount of time taken to clean
 - amount of electricity consumed
 - amount of noise generated, etc
- e.g., one point per square cleaned up in time T , or
 - one point per clean square per time step, minus one per move
 - etc

Rational agent

For each possible percept sequence, an ideal rational agent should do whatever is expected to maximise its performance measure, on the basis of the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Is the "tabulated vacuum agent" rational under these conditions:

- performance measure gives 1 point for each clean room at each time step, over a lifetime of 100 time steps
- there are 2 rooms, the agent has location and dirt sensors. Clean room stays clean, and "suck" works properly
- available actions are right, left, suck
- no noise or error in sensors

Rationality requires learning and autonomy

- Learning agent modifies knowledge of the environment from experience
- An agent is autonomous to the extent that its behaviour is determined by its own experience
 - complete autonomy from the start is too difficult
 - the agent's designer must give guidance in terms of some initial knowledge and the ability to learn and/or reason as it operates in its environment
- Example: After a dung beetle digs its nest and lays its eggs, it fetches a ball of dung from a nearby heap to plug the entrance....



Task Environment

The task environment is defined through "PEAS":

- Performance measure
- The agent's prior knowledge of the environment
- The actions that the agent can perform
- The agent's percept sequence to date

Task Environment

PEAS (Performance measure, Environment, Actuators, Sensors)

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits, minimize impact on other road users	Roads, other traffic, police, pedestrians, customers, weather	Steering, accelerator, brake, signal, horn, display, speech	Cameras, radar, speedometer, GPS, engine sensors, accelerometer, microphones, touchscreen

Medical diagnosis system agent

Performance measure??

Environment??

Actuators??

Sensors??

Medical diagnosis system agent

Performance measure : healthy patient, minimize costs, lawsuits

Environment: patient, hospital, staff

Actuators: screen display (questions, tests, diagnoses, treatments, referrals)

Sensors: keyboard (entry of symptoms, findings, patient's answers)

Fully observable environments

- Relevant parts of the state of the environment can be sensed
- No need to maintain any internal state to keep track of the world
- Example: chess, image analysis

Partially observable environments

- Parts of the environment cannot be sensed
- Agent must make informed guesses about world
- Example: poker, taxi driving, medical diagnosis

Single agent

- No other agents - there may be but as a part of the environment
- Examples: medical diagnosis, image analysis

Multi-agent

- Which entities will be viewed as "other agents"?
- The environment contains other agents whose performance measure depends on my actions and vice versa
- Competitive and cooperative interactions
- Examples: poker, chess, taxi driving

Deterministic environment

Any action has a single guaranteed effect, and no uncertainty/failure.

Example: chess, image analysis

Non-deterministic environment

- There is some uncertainty about the outcome of an action
- Multiple outcome alternatives
- Called "stochastic" if alternatives are quantified in terms of probabilities
- Example: poker, taxi driving, medical diagnosis

Properties of environments

Episodic environments

- The agent's experience is divided into atomic episodes.
- Each episode consists of the agent perceiving and then performing a single action
- The choice of action in each episode depends only on the episode itself
- Examples: image analysis

Sequential environments

- The current decision could affect all future decisions
- Examples: poker, chess, taxi driving, medical diagnosis

Properties of environments

Discrete

Finite number of distinct states, percepts and actions,

Examples: chess, poker

Continuous

Continuous time/state/actions

Example: taxi driving, medical diagnosis, image analysis

Properties of environments

Dynamic environment

- May change while an agent is deliberating
- Examples: taxi driving, medical diagnosis

Static environment

- The environment does not change
- Examples: poker

Semidynamic

- The world does not change but the agent's performance score may
- Examples: chess with clock, image analysis

Environment	Observable	Deterministic	Episodic	Discrete	Dynamic	Agent
Crossword	fully	deterministic	sequential	discrete	static	
Chess w/clock	fully	deterministic	sequential	discrete	semidynamic	
Backgammon	fully	stochastic	sequential	discrete	static	
Taxi driving	partially	stochastic	sequential	continuous	dynamic	
Medical diagnosis	partially	stochastic	sequential	continuous	dynamic	

Properties of environments

Known environment

- The agent's knowledge about how the environment works/evolves
- Note that a known environment (i.e., the agent knows all the rules that apply) may be only partially observable



Unknown environment

- The agent will have to learn how it works
- An unknown environment can be fully observable.

A simple vacuum cleaner agent

Table-driven agent:

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
:	:

Table-driven agent

```
function TABLE-DRIVEN-AGENT(percept) returns an action
  persistent: percepts, a sequence, initially empty
              table, a table of actions, indexed by percept sequences, initially fully specified

  append percept to the end of percepts
  action  $\leftarrow$  LOOKUP(percepts, table)
  return action
```

- looks up the right response in the table
- Infeasible: if there are $|P|$ percepts and a life-time of T , then need for a look-up table of size $\sum_{t=1}^T |P|^t$

Agent programs

Writing down such agent functions is not feasible

The goal of AI is to write agent programs with small code which produced the desired rational behaviour

Agent types

- simple reflex agents
- model-based reflex agent
- goal-based agents
- utility-based agents
- learning agents

Simple reflex agent

- Uses only the current percept - ignores the percept sequence,
- Implemented through condition-action rules
- Large reduction in possible percept/action situations from $\sum_{t=1}^T |P|^t$ to $|P|^3$

Example:

```
function Reflex-Vacuum-Agent( [location,status]) returns an action
    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```

³if there are $|P|$ percepts and a life-time of T, then need for a look-up table of size $\sum_{t=1}^T |P|^t$

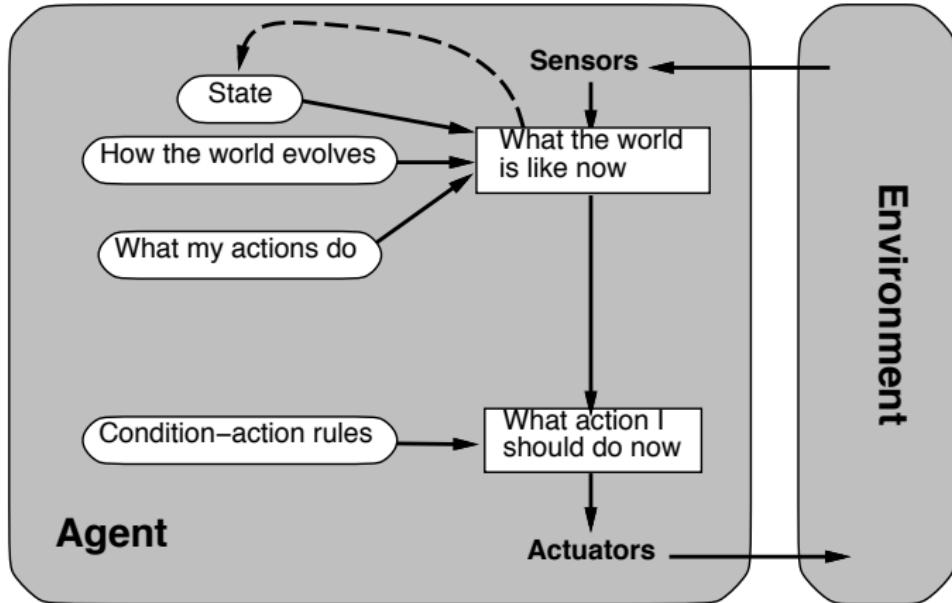
Generic Simple Reflex Agent program

A more general simple reflex agent program:

```
function SimpleReflexAgent( percept) returns an action  
  
    persistent rules  
    state ← Interpret(percept)  
    rule ← Rule-match(state, rules)  
    action ← Rule-action(rule)  
    return action
```

- Will only work if the environment is fully observable
- everything relevant needs to be determinable from the current input
- otherwise infinite loops may occur, e.g. in the vacuum world without a sensor for the room, the agent does not know whether to move right or left
 - any possible solution: ?

Model-based Reflex agents

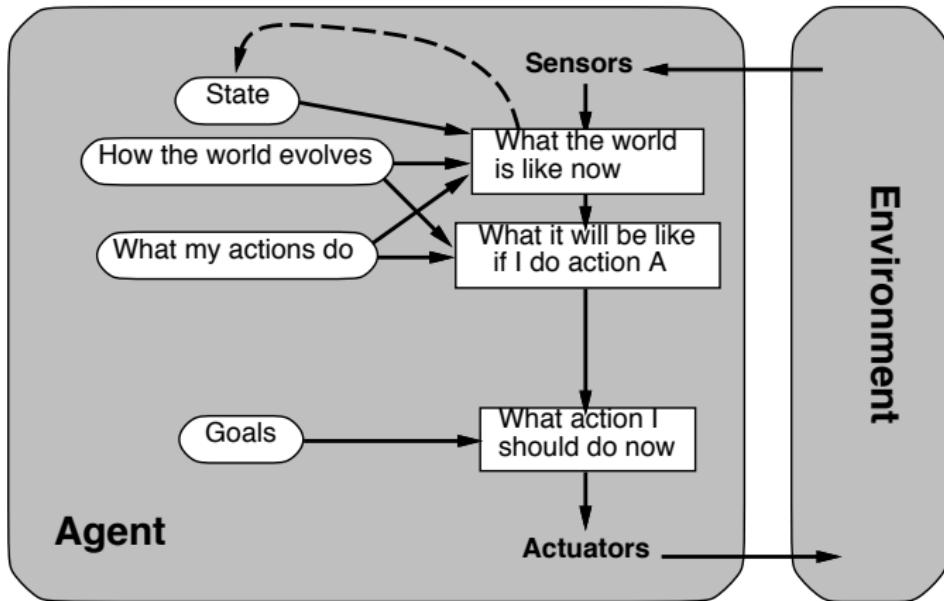


Model-based Reflex agents

```
function ModelBasedReflexAgent( percept) returns an action  
  
    persistent state, the agent's current conception of the world,  
    transition-model, a description of how the next state depends on  
    the current state and the action,  
    sensor-model, a description of how the current world state is reflected  
    in the agent's percepts  
    rules, a set of condition-action rules  
    action, the most recent action, initially none  
    state ← UpdateState(state,action,percept,transition-model,sensor-model)  
    rule ← Rule-match(state, rules)  
    action ← Rule-action(rule)  
    return action
```

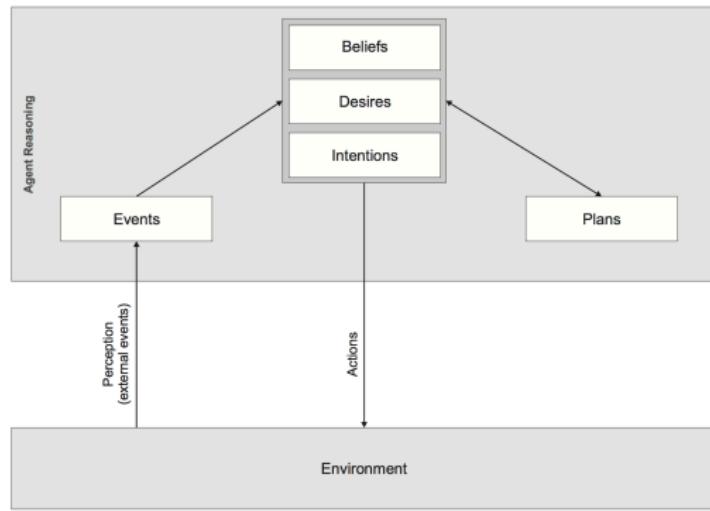
- difficult to exactly determine the current state in partially observable environments
 - independent from the kind of models used
- hence, may need to "guess" the current situation
- action decision in the same way as the simple reflex agent.

Goal-based agents



Example

- Belief Desire Intention agents (BDI) agents have a **mental state** as the basis for their reasoning.
- Three main mental attitudes: beliefs, desires and, intentions.
- Their reasoning is also called **practical reasoning** - e.g., contrary to deductive reasoning



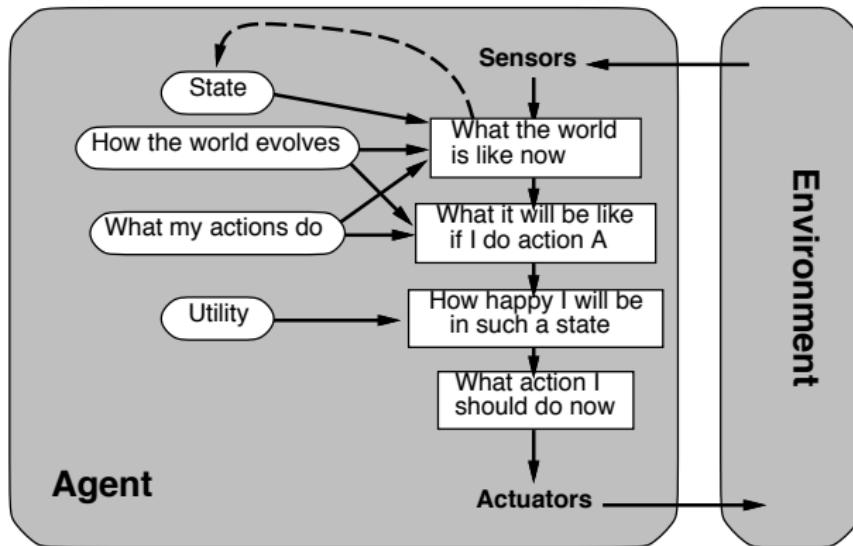
Goal-based agents

- explicit representation of goals: agent knows which states are desirable
- reasoning about goals
- more flexible since knowledge is represented explicitly and can be manipulated
- different reasoning methods than reflex - not condition-action rules and selection of rule/action
- main difference from reflex agents: deliberates about future when making decision
- long sequence of actions may be needed to achieve the goal
 - e.g., agents in chapters 3-5 (search) and chapters 10-11 planning
- less efficient than reflex agents - but more flexible

Utility-based agents

Goals provide just a binary happy/unhappy distinction while utility functions provide a continuous scale

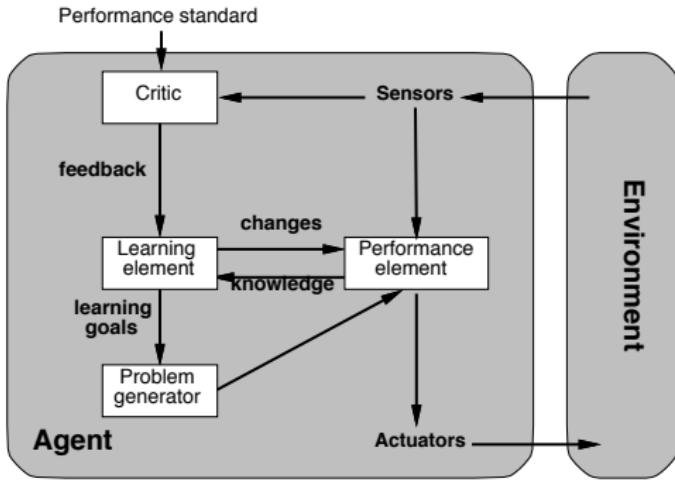
Some goals may be achieved in more than one way, with different utility values



Utility-based agents

- utilities are internalization of the performance measure
- Utility function maps a state (or a sequence of states) onto a real number
- utilities reflect agent's preferences
- not always know the utility of an action/outcome , hence "expected utility"
- agent chooses actions that maximize the expected utility of the outcomes

Learning agents



- Critic: evaluates current world state, determines how the performance should be modified
- Learning element: responsible for making improvements
- Problem generator: suggests explorations
- Performance element: responsible for selecting external actions

Next Week: Goal-based Agents

Goal-based agents that use search methods as Agent Function.
You are at the end of your holiday in Romania. Your return ticket is from Bucharest and you are leaving Arad for Bucharest.



- **Uninformed Search** method as Agent Function.
- In an environment:
 - Fully observable (the agent can see the current state of the world),
 - Deterministic (action has a single outcome)
 - Discrete
 - Known environment (knows which states can be reached through which actions - has a map of Romania).

Next week: Problem solving as search, chap 3.

Lecture 2

Solving Problems by Searching

TDT4136: Introduction to Artificial Intelligence

Xavier F. C. Sánchez Díaz

Department of Computer Science
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology

August 29, 2024

Outline

1 Problem solving and search

2 The search algorithm

3 Uninformed search strategies

Breadth-first search

Depth First Search

Depth-limited and Iterative deepening search

4 Informed search strategies

(Greedy) Best First Search

A* Search

Why searching?

Problem solving and search

- ▶ Some problems have straightforward solutions
 - ▶ Solved by applying a formula, or a well-known procedure
 - ▶ Example: differential equations
- ▶ Other problems require **search**:
 - ▶ no single standardised method
 - ▶ alternatives need to be explored to solve the problem
 - ▶ the number of alternatives to search among can be very large, even infinite.

Why searching?

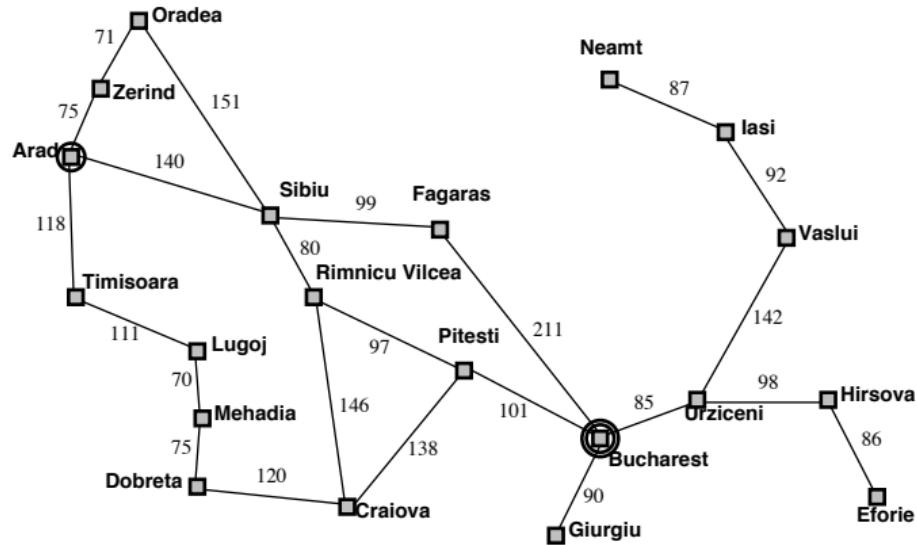
Problem solving and search

- ▶ Some problems have straightforward solutions
 - ▶ Solved by applying a formula, or a well-known procedure
 - ▶ Example: differential equations
- ▶ Other problems require **search**:
 - ▶ no single standardised method
 - ▶ alternatives need to be explored to solve the problem
 - ▶ the number of alternatives to search among can be very large, even infinite.

This happens often in the real world, where there is a **cost** associated with our **actions**.

An example about search

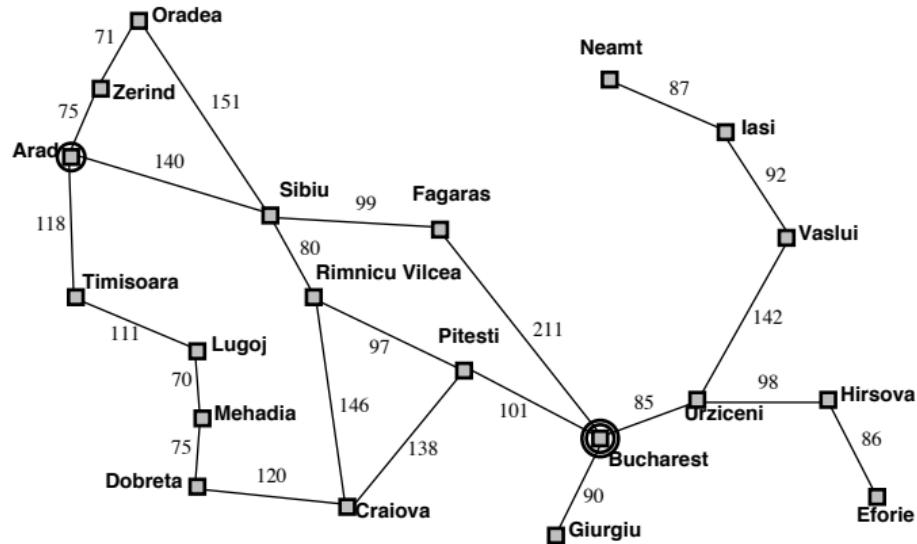
Problem solving and search



A simplified map of part of Romania, with road distances in miles.

An example about search

Problem solving and search



A simplified map of part of Romania, with road distances in miles.

Find a **sequence of cities** to drive through, from **Arad** to **Bucharest**.

How to solve it?

Problem solving and search

- ▶ Formulate the **start** and **goal states**
- ▶ What other **states** are there in the problem? What are the possible **actions** we can take?

How to solve it?

Problem solving and search

- ▶ Formulate the **start** and **goal states**
- ▶ What other **states** are there in the problem? What are the possible **actions** we can take?

And now, **search**:

- ▶ Simulate sequences of actions in the world to find a sequence that reaches the goal.
 - ▶ This sequence of actions is **the solution!**

How to solve it?

Problem solving and search

- ▶ Formulate the **start** and **goal states**
- ▶ What other **states** are there in the problem? What are the possible **actions** we can take?

And now, **search**:

- ▶ Simulate sequences of actions in the world to find a sequence that reaches the goal.
 - ▶ This sequence of actions is **the solution!**
- ▶ Execute: carry out the necessary actions in the solution, one at a time.

How to solve it?

Problem solving and search

We need to define the **state space**: all **possible states** of the environment.

How to solve it?

Problem solving and search

We need to define the **state space**: all **possible states** of the environment.

- ▶ State space: all locations in the Romania map

How to solve it?

Problem solving and search

We need to define the **state space**: all **possible states** of the environment.

- ▶ State space: all locations in the Romania map
- ▶ Initial state: *Arad*

How to solve it?

Problem solving and search

We need to define the **state space**: all **possible states** of the environment.

- ▶ State space: all locations in the Romania map
- ▶ Initial state: *Arad*
- ▶ Goal state: *Bucharest*

How to solve it?

Problem solving and search

We need to define the **state space**: all **possible states** of the environment.

- ▶ State space: all locations in the Romania map
- ▶ Initial state: *Arad*
- ▶ Goal state: *Bucharest*
- ▶ Actions: *Go from Arad to Sibiu, and to Timisoara, ...*

How to solve it?

Problem solving and search

We need to define the **state space**: all **possible states** of the environment.

- ▶ State space: all locations in the Romania map
- ▶ Initial state: *Arad*
- ▶ Goal state: *Bucharest*
- ▶ Actions: *Go from Arad to Sibiu, and to Timisoara, ...*
 - ▶ This can be translated into a **transition model** that can consider **action costs**.

How to solve it?

Problem solving and search

We need to define the **state space**: all **possible states** of the environment.

- ▶ State space: all locations in the Romania map
- ▶ Initial state: *Arad*
- ▶ Goal state: *Bucharest*
- ▶ Actions: *Go from Arad to Sibiu, and to Timisoara, ...*
 - ▶ This can be translated into a **transition model** that can consider **action costs**.

How to solve it?

Problem solving and search

We need to define the **state space**: all **possible states** of the environment.

- ▶ State space: all locations in the Romania map
- ▶ Initial state: *Arad*
- ▶ Goal state: *Bucharest*
- ▶ Actions: *Go from Arad to Sibiu, and to Timisoara, ...*
 - ▶ This can be translated into a **transition model** that can consider **action costs**.

A good problem formulation has the appropriate **level of abstraction**.

Make it formal with maths

Problem Solving and search

The **state space** is a [set](#):

$$S = \{Oradea, Zerind, Arad, Timisoara, Lugoj, Dobreta, \dots\}$$

Make it formal with maths

Problem Solving and search

The **state space** is a set:

$$S = \{Oradea, Zerind, Arad, Timisoara, Lugoj, Dobreta, \dots\}$$

The **starting** s_0 and **goal** s_g states are elements of the **state space**: $s_0 = Arad$ and

$s_g = Bucharest$

Make it formal with maths

Problem Solving and search

The **state space** is a [set](#):

$$S = \{Oradea, Zerind, Arad, Timisoara, Lugoj, Dobreta, \dots\}$$

The **starting** s_0 and **goal** s_g states are [elements](#) of the **state space**: $s_0 = Arad$ and

$$s_g = Bucharest$$

Actions can be summarised by a [function](#) A which maps a **state** (origin) to **another state** (destination): $A : S \rightarrow S$. Basically, a table (or a set, really ...)

Make it formal with maths

Problem Solving and search

The **state space** is a [set](#):

$$S = \{Oradea, Zerind, Arad, Timisoara, Lugoj, Dobreta, \dots\}$$

The **starting** s_0 and **goal** s_g states are [elements](#) of the **state space**: $s_0 = Arad$ and $s_g = Bucharest$

Actions can be summarised by a [function](#) A which maps a **state** (origin) to **another state** (destination): $A : S \rightarrow S$. Basically, a table (or a set, really ...)

The **transition model** can be another [function](#), f , which maps **an action** to a **cost** (a number): $f : A \rightarrow \mathbb{R}$

Make it formal with maths

Problem Solving and search

The **state space** is a [set](#):

$$S = \{Oradea, Zerind, Arad, Timisoara, Lugoj, Dobreta, \dots\}$$

The **starting** s_0 and **goal** s_g states are [elements](#) of the **state space**: $s_0 = Arad$ and $s_g = Bucharest$

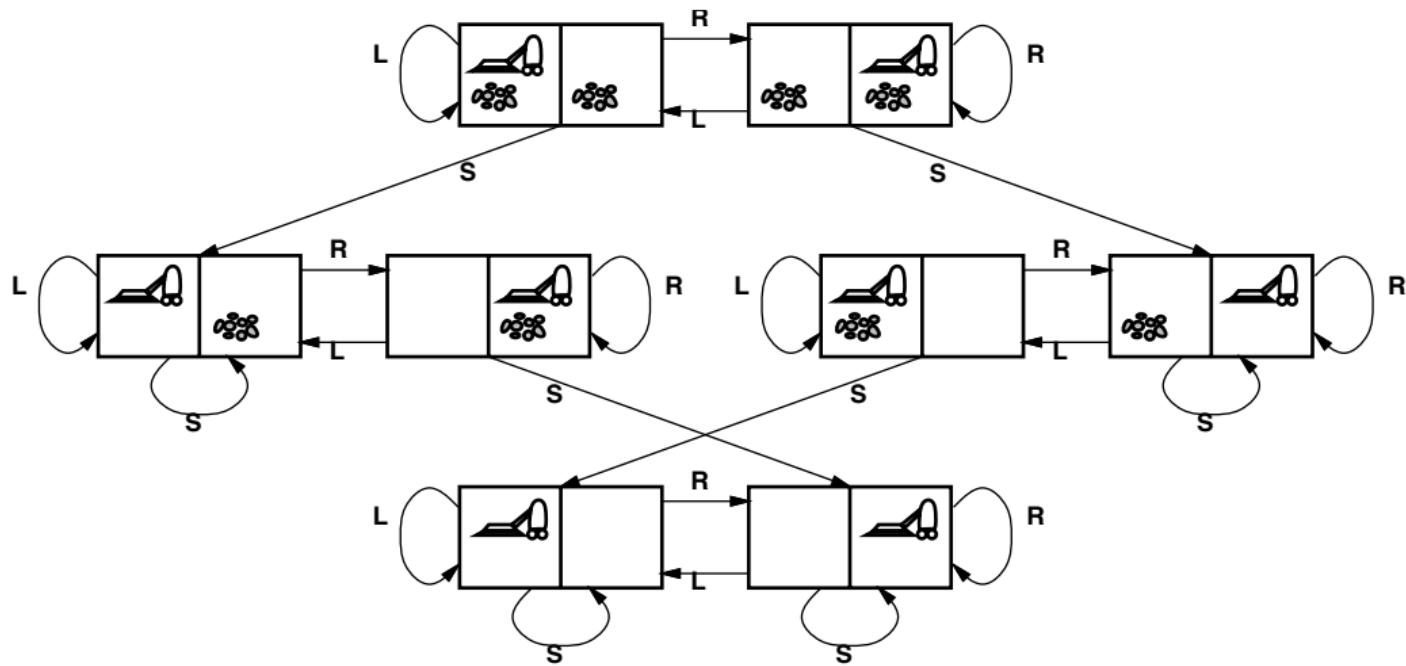
Actions can be summarised by a [function](#) A which maps a **state** (origin) to **another state** (destination): $A : S \rightarrow S$. Basically, a table (or a set, really ...)

The **transition model** can be another [function](#), f , which maps **an action** to a **cost** (a number): $f : A \rightarrow \mathbb{R}$

Having it in mathematical terms makes it [easier to code!](#)

Another example: Vacuum world

Problem solving and search

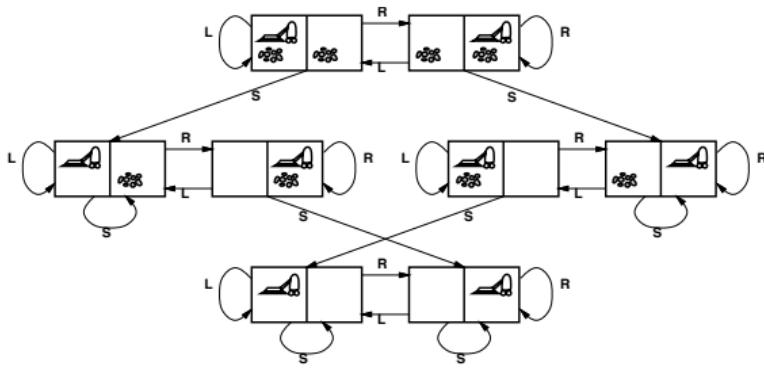


¹p. 85 in the textbook.

Another example: Vacuum world

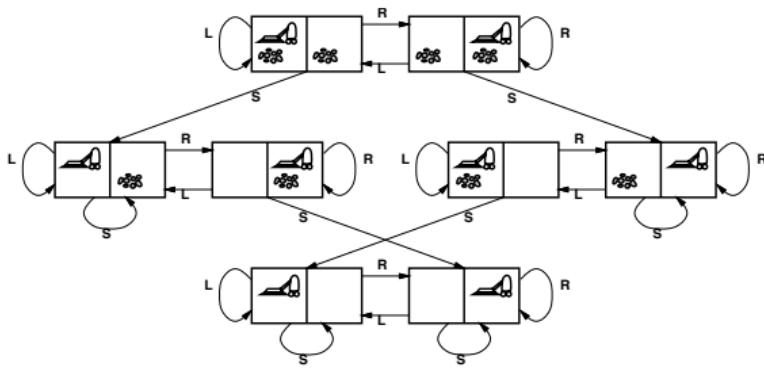
Problem solving and search

► Initial state: any state



Another example: Vacuum world

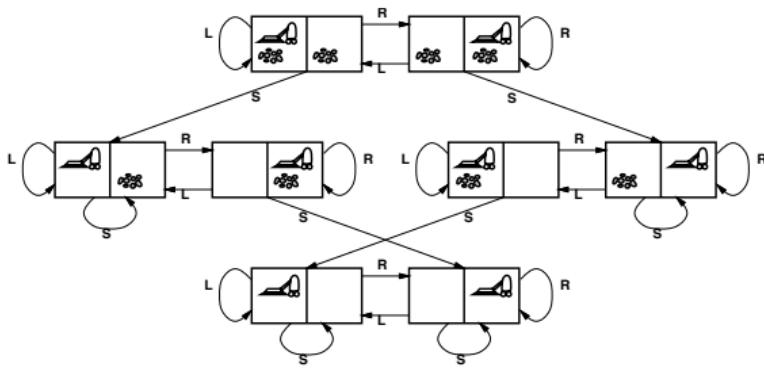
Problem solving and search



- ▶ Initial state: any state
- ▶ Goal states: no dirt

Another example: Vacuum world

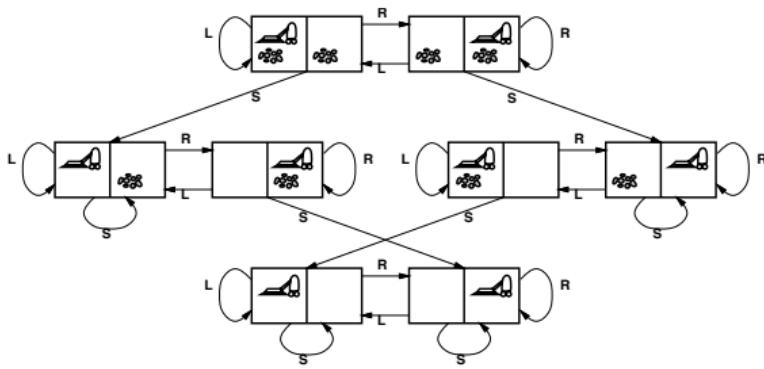
Problem solving and search



- ▶ Initial state: any state
- ▶ Goal states: no dirt
- ▶ Actions: *Suck, moveLeft, moveRight*
... or *Suck, Forward, Backward,*
TurnLeft, TurnRight, ...

Another example: Vacuum world

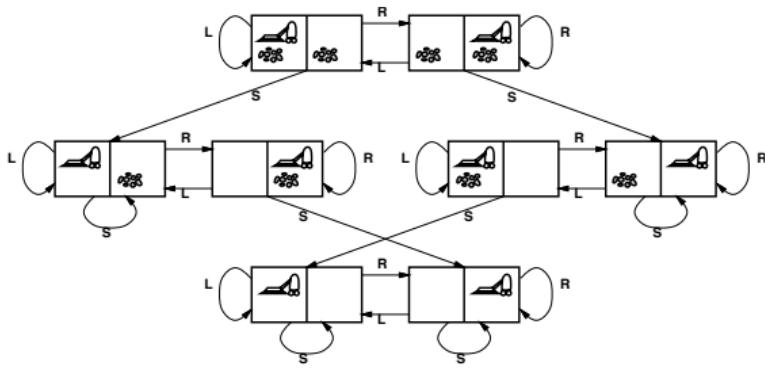
Problem solving and search



- ▶ Initial state: any state
- ▶ Goal states: no dirt
- ▶ Actions: *Suck, moveLeft, moveRight* ... or *Suck, Forward, Backward, TurnLeft, TurnRight, ...*
- ▶ Transition model: *Suck* cleans the location, *TurnLeft* changes the direction of agent 90 degrees

Another example: Vacuum world

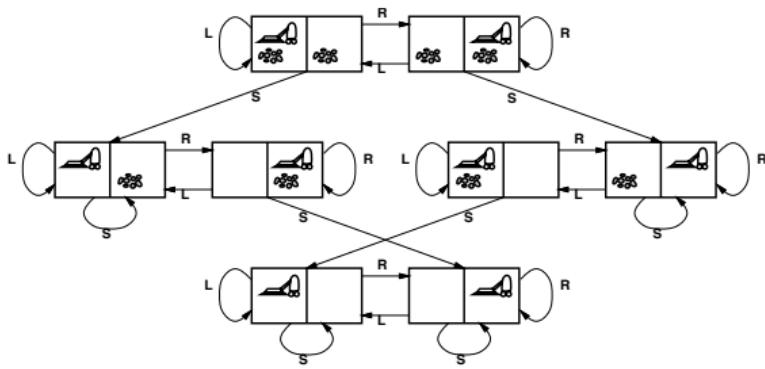
Problem solving and search



- ▶ Initial state: any state
- ▶ Goal states: no dirt
- ▶ Actions: *Suck, moveLeft, moveRight* ... or *Suck, Forward, Backward, TurnLeft, TurnRight, ...*
- ▶ Transition model: *Suck* cleans the location, *TurnLeft* changes the direction of agent 90 degrees
- ▶ Action cost: 1 unit

Another example: Vacuum world

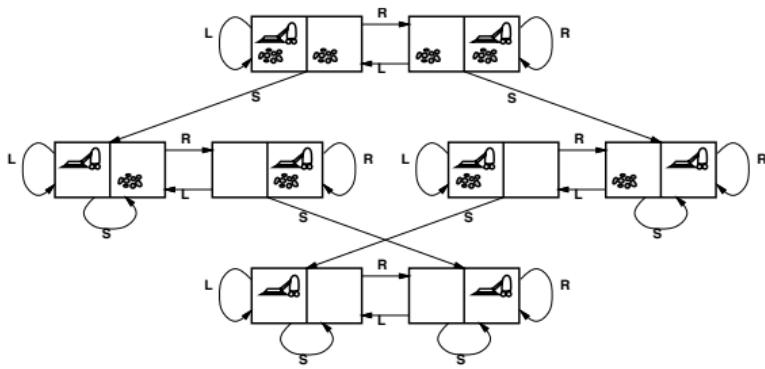
Problem solving and search



- ▶ Initial state: any state
- ▶ Goal states: no dirt
- ▶ Actions: *Suck, moveLeft, moveRight* ... or *Suck, Forward, Backward, TurnLeft, TurnRight, ...*
- ▶ Transition model: *Suck* cleans the location, *TurnLeft* changes the direction of agent 90 degrees
- ▶ Action cost: 1 unit

Another example: Vacuum world

Problem solving and search



- ▶ Initial state: any state
- ▶ Goal states: no dirt
- ▶ Actions: *Suck, moveLeft, moveRight* ... or *Suck, Forward, Backward, TurnLeft, TurnRight, ...*
- ▶ Transition model: *Suck* cleans the location, *TurnLeft* changes the direction of agent 90 degrees
- ▶ Action cost: 1 unit

Notice how we do not care much about costs here!

Applications

Problem solving and search

These kind of search problems happen all the time!



Applications

Problem solving and search

But the real world is usually **more complex!**

- ▶ Resources are limited (and **costs** become important!)

Applications

Problem solving and search

But the real world is usually **more complex!**

- ▶ Resources are limited (and **costs** become important!)
- ▶ We have **constraints** and **restrictions**

Applications

Problem solving and search

But the real world is usually **more complex!**

- ▶ Resources are limited (and **costs** become important!)
- ▶ We have **constraints** and **restrictions**
- ▶ We need to be quick and cannot freely **explore**

Applications

Problem solving and search

The Travelling Salesperson Problem: find shortest route visiting each location once and returns to initial location.

- ▶ For example: Delivery services



Applications

Problem solving and search

The Travelling Salesperson Problem: find shortest route visiting each location once and returns to initial location.

- ▶ For example: Delivery services
- ▶ The Holidays in Romania example



Applications

Problem solving and search

The Travelling Salesperson Problem: find shortest route visiting each location once and returns to initial location.

- ▶ For example: Delivery services (*and you can always make it more complicated!*)
 - ▶ Time windows
 - ▶ Closed roads
 - ▶ Traffic
- ▶ The Holidays in Romania example



Applications

Problem solving and search

Assembly problems: find an order for assembling the parts of some object.

- ▶ For example: Manufacturing and design
(and you can always make it more complicated!)
 - ▶ Find the **optimal** order (minimum cost)
 - ▶ Reduce idle time on different machines
 - ▶ Assembly lines could be **dependent** on each other



Section 2

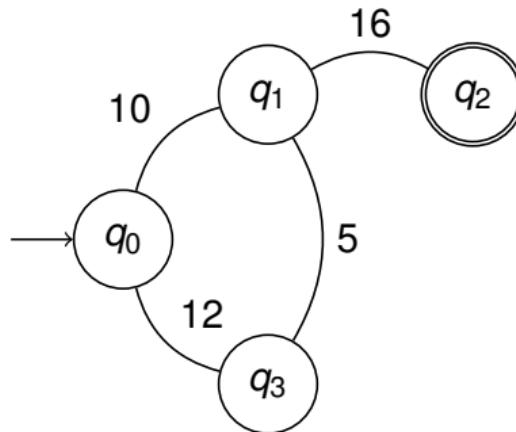
The search algorithm

What is a search algorithm?

Search algorithms

It is a **function** of the form $\text{Search}(\text{PROBLEM})$ that returns either a solution or failure.

- ▶ A **state** is a *representation of* a configuration
- ▶ Using a **state space graph** we can represent all possible states, and the transitions between them.

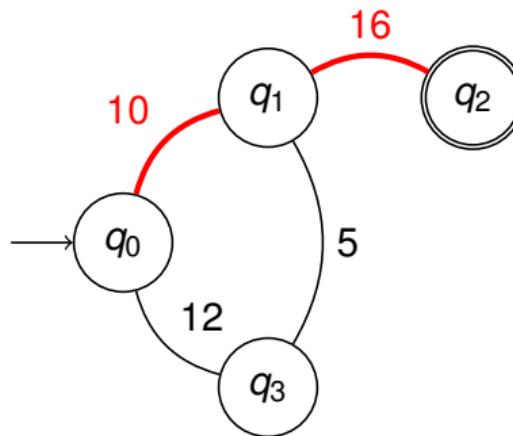


What is a search algorithm?

Search algorithms

It is a **function** of the form $\text{Search}(\text{PROBLEM})$ that returns either a solution or failure.

- ▶ A **state** is a *representation of* a configuration
- ▶ Using a **state space graph** we can represent all possible states, and the transitions between them.
- ▶ We can superimpose a **search tree** on the **space graph** and show a particular algorithm!



Exploring the state space I

Search algorithms

- ▶ Most of the time, it is not feasible (or it is too expensive) to build and represent the entire state graph.
- ▶ The problem solver agent generates a solution by **incrementally exploring** a small portion of the graph
- ▶ We **simulate** the exploration by generating **successors** of **already-explored states**.

Exploring the state space II

Search algorithms

The search procedure

1. You are standing on the initial node. What are the nodes to be explored here?
2. Is any of the nodes able to be explored, the goal? If not, generate successors of a node: **expand** the node²
3. Add the successors nodes into the list of “to be explored”.
4. Select (according to certain **criteria**) the next node to expand.

This process will be **repeated** until we either **find a solution**, or **fail** (by running out of time, of nodes, of resources...)

²Consider that the ‘goal check’ is dependent on the algorithm!

Exploring the state space

Search algorithms

The search procedure revised

You are standing on the starting node.

1. Check where you are standing: is it the goal?³
2. If not, then what are the nodes to be explored here?.
3. **Expand** the node you are in
4. Add the successors nodes into the **frontier**
5. Select (according to certain **criteria**—a function f) the next node to expand and move.

And then repeat!

³Consider that the ‘goal check’ is dependent on the algorithm!

What is a node?

Search algorithms

A **node** is a *representation* of a **state**. It is a data structure constituting a **part of a search tree**:

- ▶ The **state** of the node
- ▶ The **parent** of the node (or which state did you come from)
- ▶ The **children** of the node (or which states you can go to)
- ▶ The **path cost** of the search (at this point)

Notice how a node is not a state, but a step in the search!

Terminology and the book I

Search algorithms

If a state is in the frontier, it does not mean it has been expanded! At least not for our book.

- ▶ The **frontier** are those nodes *I can expand*
- ▶ The set of **reached** states contains both the frontier AND the **expanded** nodes

So, formally, we know that

- ▶ $\text{Frontier} \subset \text{Reached}$, and
- ▶ $\text{Frontier} \cup \text{Expanded} = \text{Reached}$

And so, $\text{Expanded} = \text{Reached} \setminus \text{Frontier}$.

Terminology and the book II

Search algorithms

The book also uses *object-oriented programming* notation to refer to *pertaining* (or *belonging*):

- ▶ node.STATE is the *STATE* of node
- ▶ node.PARENT is the *PARENT* of node ...

Operations are usually referred to as **functions**.

- ▶ $\text{Search}(\text{problem})$ is the *Search* procedure on the instance problem
- ▶ $\text{IsEmpty}(\text{frontier})$ is a function which returns true if the frontier is empty
- ▶ $\text{Pop}(\text{frontier})$ removes the top node of the frontier and returns it, while $\text{Top}(\text{frontier})$ just *peeks* at it (no removal)
- ▶ $\text{Add}(\text{node}, \text{frontier})$...

You get the idea.

Graph properties

Search algorithms

As many other graphs, search graphs and trees can contain **redundant paths** and **loops**. One can check the chain of parent nodes and make sure not to visit the same node twice. Coding is very different from the theoretical analysis we will do in the course.

The **performance** of a search algorithm can be measured in different ways:

- ▶ **Completeness:** is the algorithm guaranteed to find a solution?
- ▶ **Optimality:** the solution quality. Is it optimal? (cheaper, faster, etc.)
- ▶ **Time complexity:** how long does the algorithm take? (in seconds, operations, expanded states...)
- ▶ **Space complexity:** how much memory do we need, for example, in the *frontier* or *reached* sets?

Section 3

Uninformed search strategies

Searching with no information

Uninformed search strategies

Recall the third step in the searching procedure:

The searching procedure revised

... 5. Select (according to certain **criteria**—a function f) the next node to expand and move.

Depending on the type selection criteria and storage used, search strategies work differently!

Searching with no information

Uninformed search strategies

Recall the third step in the searching procedure:

The searching procedure revised

... 5. Select (according to certain **criteria**—a function f) the next node to expand and move.

Depending on the type selection criteria and storage used, search strategies work differently!

- ▶ Breadth-first search (BFS)

Searching with no information

Uninformed search strategies

Recall the third step in the searching procedure:

The searching procedure revised

... 5. Select (according to certain **criteria**—a function f) the next node to expand and move.

Depending on the type selection criteria and storage used, search strategies work differently!

- ▶ Breadth-first search (BFS)
- ▶ Depth-first search (DFS)

Searching with no information

Uninformed search strategies

Recall the third step in the searching procedure:

The searching procedure revised

... 5. Select (according to certain **criteria**—a function f) the next node to expand and move.

Depending on the type selection criteria and storage used, search strategies work differently!

- ▶ Breadth-first search (BFS)
- ▶ Depth-first search (DFS)
- ▶ Depth-limited search

Searching with no information

Uninformed search strategies

Recall the third step in the searching procedure:

The searching procedure revised

... 5. Select (according to certain **criteria**—a function f) the next node to expand and move.

Depending on the type selection criteria and storage used, search strategies work differently!

- ▶ Breadth-first search (BFS)
- ▶ Depth-first search (DFS)
- ▶ Depth-limited search
- ▶ Iterative deepening

Searching with no information

Uninformed search strategies

Recall the third step in the searching procedure:

The searching procedure revised

... 5. Select (according to certain **criteria**—a function f) the next node to expand and move.

Depending on the type selection criteria and storage used, search strategies work differently!

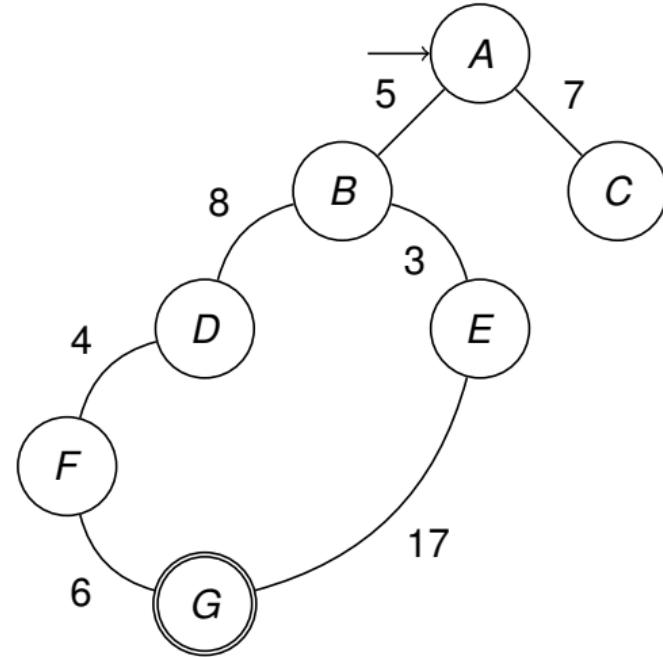
- ▶ Breadth-first search (BFS)
- ▶ Depth-first search (DFS)
- ▶ Depth-limited search
- ▶ Iterative deepening
- ▶ Uniform-cost (Dijkstra)

Breadth First Search

Uninformed search strategies

BFS prioritises old nodes first, and newly discovered ones last (hence the name, as it explores by *breadth* first)

- ▶ The *frontier* is a queue, i.e., “First In, First Out” (FIFO).
- ▶ Start at A and Goal is G.
- 1. Add A to frontier and solution.



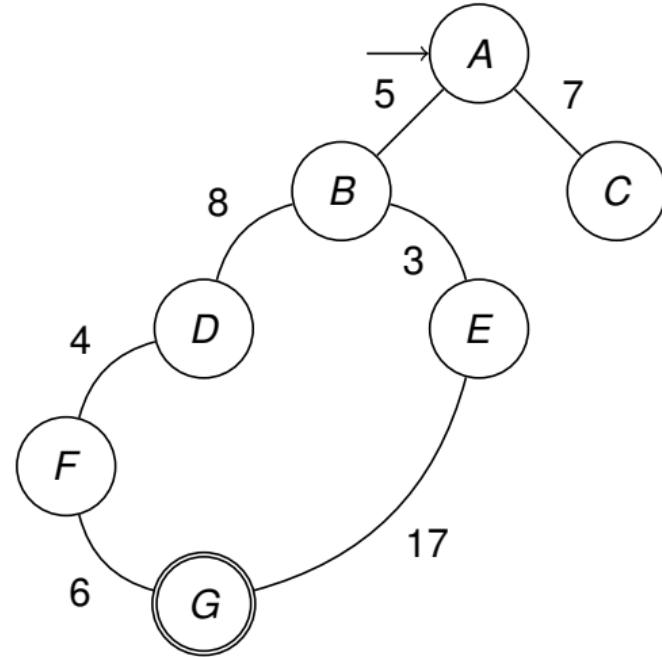
^ain BFS, we check early for the goal!

Breadth First Search

Uninformed search strategies

BFS prioritises old nodes first, and newly discovered ones last (hence the name, as it explores by *breadth* first)

- ▶ The *frontier* is a queue, i.e., “First In, First Out” (FIFO).
 - ▶ Start at A and Goal is G.
1. Add A to frontier and solution.
 2. A is not goal, so $\text{Expand}(A)$



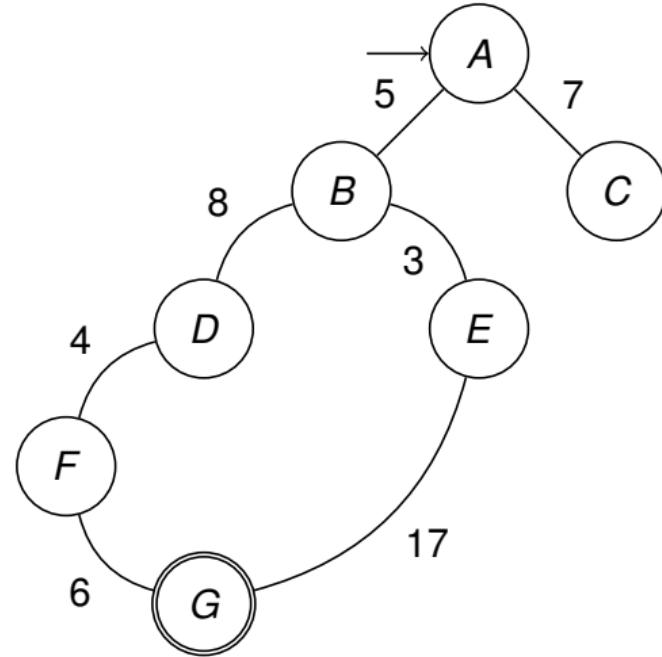
^ain BFS, we check early for the goal!

Breadth First Search

Uninformed search strategies

BFS prioritises old nodes first, and newly discovered ones last (hence the name, as it explores by *breadth* first)

- ▶ The *frontier* is a queue, i.e., “First In, First Out” (FIFO).
 - ▶ Start at A and Goal is G.
1. Add A to frontier and solution.
 2. A is not goal, so $\text{Expand}(A)$
 3. Add successors to *frontier*:



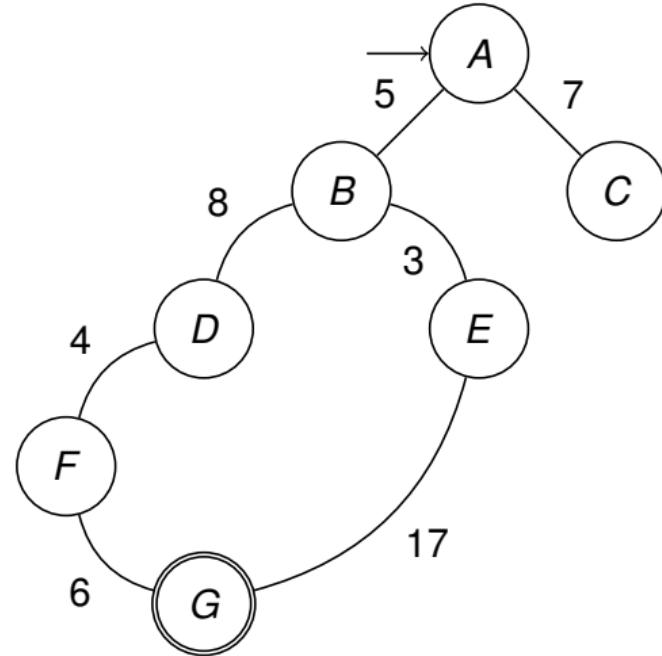
^ain BFS, we check early for the goal!

Breadth First Search

Uninformed search strategies

BFS prioritises old nodes first, and newly discovered ones last (hence the name, as it explores by *breadth* first)

- ▶ The *frontier* is a queue, i.e., “First In, First Out” (FIFO).
 - ▶ Start at A and Goal is G.
1. Add A to frontier and solution.
 2. A is not goal, so $\text{Expand}(A)$
 3. Add successors to *frontier*:
 - ▶ $\text{frontier} = \langle B, C \rangle$



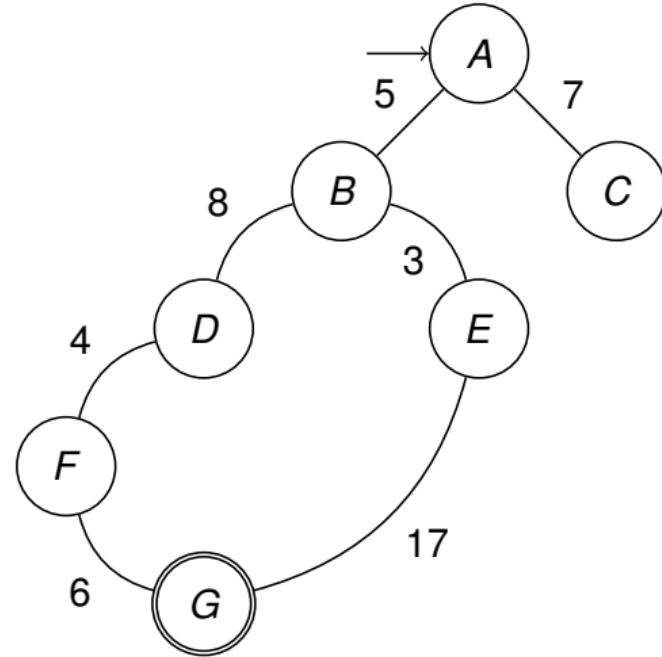
^ain BFS, we check early for the goal!

Breadth First Search

Uninformed search strategies

BFS prioritises old nodes first, and newly discovered ones last (hence the name, as it explores by *breadth* first)

- ▶ The *frontier* is a queue, i.e., “First In, First Out” (FIFO).
 - ▶ Start at A and Goal is G.
1. Add A to frontier and solution.
 2. A is not goal, so $\text{Expand}(A)$
 3. Add successors to *frontier*:
 - ▶ $\text{frontier} = \langle B, C \rangle$
 - ▶ Is any of those the goal?^a



^ain BFS, we check early for the goal!

Breadth First Search

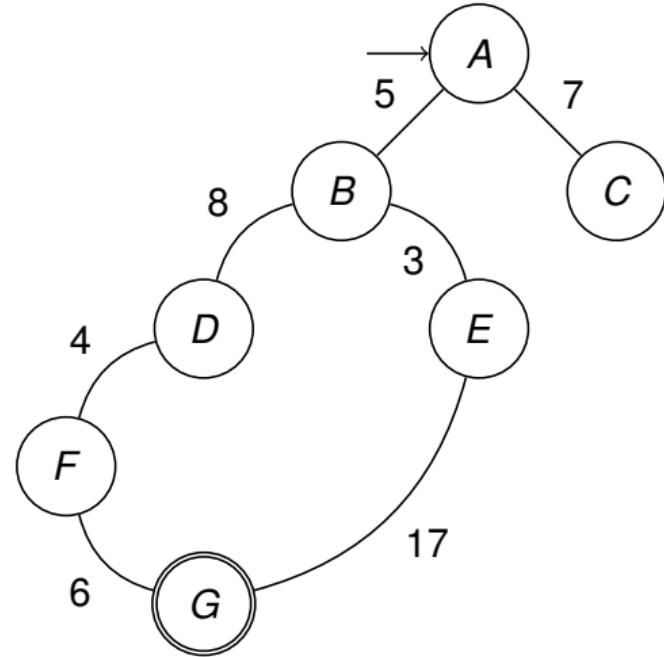
Uninformed search strategies

BFS prioritises old nodes first, and newly discovered ones last (hence the name, as it explores by *breadth* first)

- ▶ The *frontier* is a queue, i.e., “First In, First Out” (FIFO).
 - ▶ Start at A and Goal is G.
1. Add A to frontier and solution.
 2. A is not goal, so $\text{Expand}(A)$
 3. Add successors to *frontier*:
 - ▶ $\text{frontier} = \langle B, C \rangle$
 - ▶ Is any of those the goal?^a
 4. Choose *first element* in *frontier*.

And repeat...

^ain BFS, we check early for the goal!



Breadth First Search

Uninformed search strategies

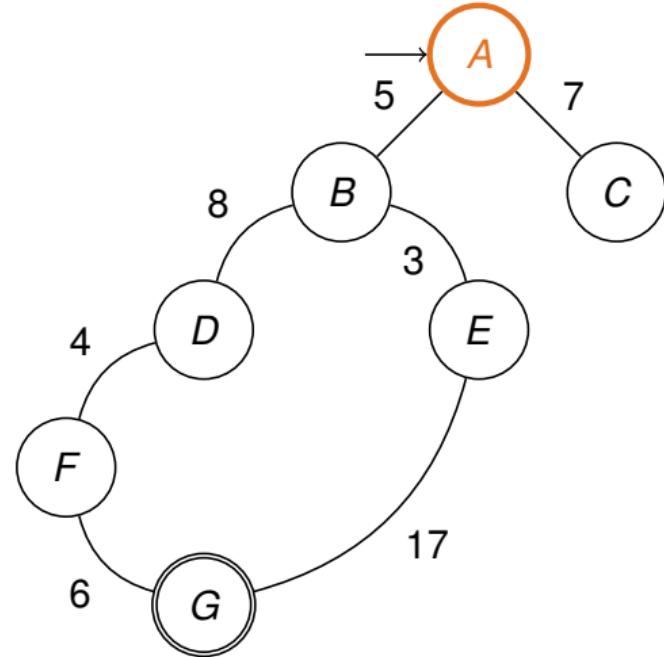
BFS prioritises old nodes first, and newly discovered ones last (hence the name, as it explores by *breadth* first)

- ▶ The *frontier* is a queue, i.e., “First In, First Out” (FIFO).

- ▶ Start at A and Goal is G.

1. Add A to *frontier* and solution.
2. A is not goal, so $\text{Expand}(A)$
3. Add successors to *frontier*:
 - ▶ $\text{frontier} = \langle B, C \rangle$. No goals.
4. Choose *first element* in *frontier*.

And repeat...



Breadth First Search

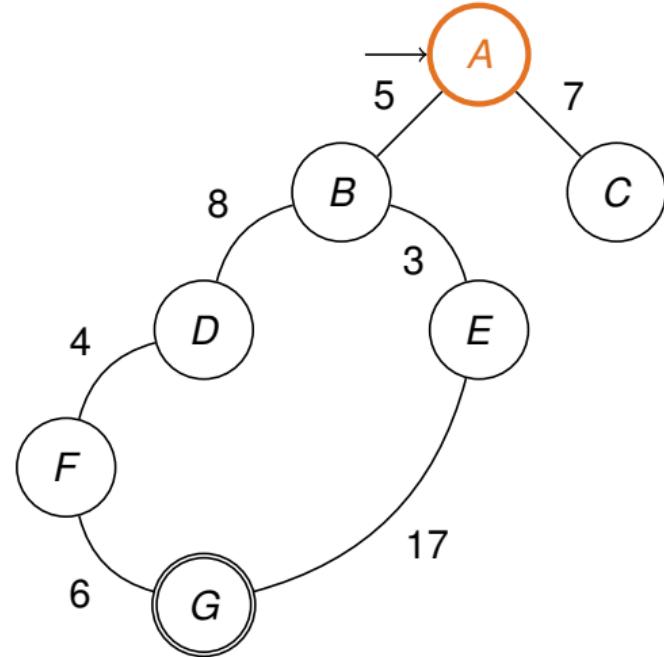
Uninformed search strategies

BFS prioritises old nodes first, and newly discovered ones last (hence the name, as it explores by *breadth* first)

- ▶ The *frontier* is a queue, i.e., “First In, First Out” (FIFO).

- ▶ Start at A and Goal is G.
1. Add A to frontier and solution.
 2. A is not goal, so *Expand(A)*
 3. Add successors to *frontier*:
 - ▶ $\text{frontier} = \langle B, C \rangle$. No goals.
 4. Choose *first element* in *frontier*.

And repeat...



Breadth First Search

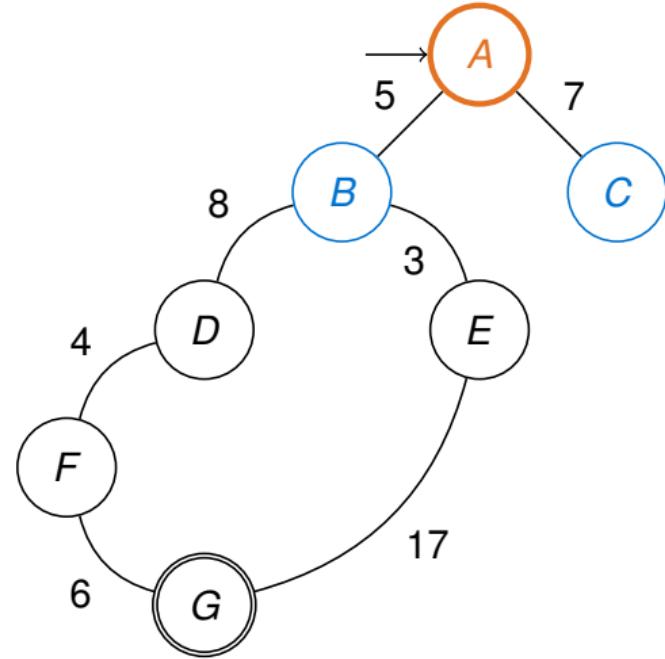
Uninformed search strategies

BFS prioritises old nodes first, and newly discovered ones last (hence the name, as it explores by *breadth* first)

- ▶ The *frontier* is a queue, i.e., “First In, First Out” (FIFO).

- ▶ Start at A and Goal is G.
1. Add A to frontier and solution.
 2. A is not goal, so $\text{Expand}(A)$
 3. **Add successors to *frontier*:**
 - ▶ $\text{frontier} = \langle B, C \rangle$. No goals.
 4. Choose *first element* in *frontier*.

And repeat...



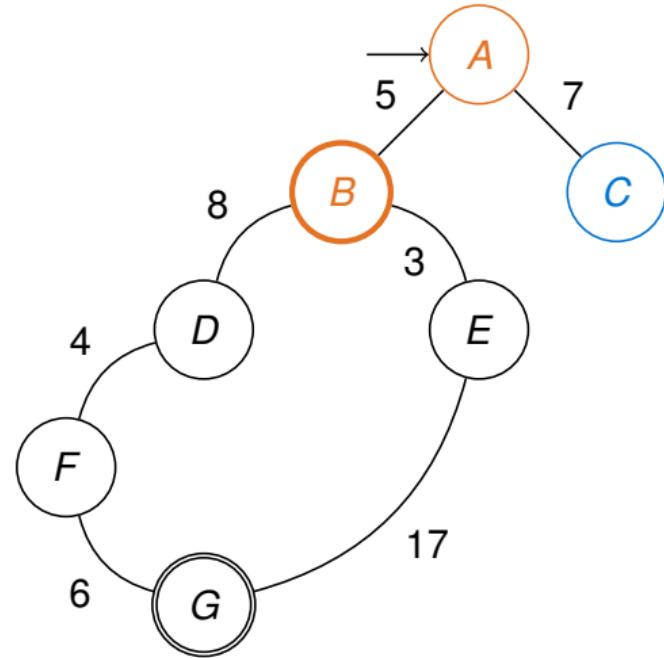
Breadth First Search

Uninformed search strategies

BFS prioritises old nodes first, and newly discovered ones last (hence the name, as it explores by *breadth* first)

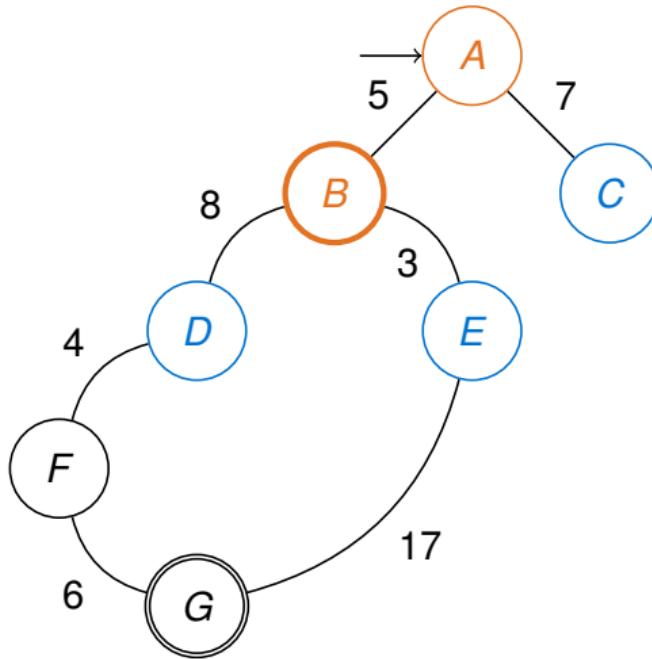
- ▶ The *frontier* is a queue, i.e., “First In, First Out” (FIFO).
 - ▶ Start at A and Goal is G.
1. Add A to frontier and solution.
 2. A is not goal, so $\text{Expand}(A)$
 3. Add successors to *frontier*:
 - ▶ $\text{frontier} = \langle B, C \rangle$. No goals.
 4. Choose first element in *frontier*.

And repeat...



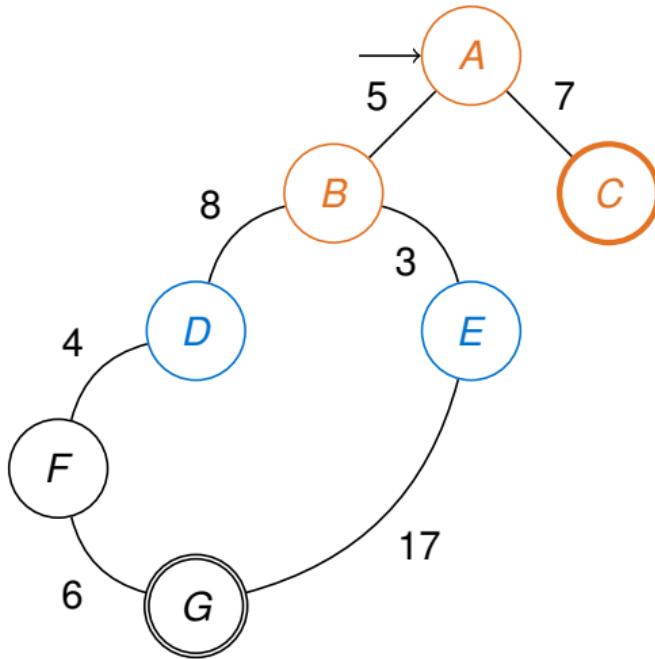
Breadth First Search

Uninformed search strategies



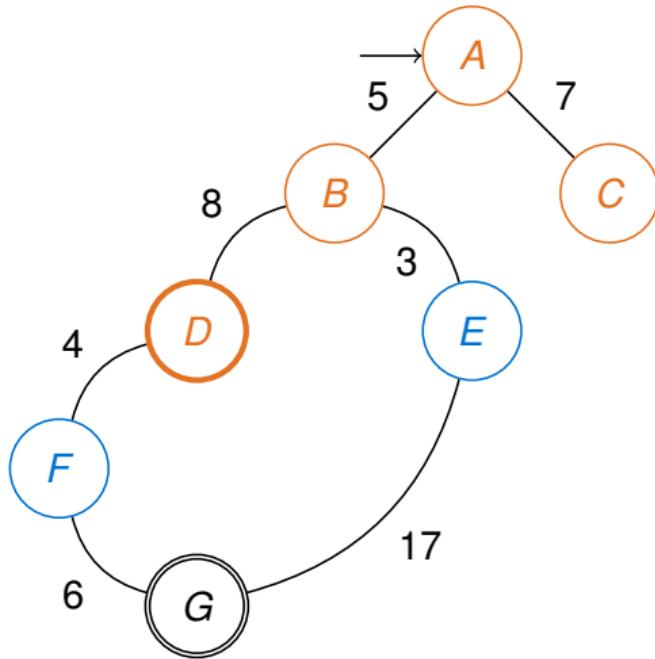
Breadth First Search

Uninformed search strategies



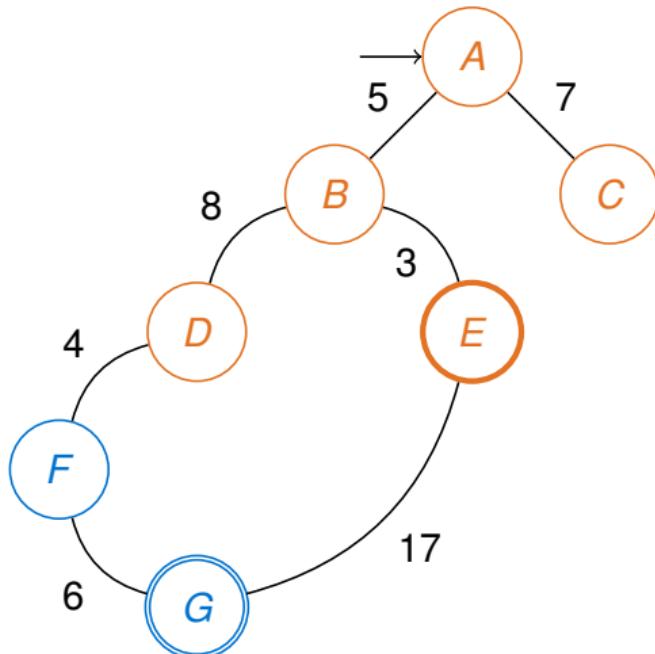
Breadth First Search

Uninformed search strategies



Breadth First Search

Uninformed search strategies

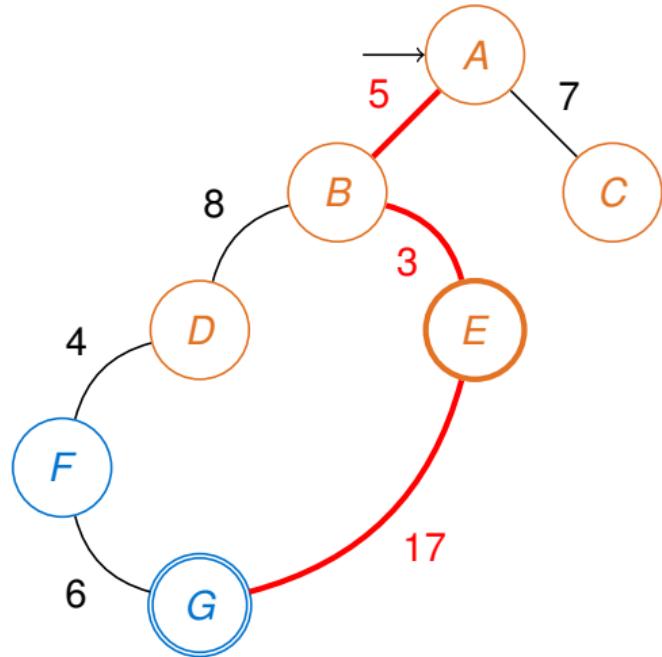


Breadth First Search

Uninformed search strategies

- ▶ We have **seen** the goal!^a
- ▶ We can reconstruct the solution by creating a *chain of parents* from the goal

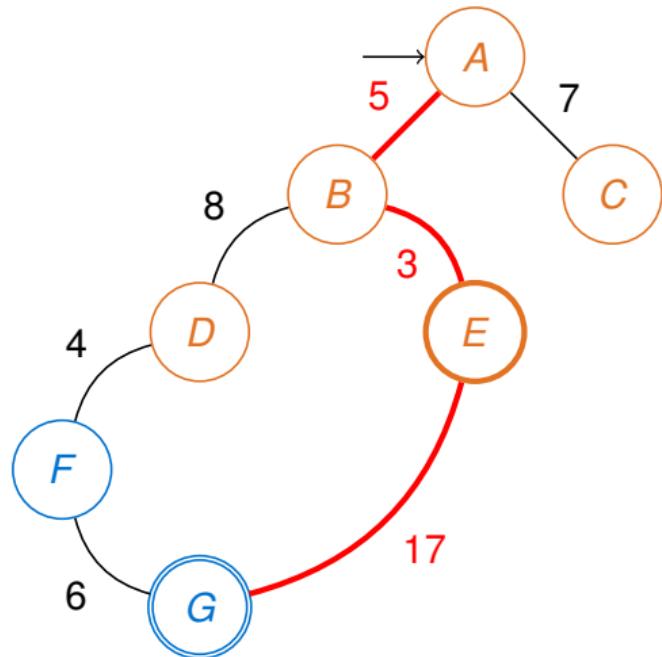
^aRemember we check for goal when adding to the frontier in BFS!



Breadth First Search

Uninformed search strategies

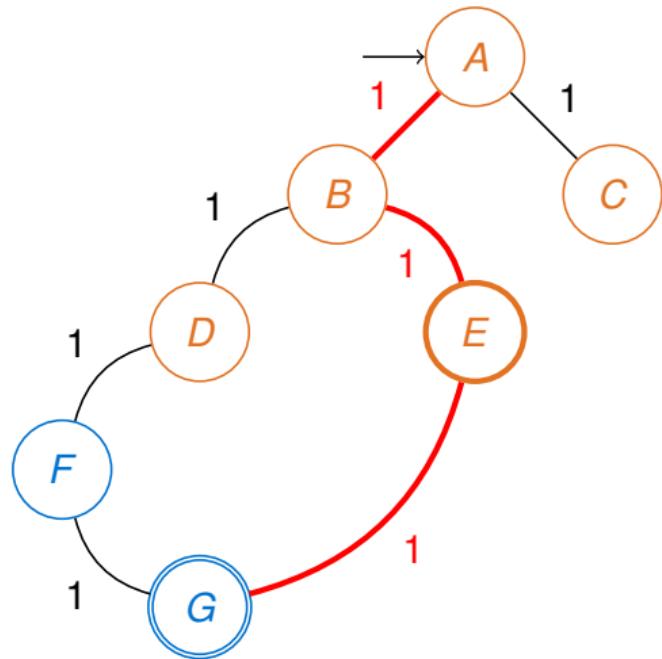
- ▶ **Not optimal**, unless all costs were equal!



Breadth First Search

Uninformed search strategies

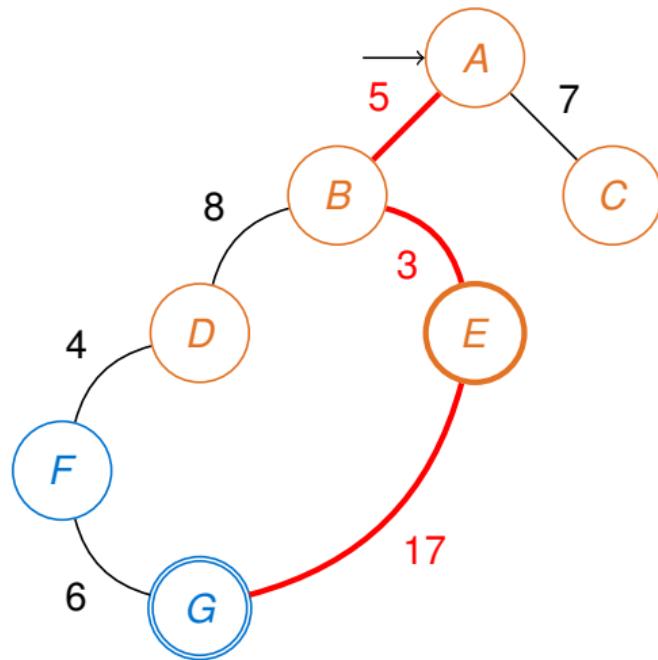
- ▶ **Not optimal**, unless all costs were equal!
- ▶ Like so!



Breadth First Search

Uninformed search strategies

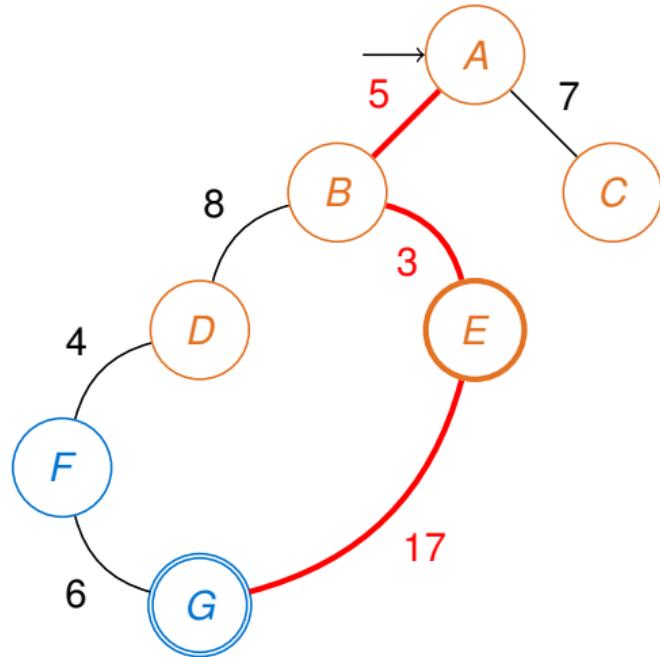
- ▶ **Not optimal**, unless all costs were equal!
- ▶ Like so!
- ▶ **Complete**: always finds a solution if space state is finite



Breadth First Search

Uninformed search strategies

- ▶ **Not optimal**, unless all costs were equal!
- ▶ Like so!
- ▶ **Complete**: always finds a solution if space state is finite
- ▶ Time and space complexity is insane $\mathcal{O}(b^d)$ where b is the branching factor (number of successors to consider) and d is the depth of the shallowest solution.



This was a very detailed explanation. The following algorithms will be **summarised**.

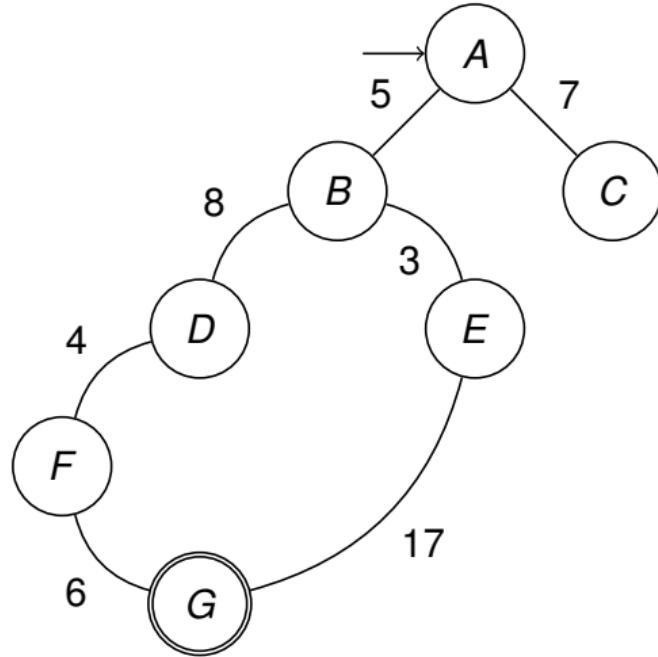
Check your book for the step by step strategies!

Depth First Search

Uninformed search strategies

DFS prioritises **new nodes first**, and previously discovered ones go last (hence the name, as it explores by *depth* first)

- ▶ The *frontier* is a **stack**, i.e., “Last In, First Out” (LIFO)
 - ▶ Start at A and Goal is G, as before
1. Add A to frontier and solution.

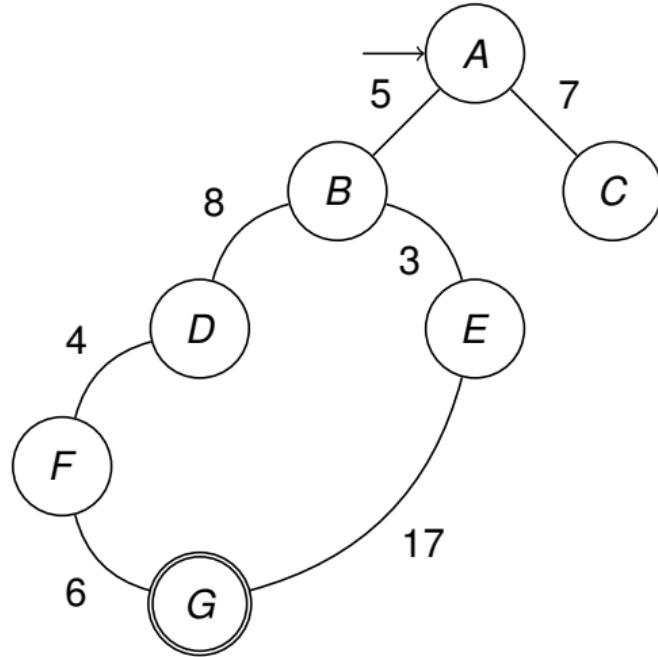


Depth First Search

Uninformed search strategies

DFS prioritises **new nodes first**, and previously discovered ones go last (hence the name, as it explores by *depth* first)

- ▶ The *frontier* is a **stack**, i.e., “Last In, First Out” (LIFO)
- ▶ Start at A and Goal is G, as before
 1. Add A to frontier and solution.
 2. A is not goal, so $\text{Expand}(A)$

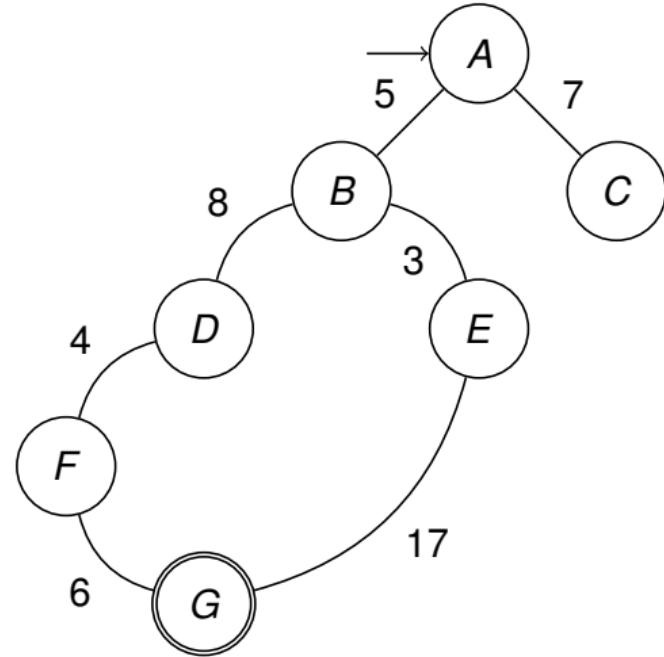


Depth First Search

Uninformed search strategies

DFS prioritises **new nodes first**, and previously discovered ones go last (hence the name, as it explores by *depth* first)

- ▶ The *frontier* is a **stack**, i.e., “Last In, First Out” (LIFO)
- ▶ Start at A and Goal is G, as before
 1. Add A to frontier and solution.
 2. A is not goal, so *Expand(A)*
 3. Add successors to *frontier* (in reverse order):

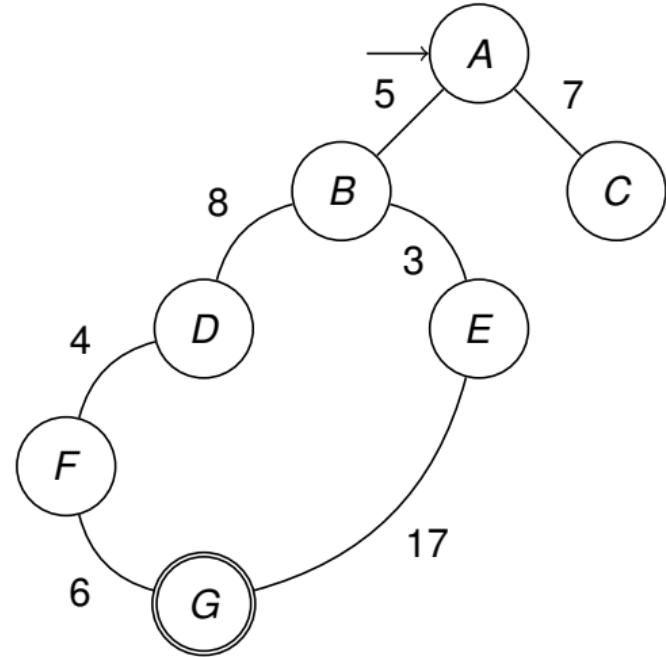


Depth First Search

Uninformed search strategies

DFS prioritises **new nodes first**, and previously discovered ones go last (hence the name, as it explores by *depth* first)

- ▶ The *frontier* is a **stack**, i.e., “Last In, First Out” (LIFO)
- ▶ Start at A and Goal is G, as before
- 1. Add A to frontier and solution.
- 2. A is not goal, so $\text{Expand}(A)$
- 3. Add successors to *frontier* (in reverse order):
 - ▶ $\text{frontier} = \langle B, C \rangle$

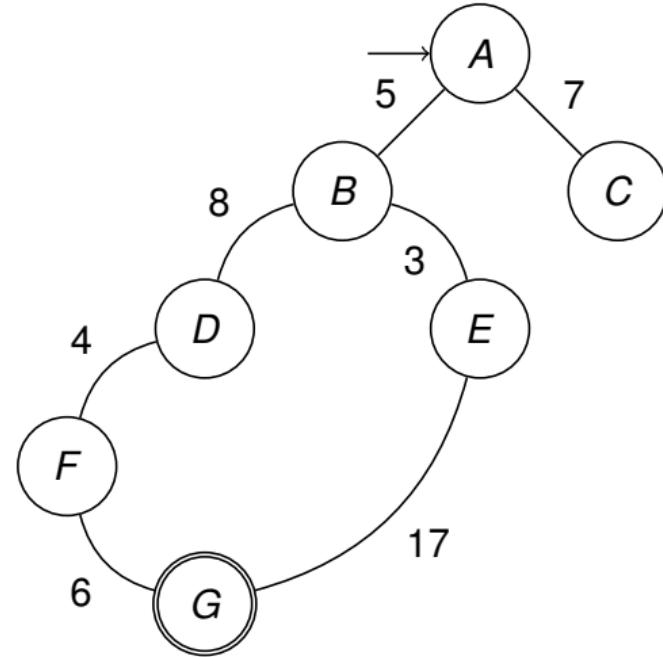


Depth First Search

Uninformed search strategies

DFS prioritises **new nodes first**, and previously discovered ones go last (hence the name, as it explores by *depth* first)

- ▶ The *frontier* is a **stack**, i.e., “Last In, First Out” (LIFO)
 - ▶ Start at A and Goal is G, as before
1. Add A to frontier and solution.
 2. A is not goal, so $\text{Expand}(A)$
 3. Add successors to *frontier* (in reverse order):
 - ▶ $\text{frontier} = \langle B, C \rangle$
 - ▶ Frontier now grows towards left!

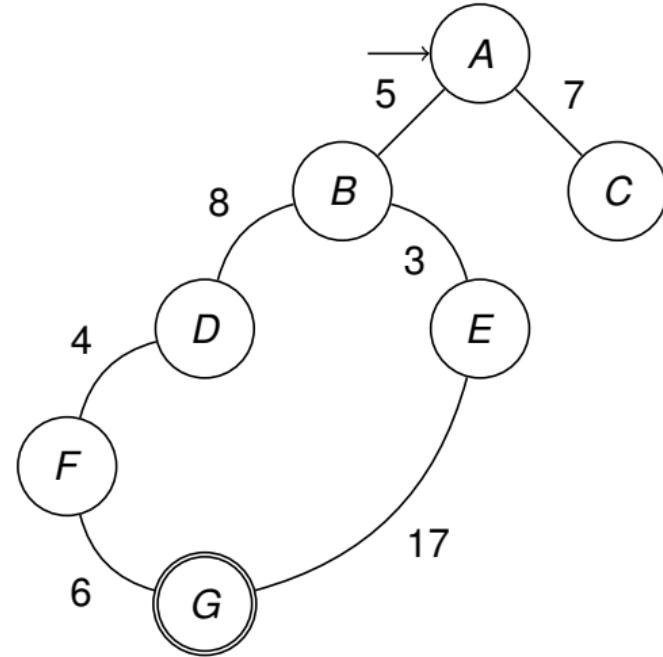


Depth First Search

Uninformed search strategies

DFS prioritises **new nodes first**, and previously discovered ones go last (hence the name, as it explores by *depth* first)

- ▶ The *frontier* is a **stack**, i.e., “Last In, First Out” (LIFO)
 - ▶ Start at A and Goal is G, as before
1. Add A to frontier and solution.
 2. A is not goal, so $\text{Expand}(A)$
 3. Add successors to *frontier* (in reverse order):
 - ▶ $\text{frontier} = \langle B, C \rangle$
 - ▶ Frontier now grows towards left!
 4. Choose **first element** in *frontier*.



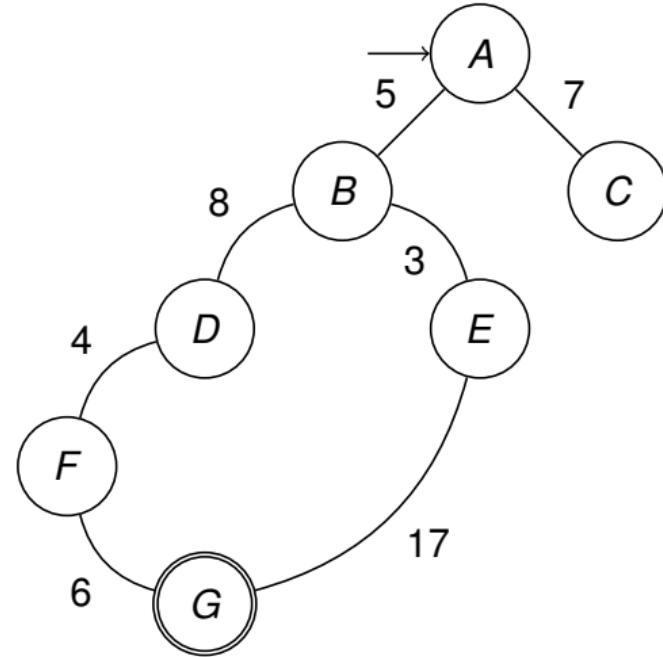
Depth First Search

Uninformed search strategies

DFS prioritises **new nodes first**, and previously discovered ones go last (hence the name, as it explores by *depth* first)

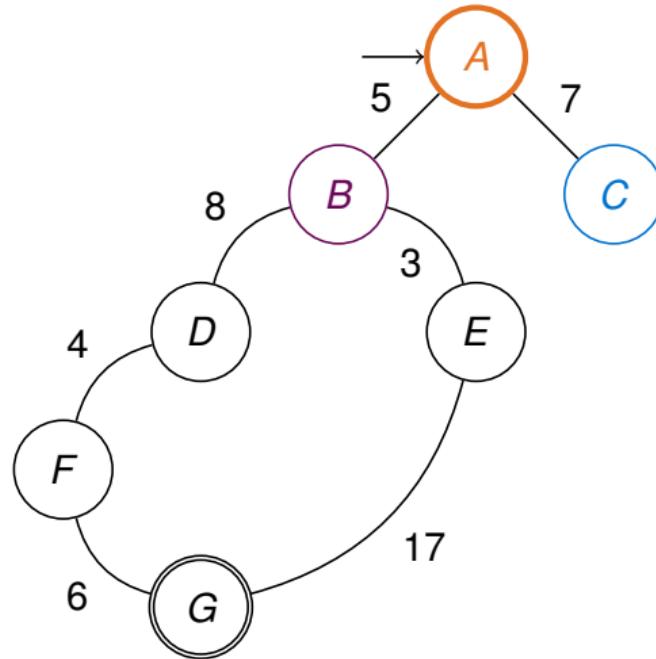
- ▶ The *frontier* is a **stack**, i.e., “Last In, First Out” (LIFO)
- ▶ Start at A and Goal is G, as before
- 1. Add A to frontier and solution.
- 2. A is not goal, so *Expand(A)*
- 3. Add successors to *frontier* (in reverse order):
 - ▶ $\text{frontier} = \langle B, C \rangle$
 - ▶ Frontier now grows towards left!
- 4. Choose **first element** in *frontier*.

And repeat...



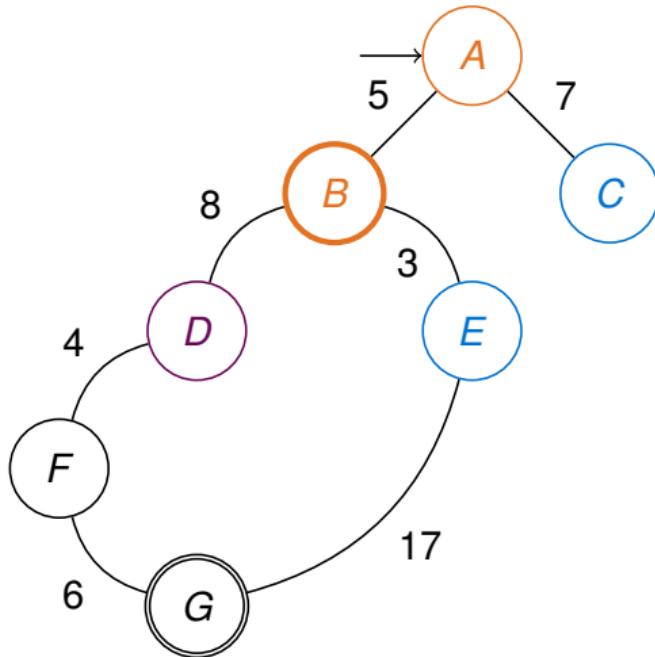
Depth First Search

Uninformed search strategies



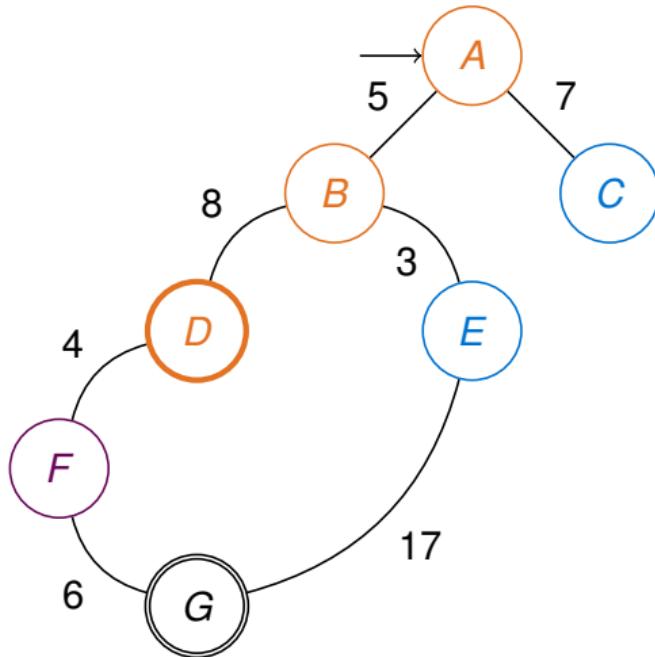
Depth First Search

Uninformed search strategies



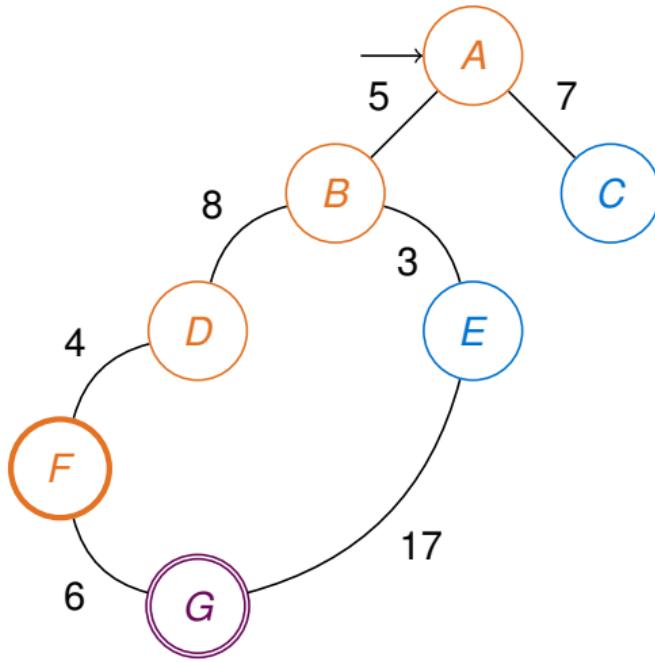
Depth First Search

Uninformed search strategies



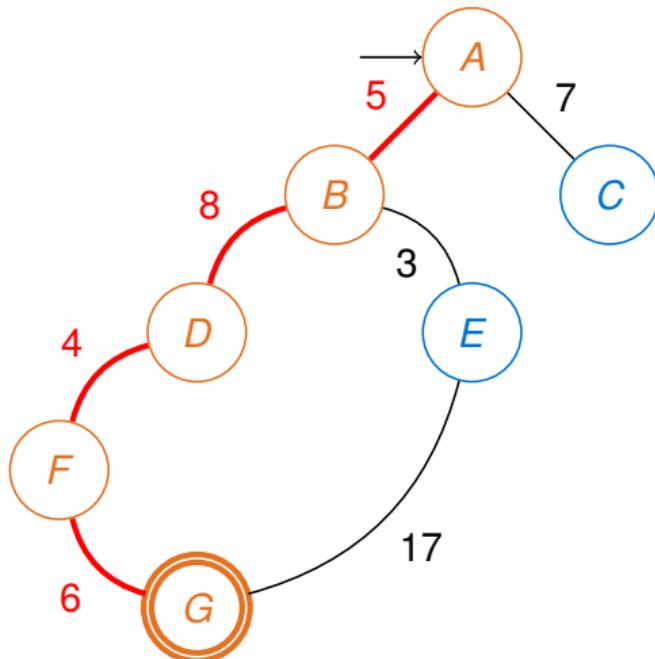
Depth First Search

Uninformed search strategies



Depth First Search

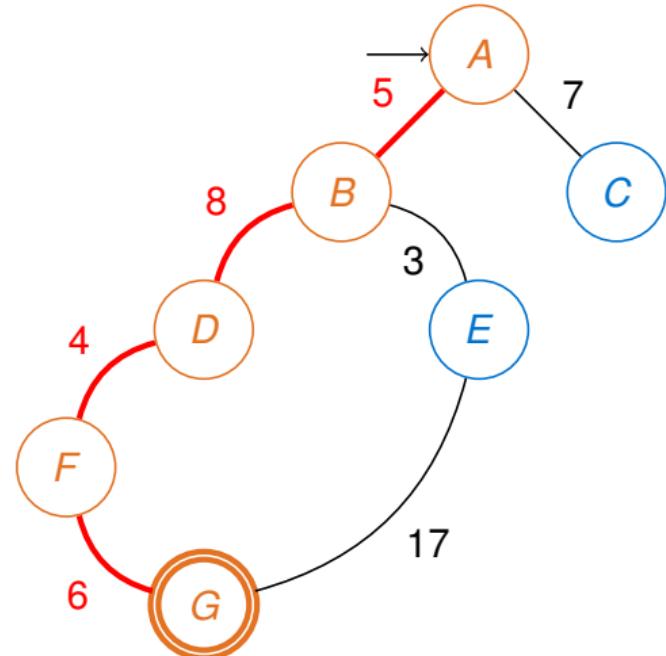
Uninformed search strategies



Depth First Search

Uninformed search strategies

- ▶ **Not always optimal**

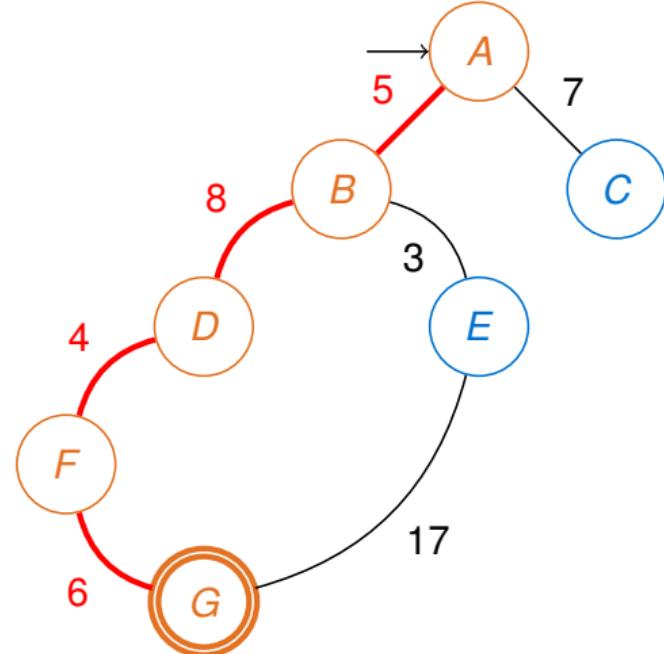


^abecause it is usually implemented as tree search

Depth First Search

Uninformed search strategies

- ▶ **Not always optimal**
- ▶ It returns the first solution found. We were lucky!

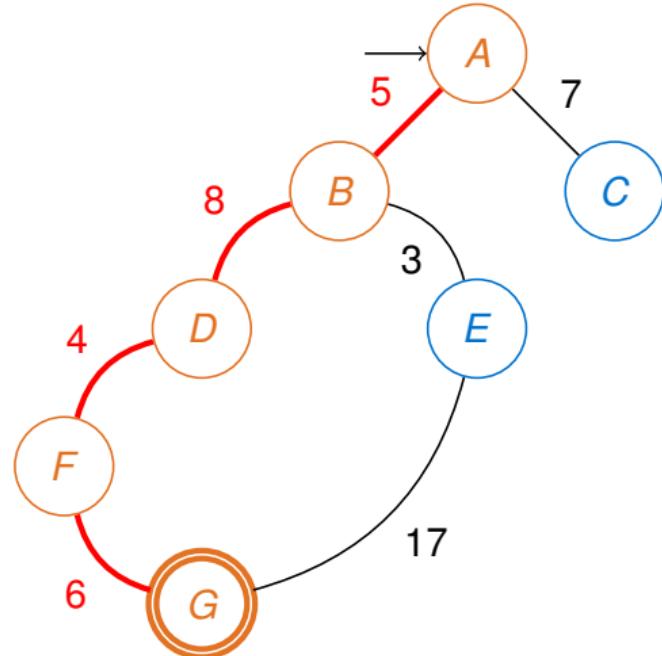


^abecause it is usually implemented as tree search

Depth First Search

Uninformed search strategies

- ▶ **Not always optimal**
- ▶ It returns the first solution found. We were lucky!
- ▶ **Not Complete:** fails in infinitely-deep spaces and spaces with loops^a

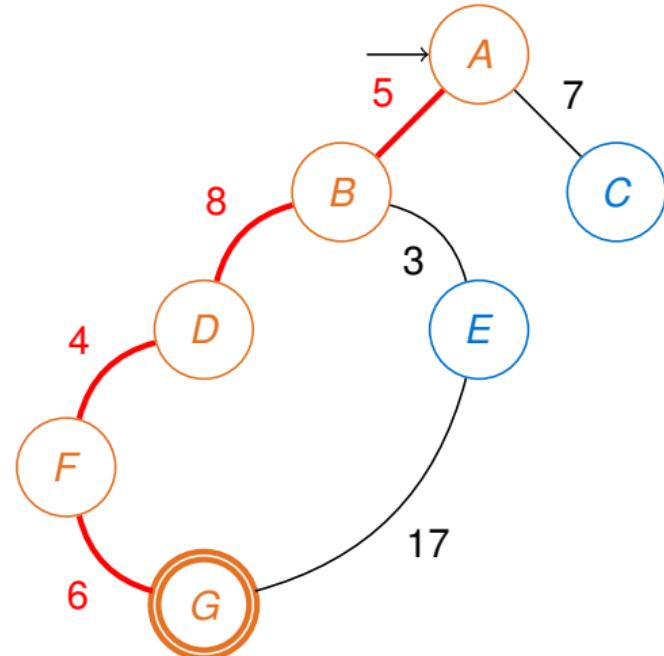


^abecause it is usually implemented as tree search

Depth First Search

Uninformed search strategies

- ▶ **Not always optimal**
- ▶ It returns the first solution found. We were lucky!
- ▶ **Not Complete:** fails in infinitely-deep spaces and spaces with loops^a
- ▶ Time complexity $\mathcal{O}(b^m)$, and space complexity is linear $\mathcal{O}(bm)$ where b is the branching factor and m is the maximum depth in the state space (tree version)

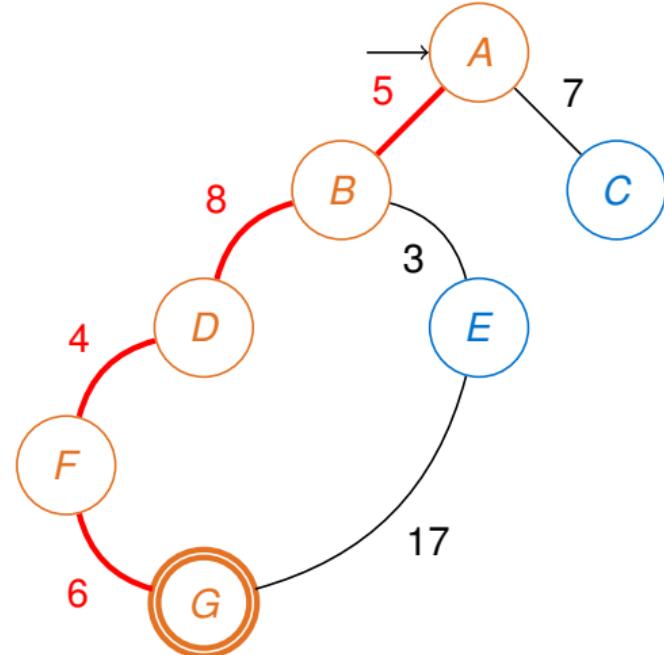


^abecause it is usually implemented as tree search

Depth First Search

Uninformed search strategies

- ▶ **Not always optimal**
- ▶ It returns the first solution found. We were lucky!
- ▶ **Not Complete:** fails in infinitely-deep spaces and spaces with loops^a
- ▶ Time complexity $\mathcal{O}(b^m)$, and space complexity is linear $\mathcal{O}(bm)$ where b is the branching factor and m is the maximum depth in the state space (tree version)
- ▶ One can make a smarter version of DFS with graph search (memory). Space complexity grows to exponential, and might still miss if on infinite spaces.



^abecause it is usually implemented as tree search

Depth-limited and Iterative deepening search

Uninformed search strategies

Two other ideas lie on imposing a limit on DFS, both as *tree search strategies*.

- ▶ Use DFS with $DepthLimit = 1$
- ▶ If no solution found, then try increasing the $DepthLimit$ *iteratively* until a set *cutoff*.

Depth-limited and Iterative deepening search

Uninformed search strategies

Two other ideas lie on imposing a limit on DFS, both as *tree search strategies*.

- ▶ Use DFS with $DepthLimit = 1$
- ▶ If no solution found, then try increasing the $DepthLimit$ *iteratively* until a set *cutoff*.
- ▶ Iterative deepening will try multiple levels and return either a **solution** if it exists, a **failure** if it does not, or a **cutoff**.
- ▶ A *cutoff* means the maximum depth we set previously was reached, so a solution might exist deeper than the levels we explored.

Depth-limited and Iterative deepening search

Uninformed search strategies

Two other ideas lie on imposing a limit on DFS, both as *tree search strategies*.

- ▶ Use DFS with $DepthLimit = 1$
- ▶ If no solution found, then try increasing the $DepthLimit$ *iteratively* until a set *cutoff*.
- ▶ Iterative deepening will try multiple levels and return either a **solution** if it exists, a **failure** if it does not, or a **cutoff**.
- ▶ A *cutoff* means the maximum depth we set previously was reached, so a solution might exist deeper than the levels we explored.
- ▶ **Always complete** if solution exists and state space is finite
- ▶ **Not cost optimal** unless costs are the same (like BFS)
- ▶ Time complexity: $\mathcal{O}(b^d)$
- ▶ Space complexity: $\mathcal{O}(bd)$ (like DFS)

Slightly better than both DFS and BFS!

Uninformed search strategies

- ▶ They systematically navigate the search space blindly—not questioning where the goal may be in the space.
- ▶ The search space is often very large.

Uninformed search strategies

- ▶ They systematically navigate the search space blindly—not questioning where the goal may be in the space.
- ▶ The search space is often very large.

Why not being *smarter* about it?

Section 4

Informed search strategies

Heuristic search

Informed search strategies

To take *better informed decisions*, we can use a domain-specific hint about how “desirable” a state can be.

Heuristic search

Informed search strategies

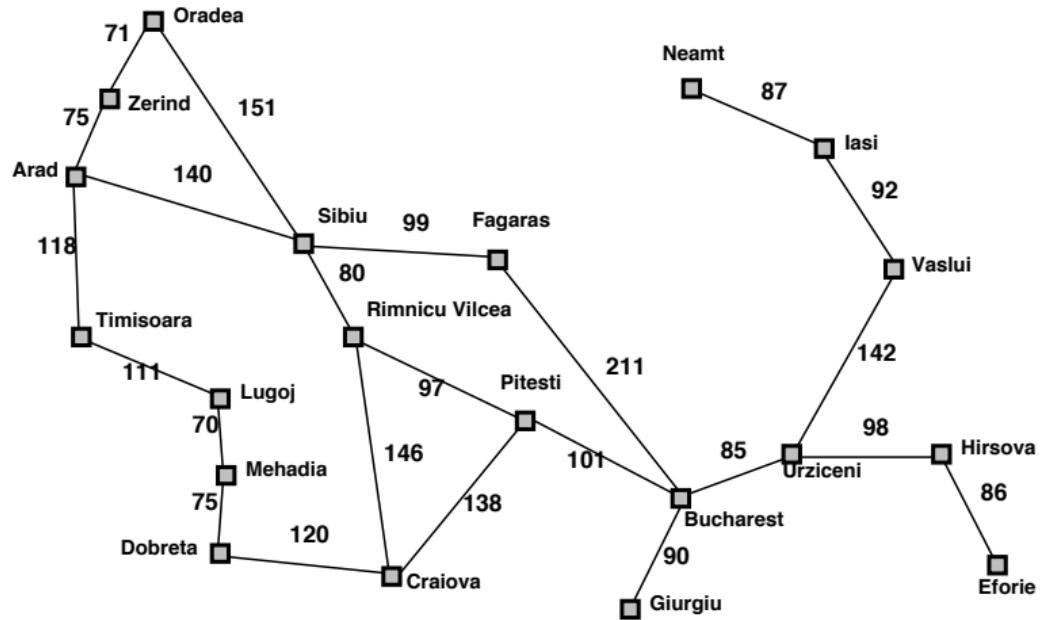
To take *better informed decisions*, we can use a domain-specific hint about how “desirable” a state can be.

This is usually done by using a **heuristic function** $h(n)$, where $h : S \rightarrow \mathbb{R}$, i.e., a *guessing function* about an estimated remaining cost to the goal.

Heuristic example: Romania

Informed search strategies

Using h as the straight line distance to goal:



(Greedy) Best First Search

Informed search strategies

Best first

Choose always the best of your expectations (cheapest estimate).

(Greedy) Best First Search

Informed search strategies

Best first

Choose always the best of your expectations (cheapest estimate).

Yet again, same idea:

(Greedy) Best First Search

Informed search strategies

Best first

Choose always the best of your expectations (cheapest estimate).

Yet again, same idea:

1. Start

(Greedy) Best First Search

Informed search strategies

Best first

Choose always the best of your expectations (cheapest estimate).

Yet again, same idea:

1. Start
2. Check for goal

(Greedy) Best First Search

Informed search strategies

Best first

Choose always the best of your expectations (cheapest estimate).

Yet again, same idea:

1. Start
2. Check for goal
3. Expand and update frontier

(Greedy) Best First Search

Informed search strategies

Best first

Choose always the best of your expectations (cheapest estimate).

Yet again, same idea:

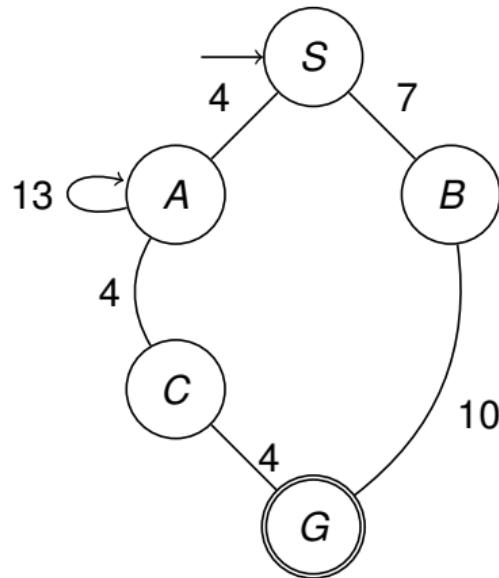
1. Start
2. Check for goal
3. Expand and update frontier
4. Choose **the best** of the estimates

(Greedy) Best First Search

Informed search strategies

With the following estimated distances to the goal:

- ▶ $h(A) = 3$
- ▶ $h(B) = 3$
- ▶ $h(C) = 3$
- ▶ $h(G) = 0$



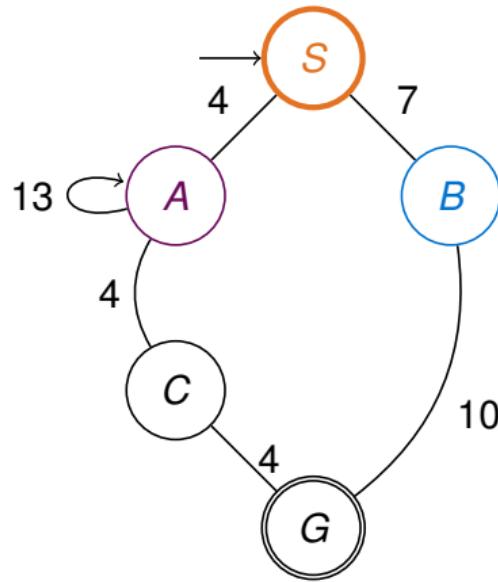
(Greedy) Best First Search

Informed search strategies

With the following estimated distances to the goal:

- ▶ $h(A) = 3$
- ▶ $h(B) = 3$
- ▶ $h(C) = 3$
- ▶ $h(G) = 0$

We choose alphabetically in case of a tie.



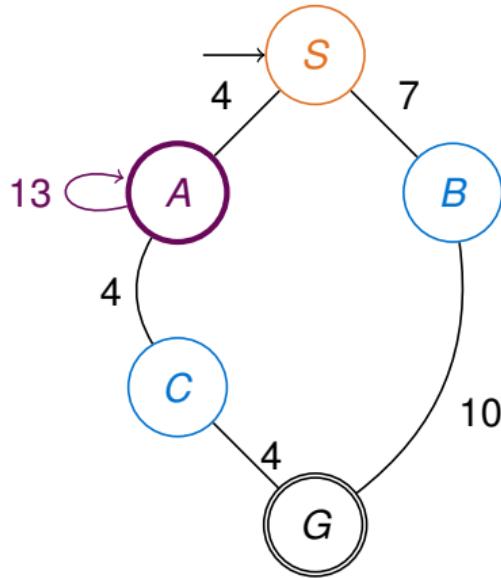
(Greedy) Best First Search

Informed search strategies

With the following estimated distances to the goal:

- ▶ $h(A) = 3$
- ▶ $h(B) = 3$
- ▶ $h(C) = 3$
- ▶ $h(G) = 0$

We choose alphabetically in case of a tie.



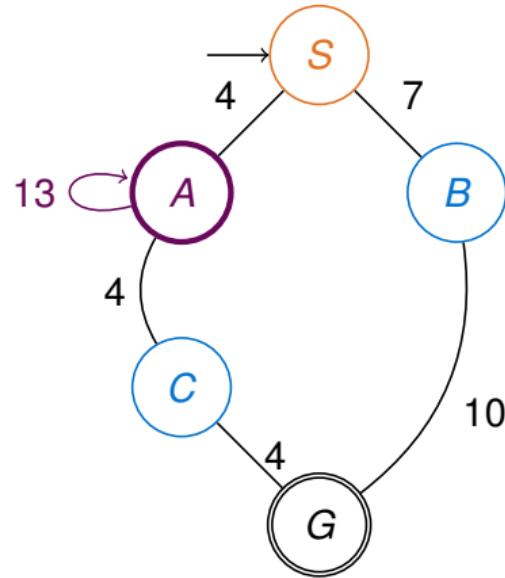
(Greedy) Best First Search

Informed search strategies

With the following estimated distances to the goal:

- ▶ $h(A) = 3$
- ▶ $h(B) = 3$
- ▶ $h(C) = 3$
- ▶ $h(G) = 0$

We choose alphabetically in case of a tie.



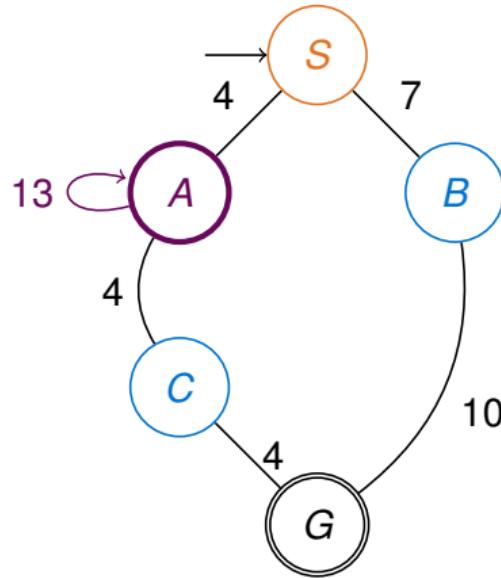
(Greedy) Best First Search

Informed search strategies

With the following estimated distances to the goal:

- ▶ $h(A) = 3$
- ▶ $h(B) = 3$
- ▶ $h(C) = 3$
- ▶ $h(G) = 0$

We choose alphabetically in case of a tie.

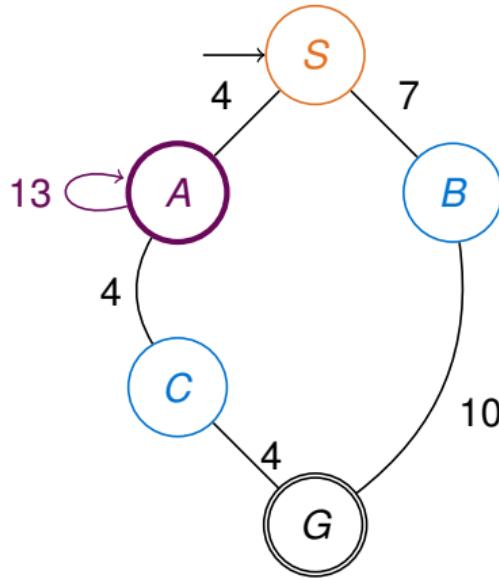


(Greedy) Best First Search

Informed search strategies

With those estimated distances to the goal:

- ▶ We have a cycle!
- ▶ Tree search would not make it past A
- ▶ By adding memory we make it smarter.
Still, space complexity increases.
- ▶ **Always Complete** in finite spaces with no loops (not our case)
- ▶ **Might not be optimal** (See Romania example!)



A^* search

Informed search strategies

What if we consider the cost *and* the heuristic?

A^* search

Informed search strategies

What if we consider the cost *and* the heuristic?

Our new **heuristic function** will consider both things:

$$f(n) = g(n) + h(n)$$

where

A^* search

Informed search strategies

What if we consider the cost *and* the heuristic?

Our new **heuristic function** will consider both things:

$$f(n) = g(n) + h(n)$$

where

- ▶ $g(n)$ is the cost we have paid so far to reach n

A^* search

Informed search strategies

What if we consider the cost *and* the heuristic?

Our new **heuristic function** will consider both things:

$$f(n) = g(n) + h(n)$$

where

- ▶ $g(n)$ is the cost we have paid so far to reach n
- ▶ $h(n)$ is the estimated cost of the node (to the goal)

*A** search

Informed search strategies

What if we consider the cost *and* the heuristic?

Our new **heuristic function** will consider both things:

$$f(n) = g(n) + h(n)$$

where

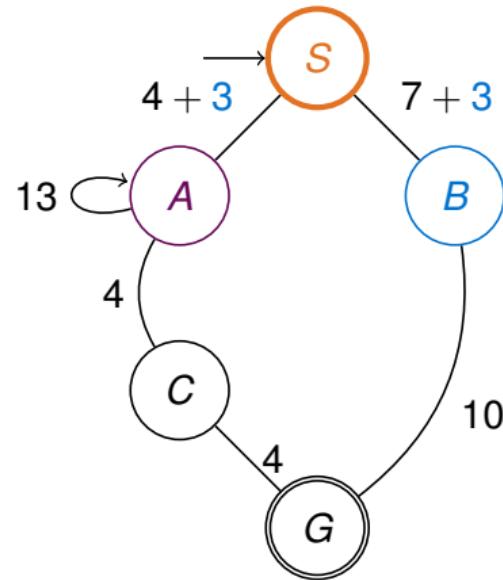
- ▶ $g(n)$ is the cost we have paid so far to reach n
- ▶ $h(n)$ is the estimated cost of the node (to the goal)
- ▶ $f(n)$ is then the estimated cost of the cheapest solution through n to the goal

A^* search

Informed search strategies

With the following estimated distances to the goal:

- ▶ $h(A) = 3$
- ▶ $h(B) = 3$
- ▶ $h(C) = 3$
- ▶ $h(G) = 0$

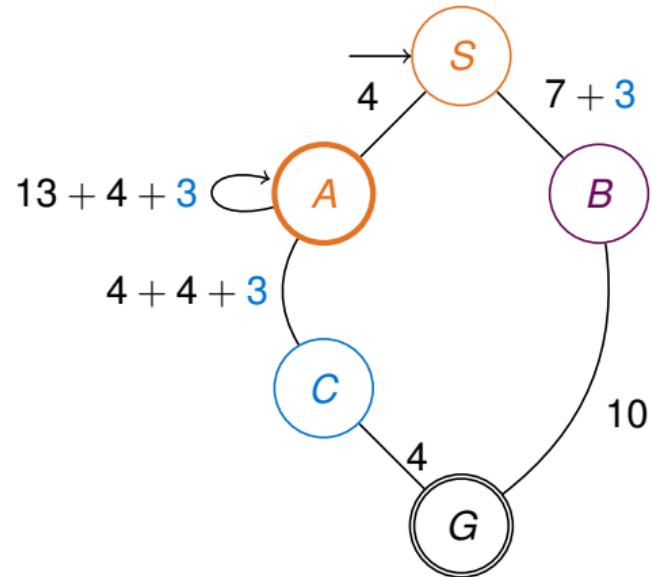


A^* search

Informed search strategies

With the following estimated distances to the goal:

- ▶ $h(A) = 3$
- ▶ $h(B) = 3$
- ▶ $h(C) = 3$
- ▶ $h(G) = 0$

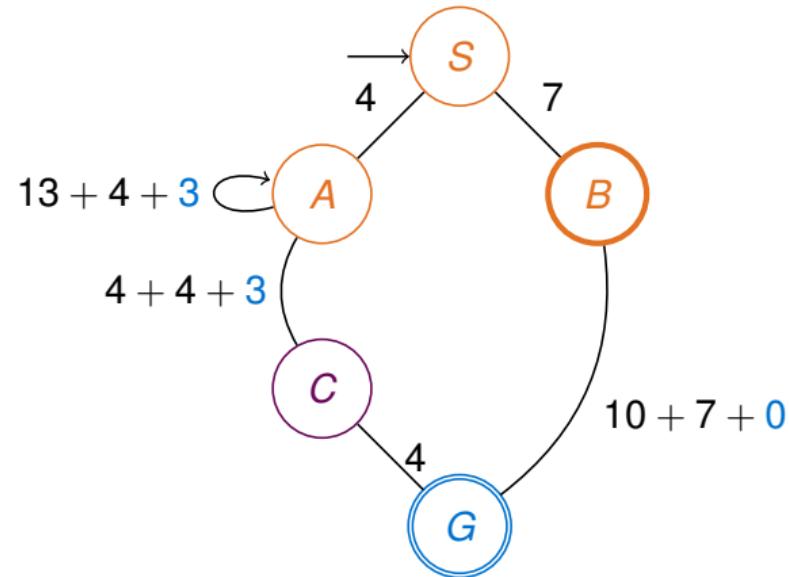


A^* search

Informed search strategies

With the following estimated distances to the goal:

- ▶ $h(A) = 3$
- ▶ $h(B) = 3$
- ▶ $h(C) = 3$
- ▶ $h(G) = 0$

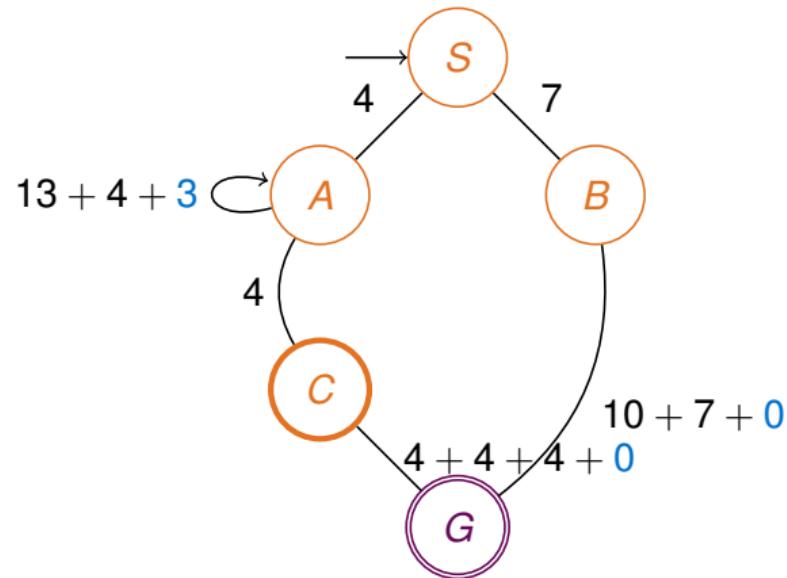


A^* search

Informed search strategies

With the following estimated distances to the goal:

- ▶ $h(A) = 3$
- ▶ $h(B) = 3$
- ▶ $h(C) = 3$
- ▶ $h(G) = 0$

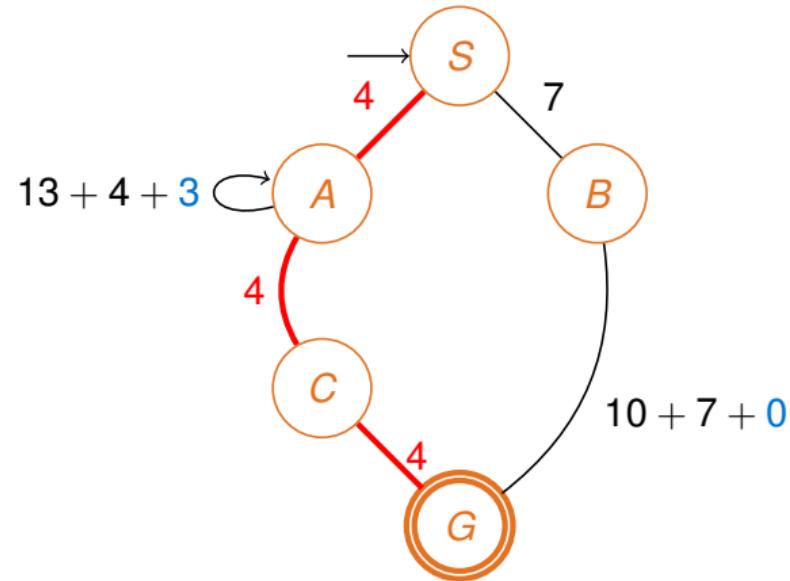


A^* search

Informed search strategies

With those estimated distances to the goal:

- We have found the goal!

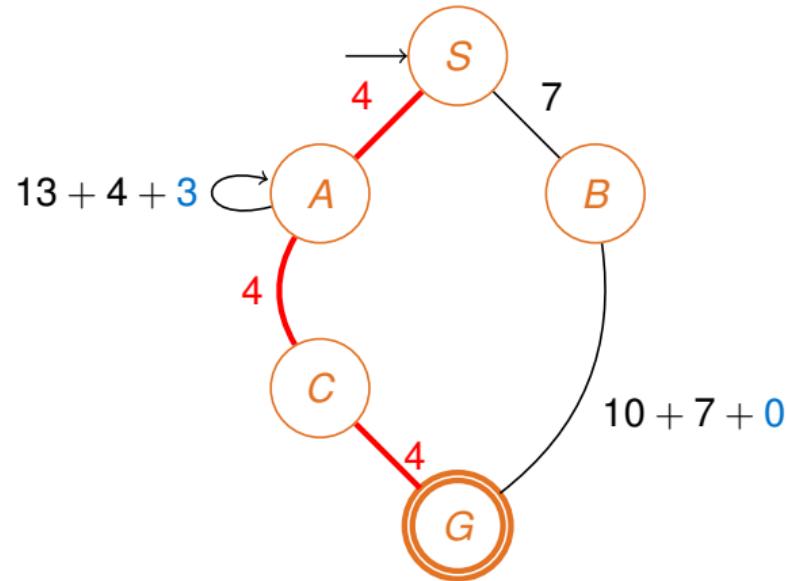


A^* search

Informed search strategies

With those estimated distances to the goal:

- ▶ We have found the goal!
- ▶ It is **Complete** for positive costs, within a finite state space and an existing solution.

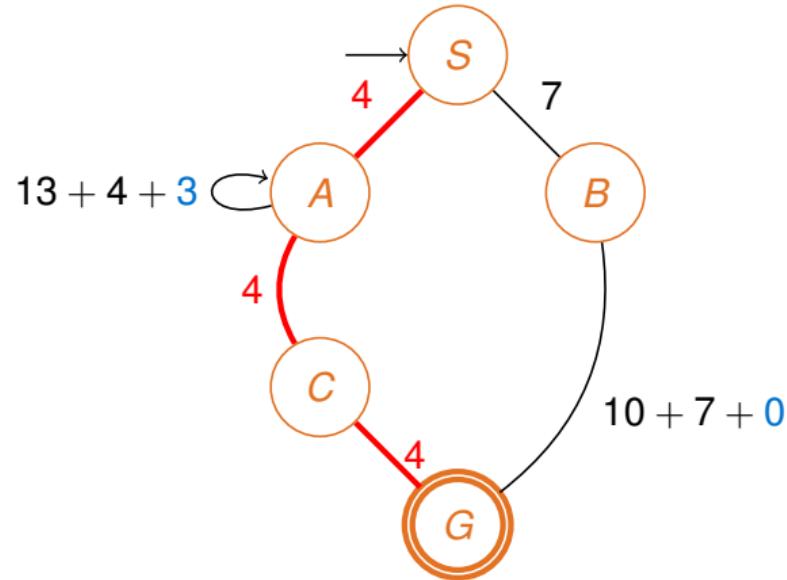


*A** search

Informed search strategies

With those estimated distances to the goal:

- ▶ We have found the goal!
- ▶ It is **Complete** for positive costs, within a finite state space and an existing solution.
- ▶ it is **Cost optimal** if **certain conditions are met**



Lecture 3

A^* Search and Search in Complex Environments

TDT4136: Introduction to Artificial Intelligence

Xavier F. C. Sánchez Díaz

Department of Computer Science
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology

September 5, 2024

Outline

1 Recap

2 More on A^*

3 Local Search Algorithms

4 Nondeterministic and partially observable environments

Recap on Uninformed Search

- ▶ Uninformed search strategies systematically navigate the search space blindly—not questioning where the goal may be in the space.
- ▶ The search space is often very large.

Recap on Uninformed Search

- ▶ Uninformed search strategies systematically navigate the search space blindly—not questioning where the goal may be in the space.
- ▶ The search space is often very large.
- ▶ We can be smarter about it using a heuristic (guess estimate)
- ▶ We covered (Greedy) Best First, where you pick the option with the best estimate
- ▶ We also covered A^* , which uses both the cost and the estimate

A^* search

Informed search strategies

What if we consider the cost *and* the heuristic?

*A** search

Informed search strategies

What if we consider the cost *and* the heuristic?

Our new **heuristic function** will consider both things:

$$f(n) = g(n) + h(n)$$

where

A^* search

Informed search strategies

What if we consider the cost *and* the heuristic?

Our new **heuristic function** will consider both things:

$$f(n) = g(n) + h(n)$$

where

- ▶ $g(n)$ is the cost we have paid so far to reach n

*A** search

Informed search strategies

What if we consider the cost *and* the heuristic?

Our new **heuristic function** will consider both things:

$$f(n) = g(n) + h(n)$$

where

- ▶ $g(n)$ is the cost we have paid so far to reach n
- ▶ $h(n)$ is the estimated cost of the node (to the goal)

*A** search

Informed search strategies

What if we consider the cost *and* the heuristic?

Our new **heuristic function** will consider both things:

$$f(n) = g(n) + h(n)$$

where

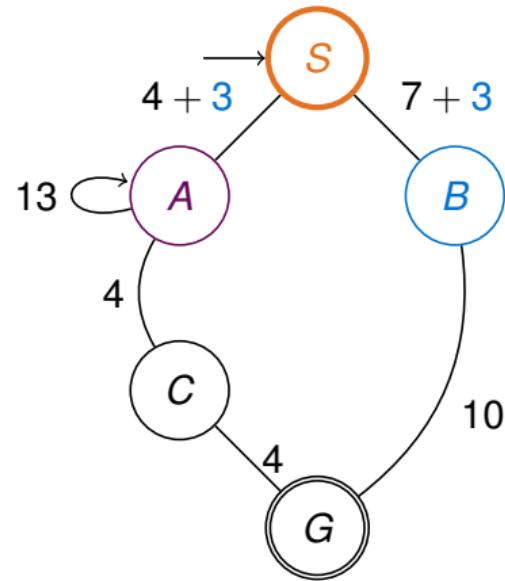
- ▶ $g(n)$ is the cost we have paid so far to reach n
- ▶ $h(n)$ is the estimated cost of the node (to the goal)
- ▶ $f(n)$ is then the estimated cost of the cheapest solution through n to the goal

A^* search

Informed search strategies

With the following estimated distances to the goal:

- ▶ $h(A) = 3$
- ▶ $h(B) = 3$
- ▶ $h(C) = 3$
- ▶ $h(G) = 0$

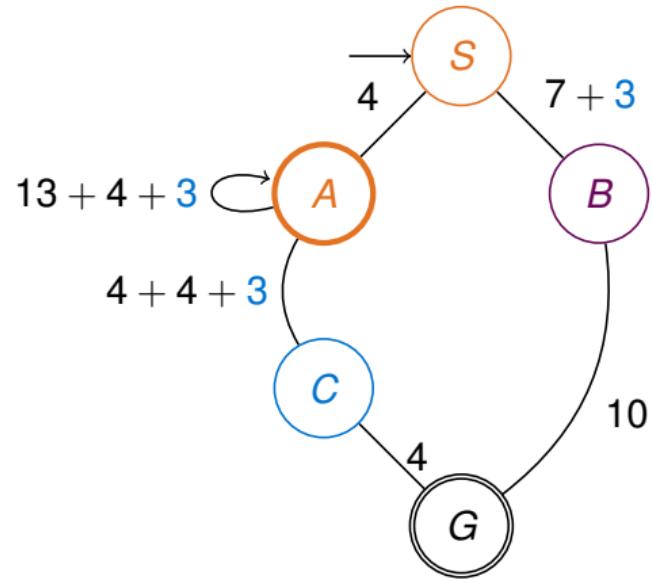


A^* search

Informed search strategies

With the following estimated distances to the goal:

- ▶ $h(A) = 3$
- ▶ $h(B) = 3$
- ▶ $h(C) = 3$
- ▶ $h(G) = 0$

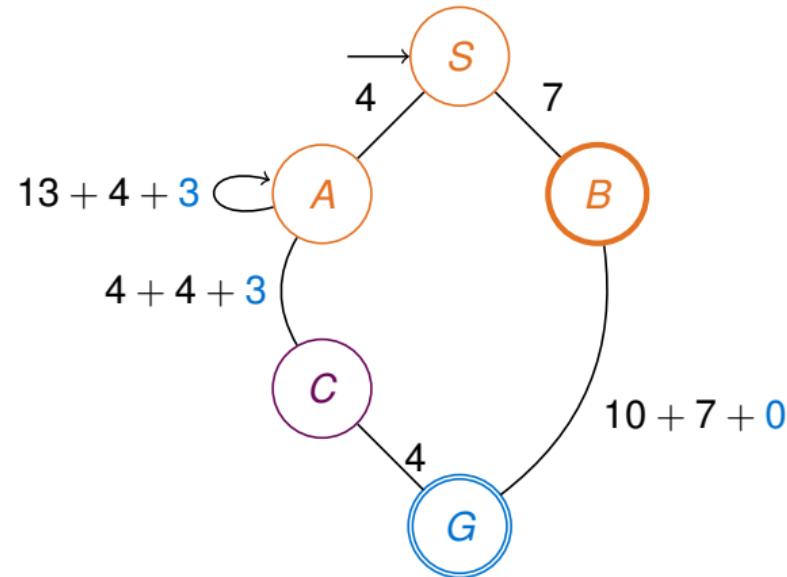


A* search

Informed search strategies

With the following estimated distances to the goal:

- ▶ $h(A) = 3$
- ▶ $h(B) = 3$
- ▶ $h(C) = 3$
- ▶ $h(G) = 0$

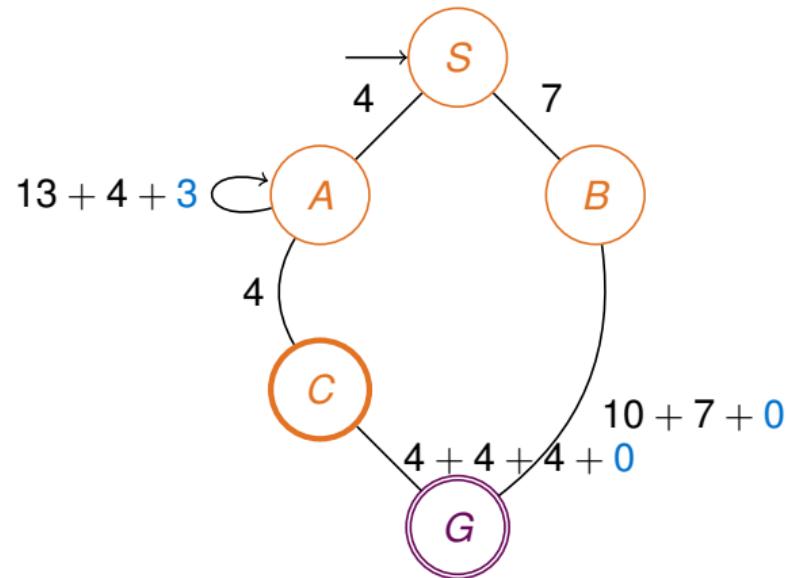


A^* search

Informed search strategies

With the following estimated distances to the goal:

- ▶ $h(A) = 3$
- ▶ $h(B) = 3$
- ▶ $h(C) = 3$
- ▶ $h(G) = 0$

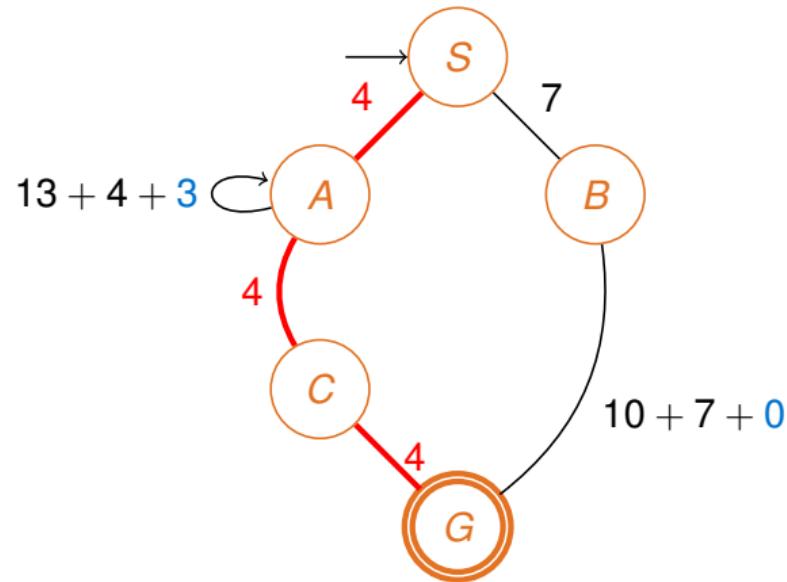


A^* search

Informed search strategies

With those estimated distances to the goal:

- We have found the goal!

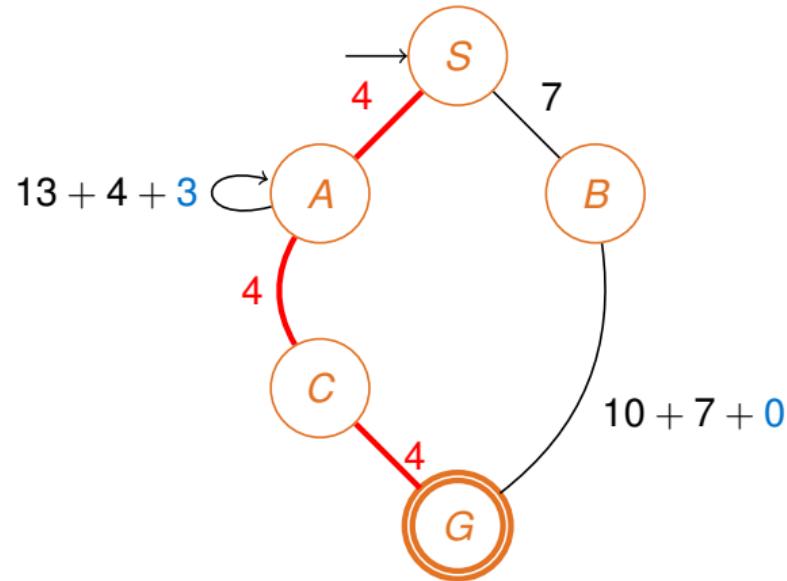


A* search

Informed search strategies

With those estimated distances to the goal:

- ▶ We have found the goal!
- ▶ It is **Complete** for positive costs, within a finite state space and an existing solution.

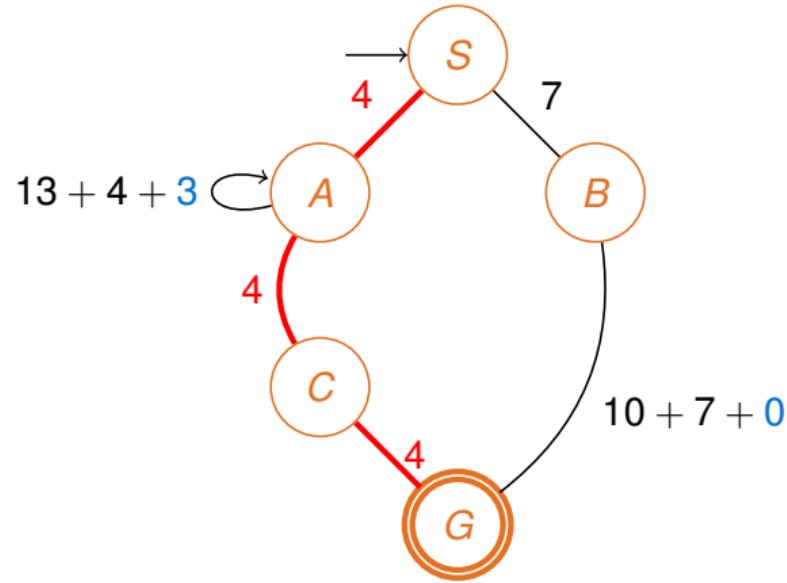


A* search

Informed search strategies

With those estimated distances to the goal:

- ▶ We have found the goal!
- ▶ It is **Complete** for positive costs, within a finite state space and an existing solution.
- ▶ It is **Cost optimal** if **certain conditions are met**



A optimality*

More on A^*

A^* is **cost optimal** if **certain conditions are met**. What are these conditions?

¹They usually are.

A optimality*

More on A^*

A^* is **cost optimal** if **certain conditions are met**. What are these conditions?

- ▶ Arc costs need to be positive¹
- ▶ The heuristic function needs to be **admissible** and non-negative.

¹They usually are.

Admissibility

More on A^*

Admissibility of h

We say a heuristic h is **admissible** if it never overestimates the cost from a node to the goal node.

Admissibility

More on A^*

Admissibility of h

We say a heuristic h is **admissible** if it never overestimates the cost from a node to the goal node.

An admissible heuristic means that for every node n :

- ▶ $h(n) \geq 0$, and
- ▶ $h(goal) = 0$

Admissibility

More on A^*

Admissibility of h

We say a heuristic h is **admissible** if it never overestimates the cost from a node to the goal node.

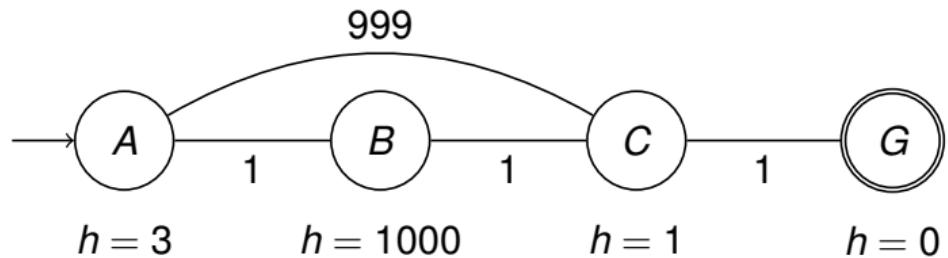
An admissible heuristic means that for every node n :

- ▶ $h(n) \geq 0$, and
- ▶ $h(goal) = 0$

An admissible heuristic is *optimistic!*

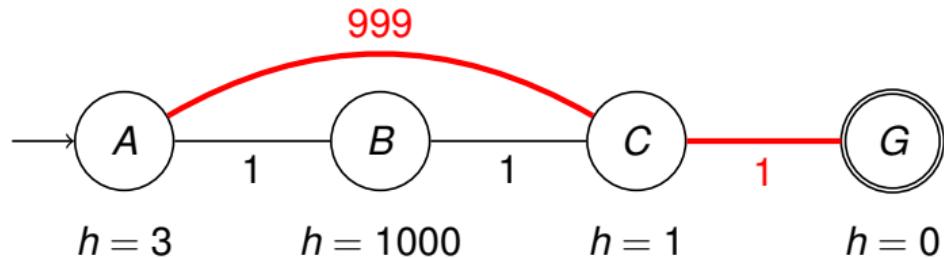
A crazy example

More on A^*



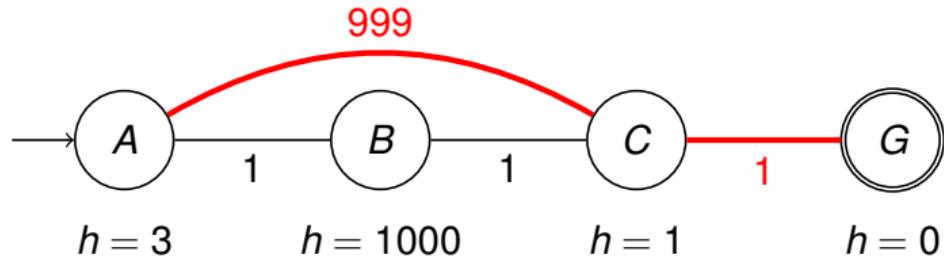
A crazy example

More on A^*



A crazy example

More on A^*



We would not choose the optimal path due to $h(B)$ being overestimated of the actual cost!

Consistency

More on A^*

Another important (and even stronger) property of a heuristic h is **consistency**.

Consistency of h

A heuristic h is **consistent** if for every node n and all of its successors n' generated by an action a , we have

$$h(n) \leq c(n, a, n') + h(n')$$

Consistency

More on A^*

Another important (and even stronger) property of a heuristic h is **consistency**.

Consistency of h

A heuristic h is **consistent** if for every node n and all of its successors n' generated by an action a , we have

$$h(n) \leq c(n, a, n') + h(n')$$

In other words, the estimate of a node should be less or equal than the the estimate of a descendant plus the cost of reaching there.

Consistency: an example

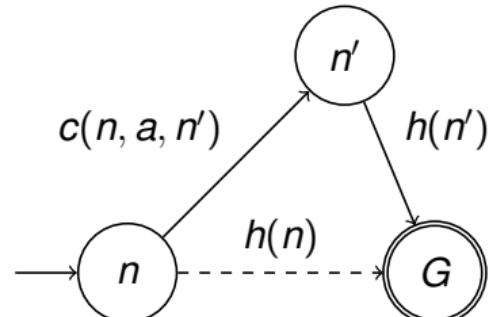
More on A^*

Consistency of h

A heuristic h is **consistent** if for every node n and all of its successors n' generated by an action a , we have

$$h(n) \leq c(n, a, n') + h(n')$$

- ▶ A triangle inequality helps picturing it!
- ▶ Moving through $h(n)$ **has** to be cheaper than going to G via the successor n'
- ▶ This **must** be true for every successor n' of n
 - ▶ Think of an euclidean grid



Consistency and admissibility

More on A^*

Why is this important?

- ▶ A heuristic that is **consistent** is always **admissible**

Consistency and admissibility

More on A^*

Why is this important?

- ▶ A heuristic that is **consistent** is always **admissible**
 - ▶ Not necessarily the other way around!

Consistency and admissibility

More on A^*

Why is this important?

- ▶ A heuristic that is **consistent** is always **admissible**
 - ▶ Not necessarily the other way around!
- ▶ Since a consistent heuristic is admissible, then a consistent heuristic is also always **optimal**

Consistency and admissibility

More on A^*

Why is this important?

- ▶ A heuristic that is **consistent** is always **admissible**
 - ▶ Not necessarily the other way around!
- ▶ Since a consistent heuristic is admissible, then a consistent heuristic is also always **optimal**
- ▶ A consistent heuristic $h(n)$ ensures that the cost function $f(n) = g(n) + h(n)$ is **monotonic nondecreasing**

Consistency and admissibility

More on A^*

Why is this important?

- ▶ A heuristic that is **consistent** is always **admissible**
 - ▶ Not necessarily the other way around!
- ▶ Since a consistent heuristic is admissible, then a consistent heuristic is also always **optimal**
- ▶ A consistent heuristic $h(n)$ ensures that the cost function $f(n) = g(n) + h(n)$ is **monotonic nondecreasing**
 - ▶ That means that $f(n)$ is non-decreasing along any path

Optimality and efficiency

More on A^*

- ▶ A^* is **optimally efficient** with a consistent heuristic
- ▶ This means that any other search algorithm with the same heuristic values must expand all nodes that A^* expanded

Optimality and efficiency

More on A^*

- ▶ A^* is **optimally efficient** with a consistent heuristic
- ▶ This means that any other search algorithm with the same heuristic values must expand all nodes that A^* expanded

However, the main issue of A^* lies on its memory use. Some ways to reduce it:

- ▶ Reference count – remove a state from *reached* when there are no more ways to reach it

Optimality and efficiency

More on A^*

- ▶ A^* is **optimally efficient** with a consistent heuristic
- ▶ This means that any other search algorithm with the same heuristic values must expand all nodes that A^* expanded

However, the main issue of A^* lies on its memory use. Some ways to reduce it:

- ▶ Reference count – remove a state from *reached* when there are no more ways to reach it
- ▶ Beam search – limit size of *frontier* to k -best candidates

Optimality and efficiency

More on A^*

- ▶ A^* is **optimally efficient** with a consistent heuristic
- ▶ This means that any other search algorithm with the same heuristic values must expand all nodes that A^* expanded

However, the main issue of A^* lies on its memory use. Some ways to reduce it:

- ▶ Reference count – remove a state from *reached* when there are no more ways to reach it
- ▶ Beam search – limit size of *frontier* to k -best candidates
- ▶ Iterative deepening A^* – gradually increase the f -cost *cutoff*.

Optimality and efficiency

More on A^*

- ▶ A^* is **optimally efficient** with a consistent heuristic
- ▶ This means that any other search algorithm with the same heuristic values must expand all nodes that A^* expanded

However, the main issue of A^* lies on its memory use. Some ways to reduce it:

- ▶ Reference count – remove a state from *reached* when there are no more ways to reach it
- ▶ Beam search – limit size of *frontier* to k -best candidates
- ▶ Iterative deepening A^* – gradually increase the f -cost *cutoff*.
- ▶ Memory-bounded A^* – expand until memory is full, and then drop the worst candidate from *frontier*

A generalised heuristic search

More on A^*

Generalised heuristic search

$$f(n) = g(n) + w \cdot h(n)$$

where w is a *weight* defining how important the heuristic $h(n)$ is.

In most other applications, we usually have w_1 and w_2 , one for $g(n)$ and one for $h(n)$. The book uses only w for $h(n)$.

A generalised heuristic search

More on A^*

$$f(n) = g(n) + w \cdot h(n)$$

A generalised heuristic search

More on A^*

$$f(n) = g(n) + w \cdot h(n)$$

- ▶ With $w = 0$ you only care about the cost of the path

A generalised heuristic search

More on A^*

$$f(n) = g(n) + w \cdot h(n)$$

- ▶ With $w = 0$ you only care about the cost of the path
 - ▶ Choose the cheapest!

A generalised heuristic search

More on A^*

$$f(n) = g(n) + w \cdot h(n)$$

- ▶ With $w = 0$ you only care about the cost of the path
 - ▶ Choose the cheapest!
 - ▶ This is called **uniform-cost search** and it's an uninformed search.

A generalised heuristic search

More on A^*

$$f(n) = g(n) + w \cdot h(n)$$

- ▶ With $w = 0$ you only care about the cost of the path
 - ▶ Choose the cheapest!
 - ▶ This is called **uniform-cost search** and it's an uninformed search.
 - ▶ It is also known as [Dijkstra's algorithm](#).

A generalised heuristic search

More on A^*

$$f(n) = g(n) + w \cdot h(n)$$

- ▶ With $w = 0$ you only care about the cost of the path
 - ▶ Choose the cheapest!
 - ▶ This is called **uniform-cost search** and it's an uninformed search.
 - ▶ It is also known as Dijkstra's algorithm.
- ▶ With $w = \infty$ you only care about the estimate

A generalised heuristic search

More on A^*

$$f(n) = g(n) + w \cdot h(n)$$

- ▶ With $w = 0$ you only care about the cost of the path
 - ▶ Choose the cheapest!
 - ▶ This is called **uniform-cost search** and it's an uninformed search.
 - ▶ It is also known as Dijkstra's algorithm.
- ▶ With $w = \infty$ you only care about the estimate
 - ▶ Choose the one that *seems* the cheapest

A generalised heuristic search

More on A^*

$$f(n) = g(n) + w \cdot h(n)$$

- ▶ With $w = 0$ you only care about the cost of the path
 - ▶ Choose the cheapest!
 - ▶ This is called **uniform-cost search** and it's an uninformed search.
 - ▶ It is also known as Dijkstra's algorithm.
- ▶ With $w = \infty$ you only care about the estimate
 - ▶ Choose the one that *seems* the cheapest
 - ▶ This is **Greedy Best-First search**

A generalised heuristic search

More on A^*

$$f(n) = g(n) + w \cdot h(n)$$

- ▶ With $w = 0$ you only care about the cost of the path
 - ▶ Choose the cheapest!
 - ▶ This is called **uniform-cost search** and it's an uninformed search.
 - ▶ It is also known as Dijkstra's algorithm.
- ▶ With $w = \infty$ you only care about the estimate
 - ▶ Choose the one that *seems* the cheapest
 - ▶ This is **Greedy Best-First search**
- ▶ with $w = 1$ you care equally about the path cost and the estimates

A generalised heuristic search

More on A^*

$$f(n) = g(n) + w \cdot h(n)$$

- ▶ With $w = 0$ you only care about the cost of the path
 - ▶ Choose the cheapest!
 - ▶ This is called **uniform-cost search** and it's an uninformed search.
 - ▶ It is also known as Dijkstra's algorithm.
- ▶ With $w = \infty$ you only care about the estimate
 - ▶ Choose the one that *seems* the cheapest
 - ▶ This is **Greedy Best-First search**
- ▶ with $w = 1$ you care equally about the path cost and the estimates
 - ▶ This is A^*

A generalised heuristic search

More on A^*

$$f(n) = g(n) + w \cdot h(n)$$

- ▶ With $w = 0$ you only care about the cost of the path
 - ▶ Choose the cheapest!
 - ▶ This is called **uniform-cost search** and it's an uninformed search.
 - ▶ It is also known as Dijkstra's algorithm.
- ▶ With $w = \infty$ you only care about the estimate
 - ▶ Choose the one that *seems* the cheapest
 - ▶ This is **Greedy Best-First search**
- ▶ with $w = 1$ you care equally about the path cost and the estimates
 - ▶ This is A^*

A generalised heuristic search

More on A^*

$$f(n) = g(n) + w \cdot h(n)$$

- ▶ With $w = 0$ you only care about the cost of the path
 - ▶ Choose the cheapest!
 - ▶ This is called **uniform-cost search** and it's an uninformed search.
 - ▶ It is also known as Dijkstra's algorithm.
- ▶ With $w = \infty$ you only care about the estimate
 - ▶ Choose the one that *seems* the cheapest
 - ▶ This is **Greedy Best-First search**
- ▶ with $w = 1$ you care equally about the path cost and the estimates
 - ▶ This is A^*

Of course you can set w to something else, depending for example if there is *uncertainty* on your heuristic (but this then becomes a whole other course :^))

Building heuristics

More on A^*



How far are we from solving this sliding puzzle?

- ▶ $h_1(n)$ will be the number of misplaced tiles
- ▶ $h_2(n)$ will be the total Manhattan distance^a

^anumber of squares away from the desired location

Building heuristics

More on A^*



How far are we from solving this sliding puzzle?

- ▶ $h_1(n)$ will be the number of misplaced tiles
- ▶ $h_2(n)$ will be the total Manhattan distance^a

^anumber of squares away from the desired location

Remember that each configuration is a state!

Other ideas for building heuristics

More on A^*

- ▶ Consider **relaxations** of the problem
- ▶ Consider creating the heuristic by looking **backwards from the goal**.
- ▶ Consider dividing into subproblems!
 - ▶ For example, instead of solving the whole sliding puzzle at once, consider getting in place four tiles only
 - ▶ Then store all these solutions in a DB. Create an admissible heuristic for this subproblem
 - ▶ Combine the subproblems to choose **the best heuristic**

The process of choosing the appropriate representation, data structures and heuristics for a problem is known as **modelling** and is crucial for AI developers and researchers!

Dominance: comparing heuristics

More on A^*

Which of the heuristics is better?

Dominance: comparing heuristics

More on A^*

Which of the heuristics is better?

Admissible heuristics can be compared by looking at their values.

Heuristic Domination

An admissible heuristic h_2 it is said to **dominate** another admissible heuristic h_1 if for all nodes n if $h_2(n) \geq h_1(n)$.

This will reflect in A^* expanding fewer nodes on h_2 , and thus find an optimal solution, faster.

Dominance: comparing heuristics

More on A^*

Which of the heuristics is better?

Admissible heuristics can be compared by looking at their values.

Heuristic Domination

An admissible heuristic h_2 it is said to **dominate** another admissible heuristic h_1 if for all nodes n if $h_2(n) \geq h_1(n)$.

This will reflect in A^* expanding fewer nodes on h_2 , and thus find an optimal solution, faster.

A generalisation of this would then be

$$h_{best}(n) = \max(h_a(n), h_b(n), \dots)$$

Section 3

Search in Complex Environments

Searching in complex environments

- ▶ Both informed and uninformed searching strategies are designed to explore search spaces systematically
- ▶ They keep one or more paths in memory, and record which alternatives have been explored at each point along the path
- ▶ The path to that goal constitutes a solution
- ▶ But in most problems in the real world, the path to a solution **might be irrelevant**

Searching in complex environments

- ▶ Both informed and uninformed searching strategies are designed to explore search spaces systematically
- ▶ They keep one or more paths in memory, and record which alternatives have been explored at each point along the path
- ▶ The path to that goal constitutes a solution
- ▶ But in most problems in the real world, the path to a solution **might be irrelevant**

If we only care about finding a solution, then there are better ways to search the space!

Local Search

- ▶ It uses a single **current node** and move to **neighbouring** nodes

²as in most real world applications

Local Search

- ▶ It uses a single **current node** and move to **neighbouring** nodes
- ▶ It eases up on the completeness and optimality in the interest of improving time and space complexity²

²as in most real world applications

Local Search

- ▶ It uses a single **current node** and move to **neighbouring** nodes
- ▶ It eases up on the completeness and optimality in the interest of improving time and space complexity²
- ▶ Local Search algorithms use “little” memory (usually a constant amount)

²as in most real world applications

Local Search

- ▶ It uses a single **current node** and move to **neighbouring** nodes
- ▶ It eases up on the completeness and optimality in the interest of improving time and space complexity²
- ▶ Local Search algorithms use “little” memory (usually a constant amount)
- ▶ They can often find reasonable solutions in very large (or infinite) state spaces

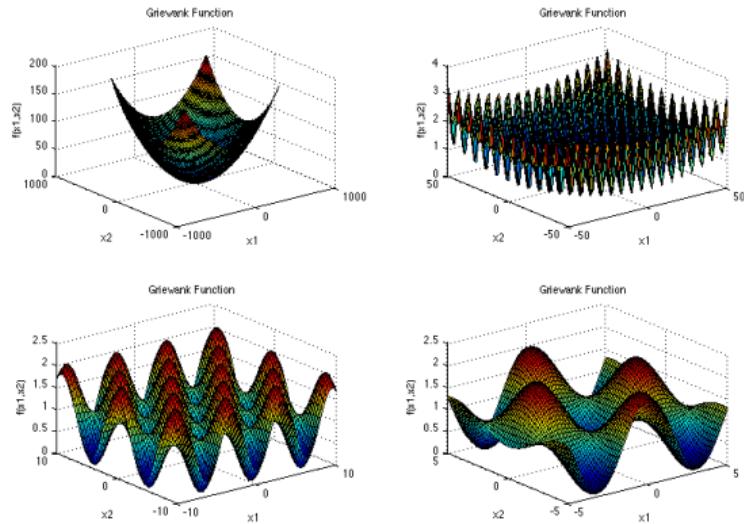
²as in most real world applications

The search landscape

Search in complex environments

Usually, the **state space** is referred to as the **search space**. We can **visualise** this space by looking at the heuristic function!

$$f(x) = \sum_{i=1}^d \frac{x_i^2}{4000} - \prod_{i=1}^d \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

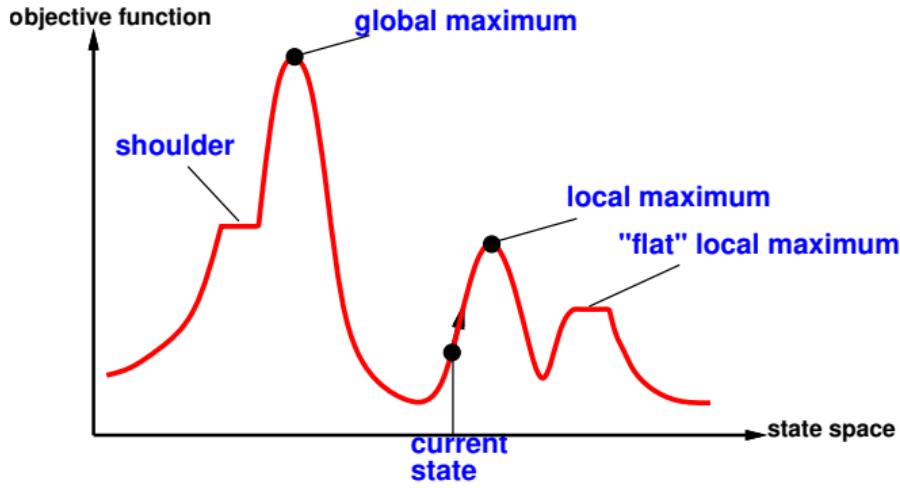


The Griewank function. Image from Surjanovic & Bingham

<https://www.sfu.ca/~ssurjano/griewank.html>

The search landscape

Search in complex environments

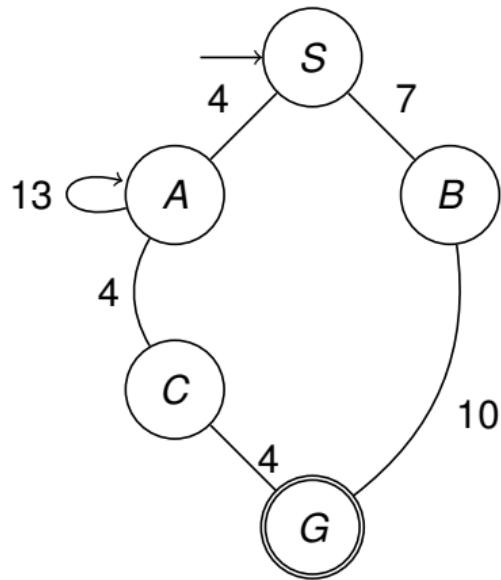


- ▶ Each point in the landscape represents a state in the search space and has “an elevation” (its $h(n)$)
- ▶ If the elevation corresponds to an objective function, then the aim is to find the highest peak (or **maximum**)
- ▶ If the elevation corresponds to a cost function, then we look for the lowest valley (or **minimum**)

The search landscape

Search in complex environments

Recall our search problems.



- ▶ A is a neighbour of S , C and itself because those are the states than can be reached from A .
- ▶ The **neighbourhood** of A is then $\{A, C, S\}$.
- ▶ This concept of **neighbourhood** is very important for **local search**, as we decide *where to move next* by looking around us!

Section 4

Local Search Algorithms

As the last time with algorithms, please check
the full details on the book!

Hill climbing and gradient descent

Local search algorithms

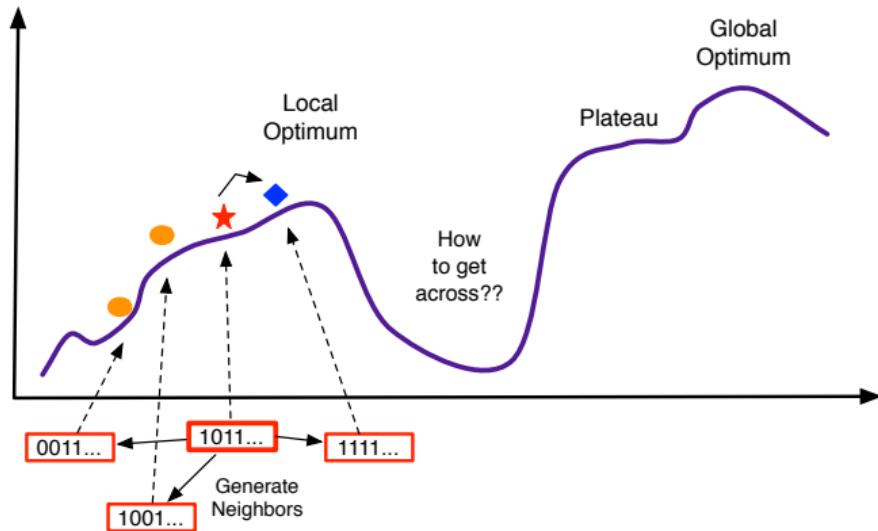
Idea: Go to the best spot you see now.

Hill climbing and gradient descent

Local search algorithms

Idea: Go to the best spot you see now.

- ▶ Assume you are doing **maximisation**
- ▶ You then want to **climb** the tallest peak
- ▶ This is called **hill-climbing!**



If you are **minimising** instead, then the procedure is called **gradient descent** as we want to move towards the direction where the difference in “height” is largest.

How to get across?

Both **hill-climbing** and **gradient descent** get stuck in **local optima**. How do we get out of this mess?

How to get across?

Both **hill-climbing** and **gradient descent** get stuck in **local optima**. How do we get out of this mess?

- ▶ Idea: take some *not so good* decisions every now and then!

This line of research is usually referred to as **metaheuristics**.

How to get across?

Both **hill-climbing** and **gradient descent** get stuck in **local optima**. How do we get out of this mess?

- ▶ Idea: take some *not so good* decisions every now and then!
 - ▶ This is what we call **stochastic local search**.

This line of research is usually referred to as **metaheuristics**.

How to get across?

Both **hill-climbing** and **gradient descent** get stuck in **local optima**. How do we get out of this mess?

- ▶ Idea: take some *not so good* decisions every now and then!
 - ▶ This is what we call **stochastic local search**.
- ▶ Idea 2: Make it so that you gradually reduce the frequency of taking such “bad” decisions

This line of research is usually referred to as **metaheuristics**.

How to get across?

Both **hill-climbing** and **gradient descent** get stuck in **local optima**. How do we get out of this mess?

- ▶ Idea: take some *not so good* decisions every now and then!
 - ▶ This is what we call **stochastic local search**.
- ▶ Idea 2: Make it so that you gradually reduce the frequency of taking such “bad” decisions
 - ▶ This is the key to the **simulated annealing** algorithm

This line of research is usually referred to as **metaheuristics**.

How to get across?

Both **hill-climbing** and **gradient descent** get stuck in **local optima**. How do we get out of this mess?

- ▶ Idea: take some *not so good* decisions every now and then!
 - ▶ This is what we call **stochastic local search**.
- ▶ Idea 2: Make it so that you gradually reduce the frequency of taking such “bad” decisions
 - ▶ This is the key to the **simulated annealing** algorithm
- ▶ Idea 3: Search multiple paths in batches

This line of research is usually referred to as **metaheuristics**.

How to get across?

Both **hill-climbing** and **gradient descent** get stuck in **local optima**. How do we get out of this mess?

- ▶ Idea: take some *not so good* decisions every now and then!
 - ▶ This is what we call **stochastic local search**.
- ▶ Idea 2: Make it so that you gradually reduce the frequency of taking such “bad” decisions
 - ▶ This is the key to the **simulated annealing** algorithm
- ▶ Idea 3: Search multiple paths in batches
 - ▶ This is the key idea behind **population-based optimisation**

This line of research is usually referred to as **metaheuristics**.

How to get across?

Both **hill-climbing** and **gradient descent** get stuck in **local optima**. How do we get out of this mess?

- ▶ Idea: take some *not so good* decisions every now and then!
 - ▶ This is what we call **stochastic local search**.
- ▶ Idea 2: Make it so that you gradually reduce the frequency of taking such “bad” decisions
 - ▶ This is the key to the **simulated annealing** algorithm
- ▶ Idea 3: Search multiple paths in batches
 - ▶ This is the key idea behind **population-based optimisation**
- ▶ Idea 4: Increase the **neighbourhood size**

This line of research is usually referred to as **metaheuristics**.

How to get across?

Both **hill-climbing** and **gradient descent** get stuck in **local optima**. How do we get out of this mess?

- ▶ Idea: take some *not so good* decisions every now and then!
 - ▶ This is what we call **stochastic local search**.
- ▶ Idea 2: Make it so that you gradually reduce the frequency of taking such “bad” decisions
 - ▶ This is the key to the **simulated annealing** algorithm
- ▶ Idea 3: Search multiple paths in batches
 - ▶ This is the key idea behind **population-based optimisation**
- ▶ Idea 4: Increase the **neighbourhood size**
 - ▶ For example, consider 2-moves-away adjacency instead

This line of research is usually referred to as **metaheuristics**.

How to get across?

Both **hill-climbing** and **gradient descent** get stuck in **local optima**. How do we get out of this mess?

- ▶ Idea: take some *not so good* decisions every now and then!
 - ▶ This is what we call **stochastic local search**.
- ▶ Idea 2: Make it so that you gradually reduce the frequency of taking such “bad” decisions
 - ▶ This is the key to the **simulated annealing** algorithm
- ▶ Idea 3: Search multiple paths in batches
 - ▶ This is the key idea behind **population-based optimisation**
- ▶ Idea 4: Increase the **neighbourhood size**
 - ▶ For example, consider 2-moves-away adjacency instead
- ▶ Idea 5: Jump!

This line of research is usually referred to as **metaheuristics**.

How to get across?

Both **hill-climbing** and **gradient descent** get stuck in **local optima**. How do we get out of this mess?

- ▶ Idea: take some *not so good* decisions every now and then!
 - ▶ This is what we call **stochastic local search**.
- ▶ Idea 2: Make it so that you gradually reduce the frequency of taking such “bad” decisions
 - ▶ This is the key to the **simulated annealing** algorithm
- ▶ Idea 3: Search multiple paths in batches
 - ▶ This is the key idea behind **population-based optimisation**
- ▶ Idea 4: Increase the **neighbourhood size**
 - ▶ For example, consider 2-moves-away adjacency instead
- ▶ Idea 5: Jump!
 - ▶ Either via *long* jumps when you are not doing very good

This line of research is usually referred to as **metaheuristics**.

How to get across?

Both **hill-climbing** and **gradient descent** get stuck in **local optima**. How do we get out of this mess?

- ▶ Idea: take some *not so good* decisions every now and then!
 - ▶ This is what we call **stochastic local search**.
- ▶ Idea 2: Make it so that you gradually reduce the frequency of taking such “bad” decisions
 - ▶ This is the key to the **simulated annealing** algorithm
- ▶ Idea 3: Search multiple paths in batches
 - ▶ This is the key idea behind **population-based optimisation**
- ▶ Idea 4: Increase the **neighbourhood size**
 - ▶ For example, consider 2-moves-away adjacency instead
- ▶ Idea 5: Jump!
 - ▶ Either via *long* jumps when you are not doing very good
 - ▶ Or doing *short* hops when you are in a promising state (you do not want to miss it)

This line of research is usually referred to as **metaheuristics**.

Genetic Algorithms

Local search algorithms

A well-known metaheuristic in the family of **population-based** optimisers is the **genetic algorithm**.

Genetic Algorithms

Local search algorithms

A well-known metaheuristic in the family of **population-based** optimisers is the **genetic algorithm**.

1. Start with a population of k randomly generated states

Genetic Algorithms

Local search algorithms

A well-known metaheuristic in the family of **population-based** optimisers is the **genetic algorithm**.

1. Start with a population of k randomly generated states
2. Randomly choose two parent states weighted by their fitness (objective function)

Genetic Algorithms

Local search algorithms

A well-known metaheuristic in the family of **population-based** optimisers is the **genetic algorithm**.

1. Start with a population of k randomly generated states
2. Randomly choose two parent states weighted by their fitness (objective function)
3. Generate child states by combining parent states randomly

Genetic Algorithms

Local search algorithms

A well-known metaheuristic in the family of **population-based** optimisers is the **genetic algorithm**.

1. Start with a population of k randomly generated states
2. Randomly choose two parent states weighted by their fitness (objective function)
3. Generate child states by combining parent states randomly
4. Add child states to the population

Genetic Algorithms

Local search algorithms

A well-known metaheuristic in the family of **population-based** optimisers is the **genetic algorithm**.

1. Start with a population of k randomly generated states
2. Randomly choose two parent states weighted by their fitness (objective function)
3. Generate child states by combining parent states randomly
4. Add child states to the population
5. Replace the old population by the new

Genetic Algorithms

Local search algorithms

A well-known metaheuristic in the family of **population-based** optimisers is the **genetic algorithm**.

1. Start with a population of k randomly generated states
2. Randomly choose two parent states weighted by their fitness (objective function)
3. Generate child states by combining parent states randomly
4. Add child states to the population
5. Replace the old population by the new

Genetic Algorithms

Local search algorithms

A well-known metaheuristic in the family of **population-based** optimisers is the **genetic algorithm**.

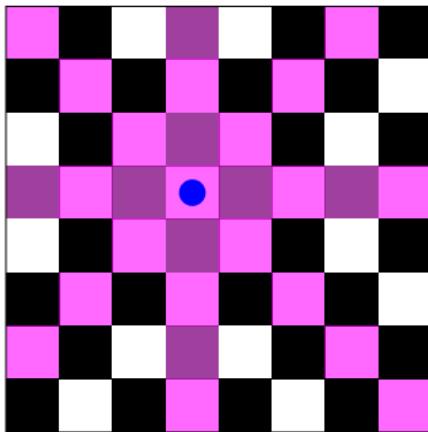
1. Start with a population of k randomly generated states
2. Randomly choose two parent states weighted by their fitness (objective function)
3. Generate child states by combining parent states randomly
4. Add child states to the population
5. Replace the old population by the new

This process will be repeated until a solution has been found, or until enough *generations* have been replaced.

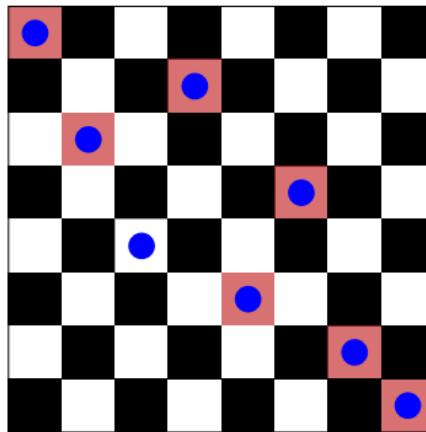
We have a whole course on evolutionary computation methods during the spring semester:
IT3708 Bio-Inspired AI!

The 8-queens problem

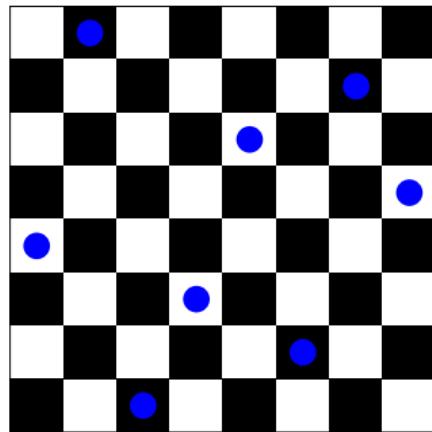
Place 8 queens in a chess board such that no queen checks each other.



(a) Queen constraints



(b) Generated conflicts



(c) Possible solution

Figure: The 8-queens problem. 1a shows the constraints (in pink) imposed by the placement of a single queen piece (in blue). 1b highlights the conflicts arising from a possible configuration of the board. 1c illustrates one possible solution with no conflicts.

See a worked example in https://ntnu-ai-lab.github.io/EvoLP.jl/stable/tuto/8_queens.html

Section 5

Nondeterministic and partially observable environments

Searching with Nondeterminism

- So far, we have assumed that actions are deterministic

Searching with Nondeterminism

- ▶ So far, we have assumed that actions are deterministic
 - ▶ That our intended action will always yield the result we expect

Searching with Nondeterminism

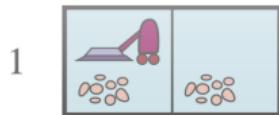
- ▶ So far, we have assumed that actions are deterministic
 - ▶ That our intended action will always yield the result we expect
- ▶ In the real-world, things do not always go as expected

Searching with Nondeterminism

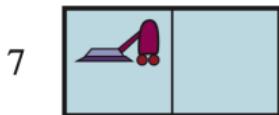
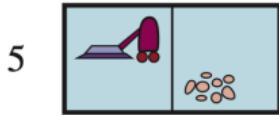
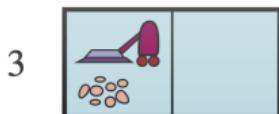
- ▶ So far, we have assumed that actions are deterministic
 - ▶ That our intended action will always yield the result we expect
- ▶ In the real-world, things do not always go as expected
- ▶ To account for different possible outcomes, we need to come up with a contingency plan instead of a single path of actions

Example: The Erratic Vacuum World

Searching with Nondeterminism

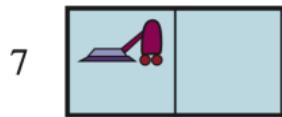
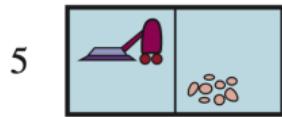
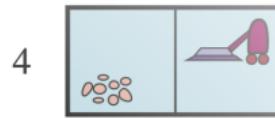
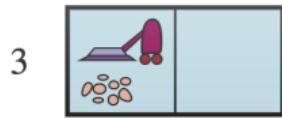
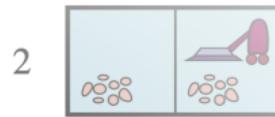
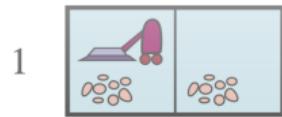


- ▶ Nondeterministic *suck* action:
 $\text{suck}(s_1) = \{s_5, s_7\}$



Example: The Erratic Vacuum World

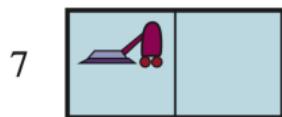
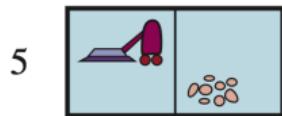
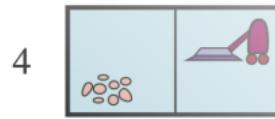
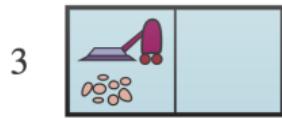
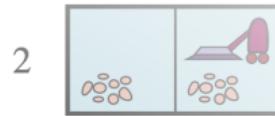
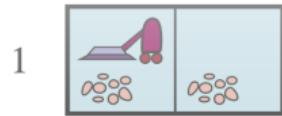
Searching with Nondeterminism



- ▶ Nondeterministic *suck* action:
 $suck(s_1) = \{s_5, s_7\}$
 - ▶ which means both states s_5 and s_7 are possible outcomes of executing a *suck* action on state s_1

Example: The Erratic Vacuum World

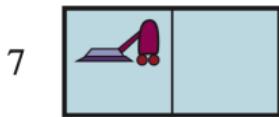
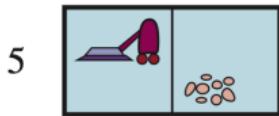
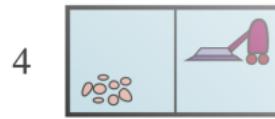
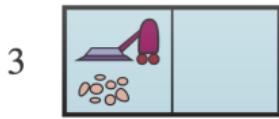
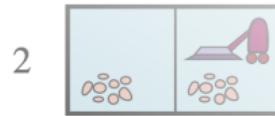
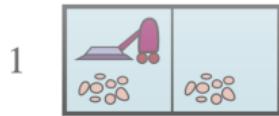
Searching with Nondeterminism



- ▶ Nondeterministic *suck* action:
 $suck(s_1) = \{s_5, s_7\}$
 - ▶ which means both states s_5 and s_7 are possible outcomes of executing a *suck* action on state s_1
 - ▶ $suck(s_7) = \{s_3, s_7\}$

Example: The Erratic Vacuum World

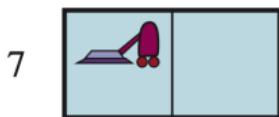
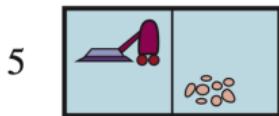
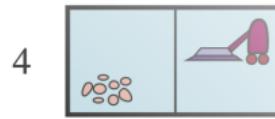
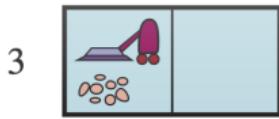
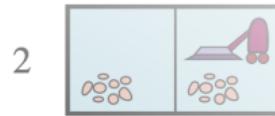
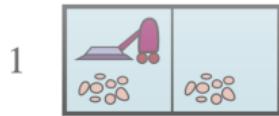
Searching with Nondeterminism



- ▶ Nondeterministic *suck* action:
 $suck(s_1) = \{s_5, s_7\}$
 - ▶ which means both states s_5 and s_7 are possible outcomes of executing a *suck* action on state s_1
- ▶ $suck(s_7) = \{s_3, s_7\}$
 - ▶ Which means both s_3 and s_7 are possible outcomes of *suck* on s_7

Example: The Erratic Vacuum World

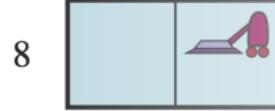
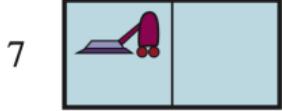
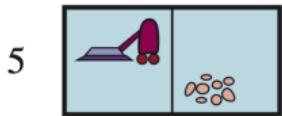
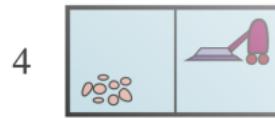
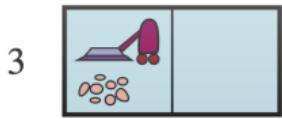
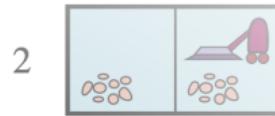
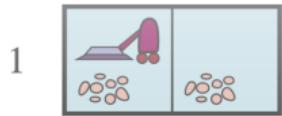
Searching with Nondeterminism



- ▶ Nondeterministic *suck* action:
 $suck(s_1) = \{s_5, s_7\}$
 - ▶ which means both states s_5 and s_7 are possible outcomes of executing a *suck* action on state s_1
- ▶ $suck(s_7) = \{s_3, s_7\}$
 - ▶ Which means both s_3 and s_7 are possible outcomes of *suck* on s_7

Example: The Erratic Vacuum World

Searching with Nondeterminism



- ▶ Nondeterministic *suck* action:
 $suck(s_1) = \{s_5, s_7\}$
 - ▶ which means both states s_5 and s_7 are possible outcomes of executing a *suck* action on state s_1
- ▶ $suck(s_7) = \{s_3, s_7\}$
 - ▶ Which means both s_3 and s_7 are possible outcomes of *suck* on s_7

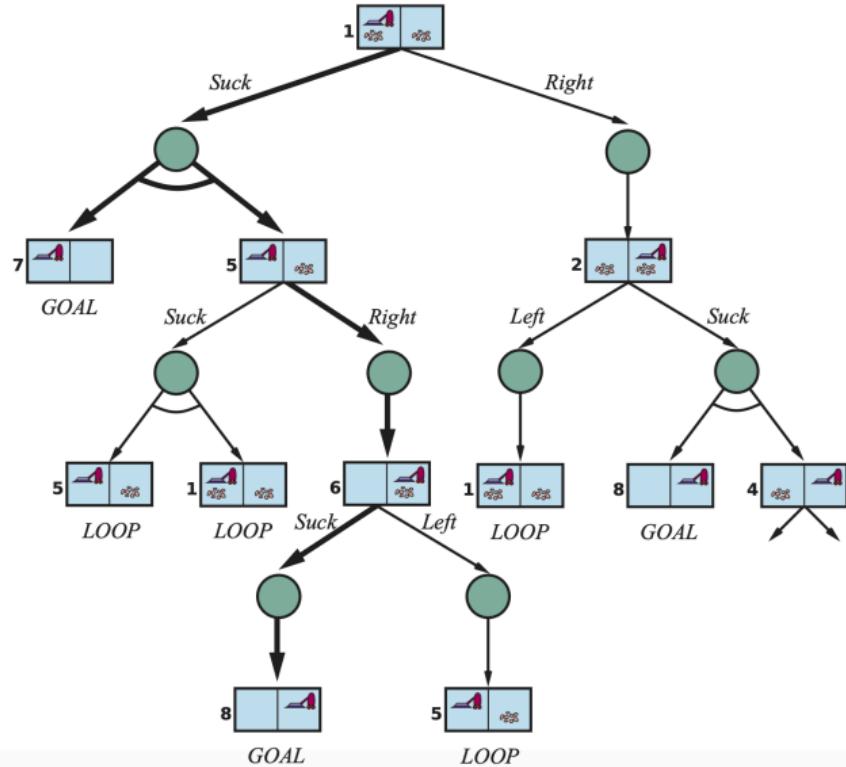
Nondeterminism can happen with other actions like *moveRight!* See the *slippery vacuum world* in the book!

AND-OR search trees

Searching with Nondeterminism

One way to handle these, is to consider *compound nodes*, made up of the possible states after a given action

- ▶ OR nodes represent actions
- ▶ AND nodes represent outcomes
- ▶ Since it is a **tree**, we can **search** in it
 - ▶ This is called **AND-OR search**
 - ▶ It is recursive, with a base case of either failure or an empty plan



Searching in Partially Observable Environments

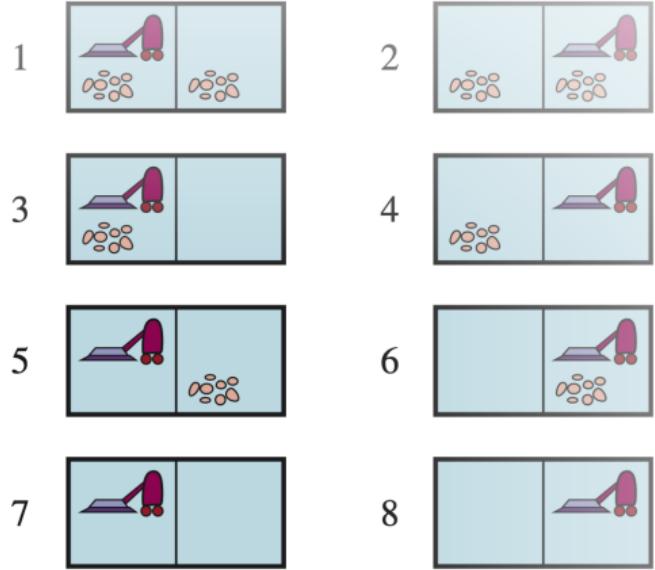
- ▶ So far, we have assumed that the agent knows exactly the state of its environment
- ▶ In reality, an agent receives partial (and possibly noisy) observations
- ▶ Therefore, the state can only be *estimated* through a “belief”

Sensorless deterministic vacuum world

Searching in partially observable environments



$\text{Result}(\{1, 2, 3, 4, 5, 6, 7, 8\}, \text{moveRight}) = \{2, 4, 6, 8\}$



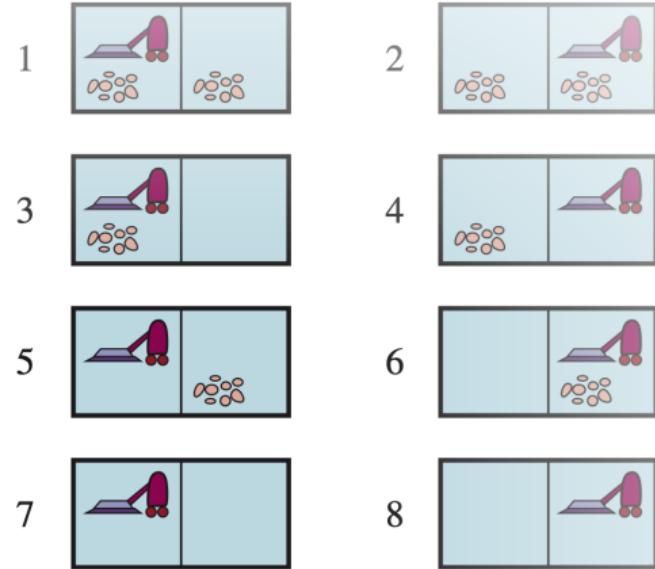
Sensorless deterministic vacuum world

Searching in partially observable environments



$\text{Result}(\{1, 2, 3, 4, 5, 6, 7, 8\}, \text{moveRight}) = \{2, 4, 6, 8\}$

- Which means that executing moveRight on any state $s \in S$ will yield a result in $\{2, 4, 6, 8\}$



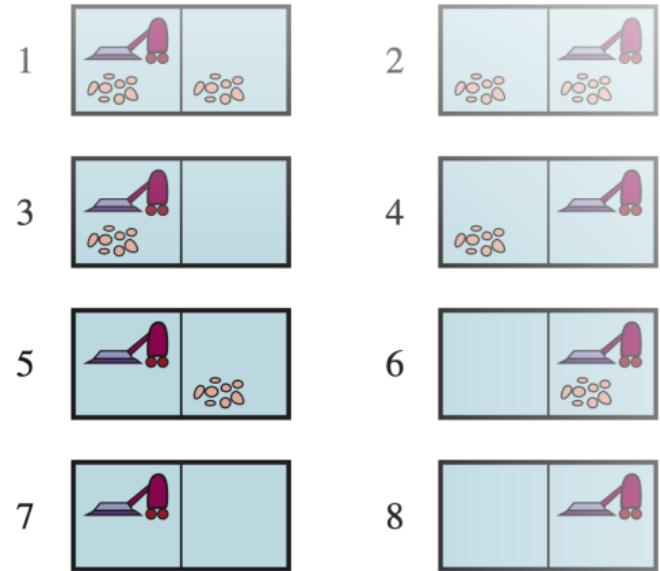
Sensorless deterministic vacuum world

Searching in partially observable environments

- ▶ $\text{Result}(\{1, 2, 3, 4, 5, 6, 7, 8\}, \text{moveRight}) = \{2, 4, 6, 8\}$

▶ Which means that executing *moveRight* on any state $s \in S$ will yield a result in $\{2, 4, 6, 8\}$

- ▶ $\text{Result}(\{2, 4, 6, 8\}, \text{Suck}) = \{4, 8\}$



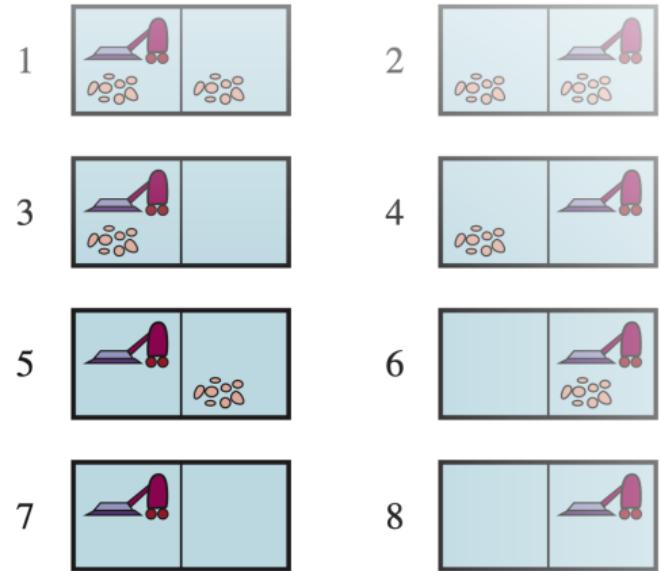
Sensorless deterministic vacuum world

Searching in partially observable environments

- ▶ $\text{Result}(\{1, 2, 3, 4, 5, 6, 7, 8\}, \text{moveRight}) = \{2, 4, 6, 8\}$

▶ Which means that executing *moveRight* on any state $s \in S$ will yield a result in $\{2, 4, 6, 8\}$

- ▶ $\text{Result}(\{2, 4, 6, 8\}, \text{Suck}) = \{4, 8\}$
- ▶ $\text{Result}(\{4, 8\}, \text{Left}) = \{1, 7\}$



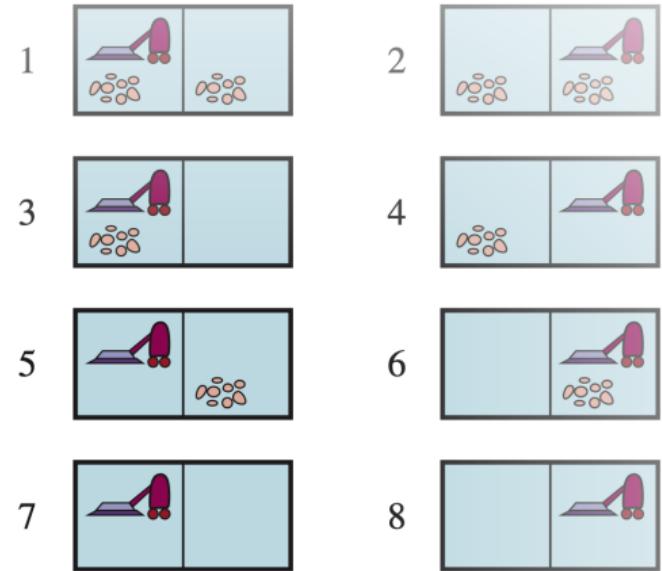
Sensorless deterministic vacuum world

Searching in partially observable environments

- ▶ $\text{Result}(\{1, 2, 3, 4, 5, 6, 7, 8\}, \text{moveRight}) = \{2, 4, 6, 8\}$

▶ Which means that executing *moveRight* on any state $s \in S$ will yield a result in $\{2, 4, 6, 8\}$

- ▶ $\text{Result}(\{2, 4, 6, 8\}, \text{Suck}) = \{4, 8\}$
- ▶ $\text{Result}(\{4, 8\}, \text{Left}) = \{1, 7\}$
- ▶ $\text{Result}(\{1, 7\}, \text{Suck}) = \{7\}$



Sensorless deterministic vacuum world

Searching in partially observable environments

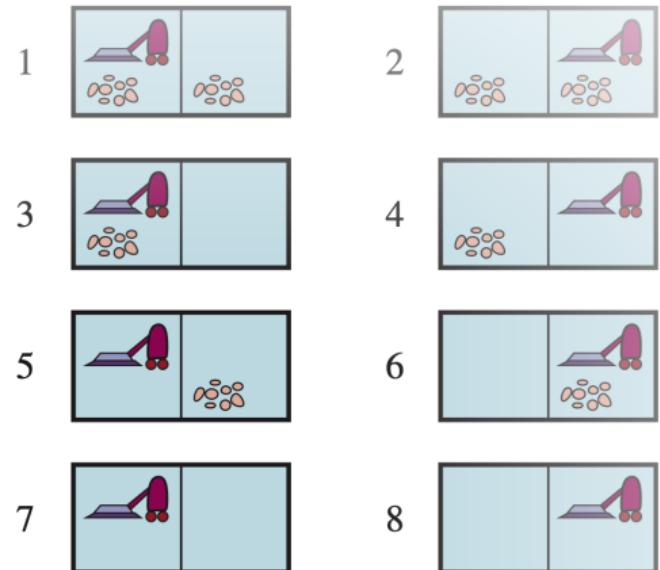
- ▶ $\text{Result}(\{1, 2, 3, 4, 5, 6, 7, 8\}, \text{moveRight}) = \{2, 4, 6, 8\}$

▶ Which means that executing moveRight on any state $s \in S$ will yield a result in $\{2, 4, 6, 8\}$

▶ $\text{Result}(\{2, 4, 6, 8\}, \text{Suck}) = \{4, 8\}$

▶ $\text{Result}(\{4, 8\}, \text{Left}) = \{1, 7\}$

▶ $\text{Result}(\{1, 7\}, \text{Suck}) = \{7\}$

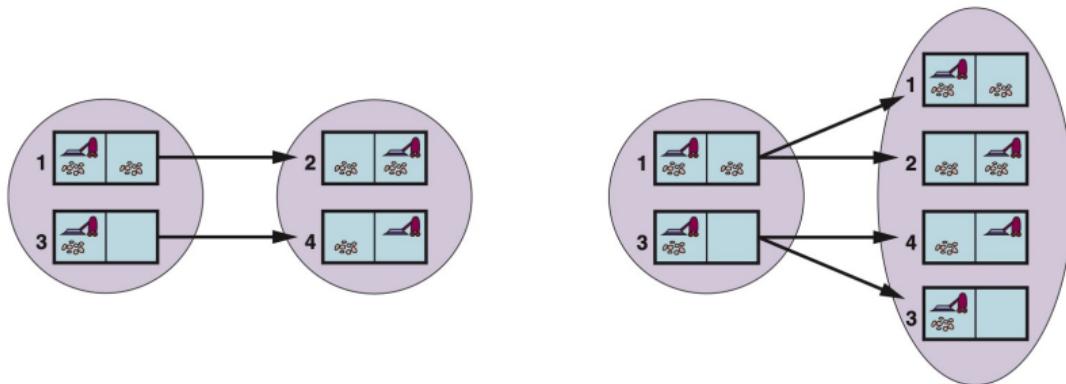


Think of 5D-chess: you solve the problem on multiple paths at the same time!

Predicting the next state with sensorless agents

Searching in partially observable environments

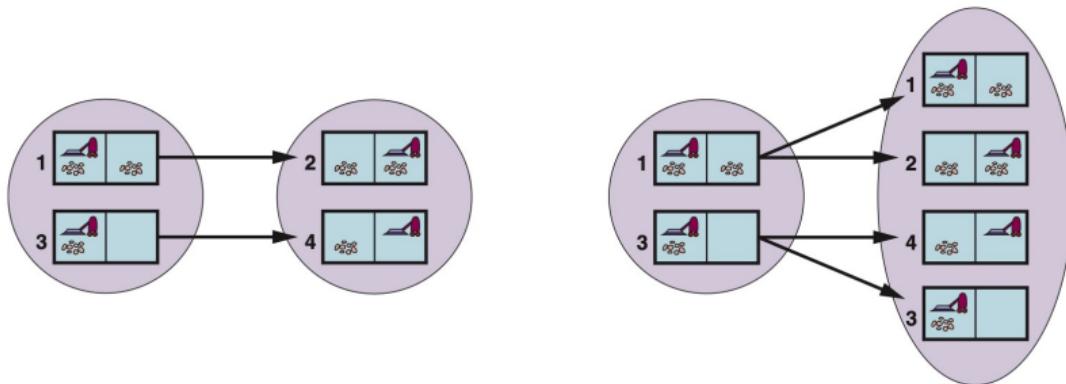
We are, in a way, making compound nodes with multiple outcomes in, where some of our actions lead to specific environment settings inside those belief states.



Predicting the next state with sensorless agents

Searching in partially observable environments

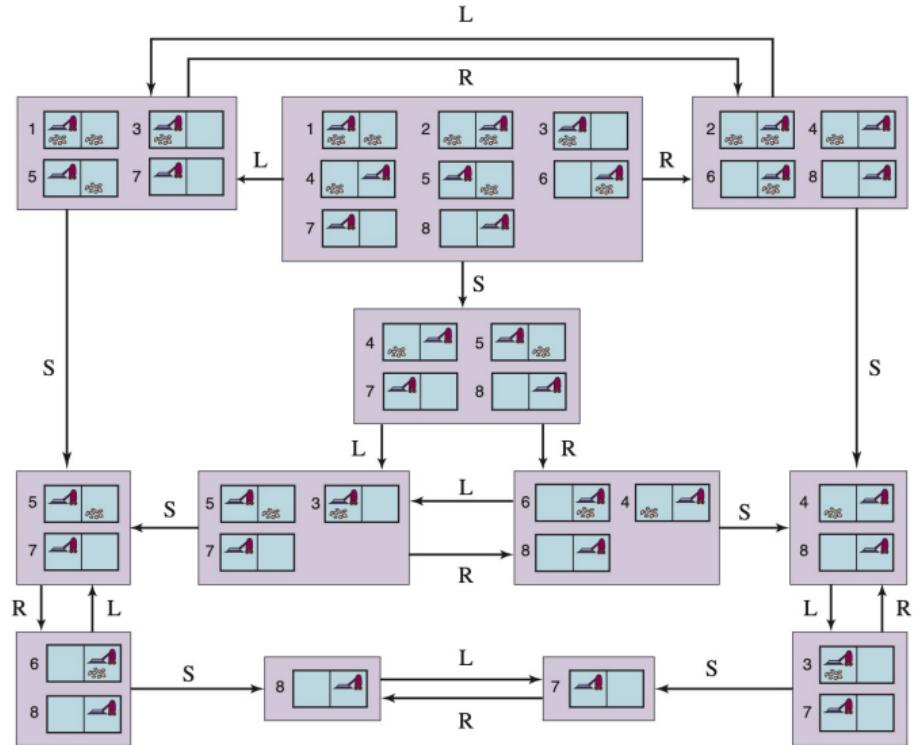
We are, in a way, making compound nodes with multiple outcomes in, where some of our actions lead to specific environment settings inside those belief states.



Of course it can be **both** nondeterministic and partially observable!

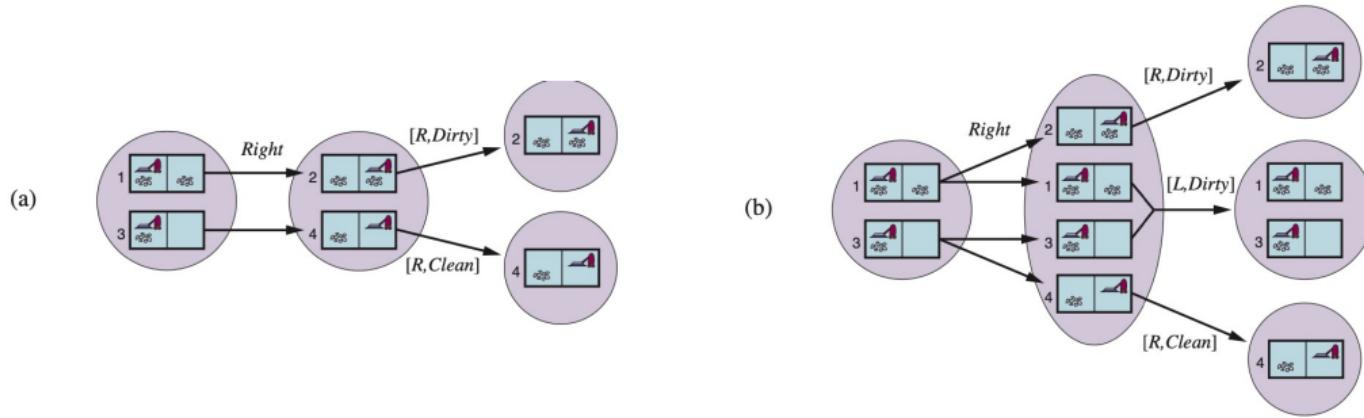
Searching through the belief space in deterministic environments

If we have a deterministic setting, we can use an ordinary search algorithm.



Searching through the belief space in partially observable environments

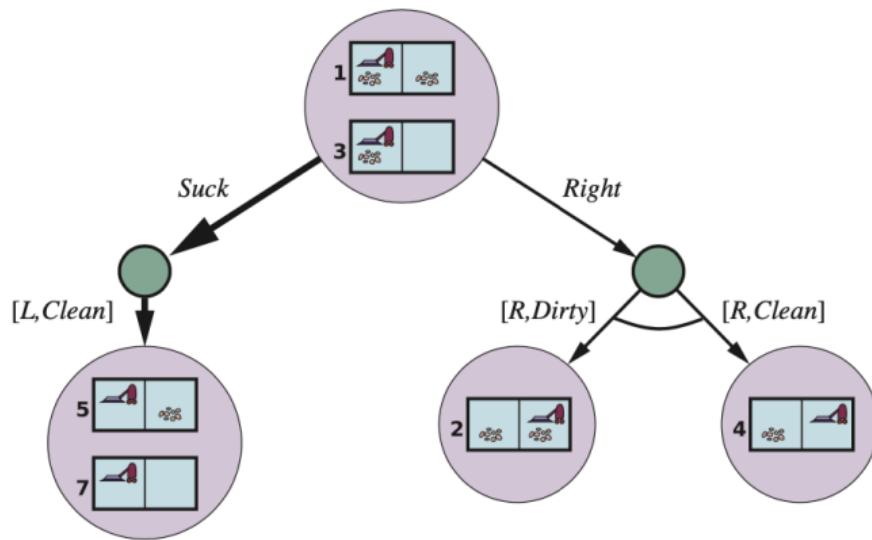
With sensors



- ▶ The agent knows where it is and see the dirt (if any) on its spot
- ▶ The **transition model** becomes a **function** of a **belief state**, an **action**, and a **another belief state**
 - ▶ In case of nondeterminism (right), we do like Dr. Strange and consider possible outcomes on different universes. **How?**

Searching through the belief space in partially observable environments

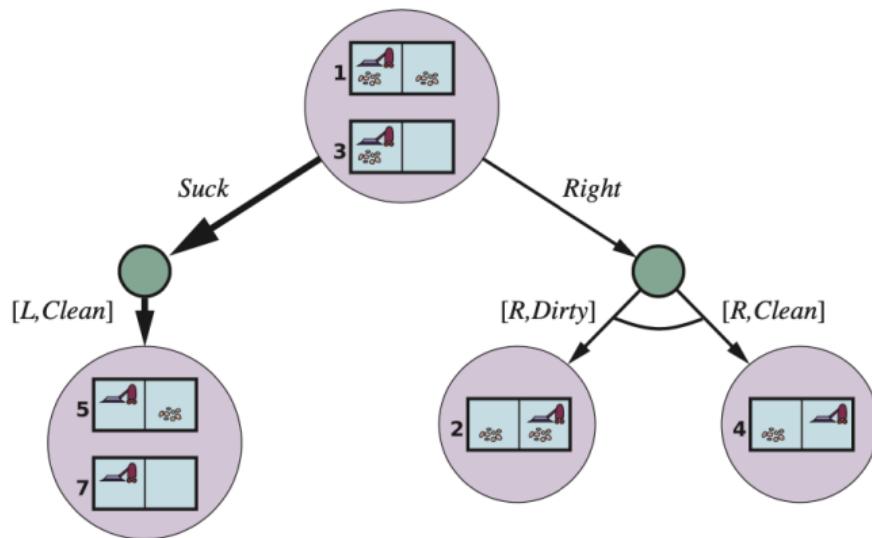
With sensors, in a nondeterministic world



- ▶ Using an **AND-OR tree**

Searching through the belief space in partially observable environments

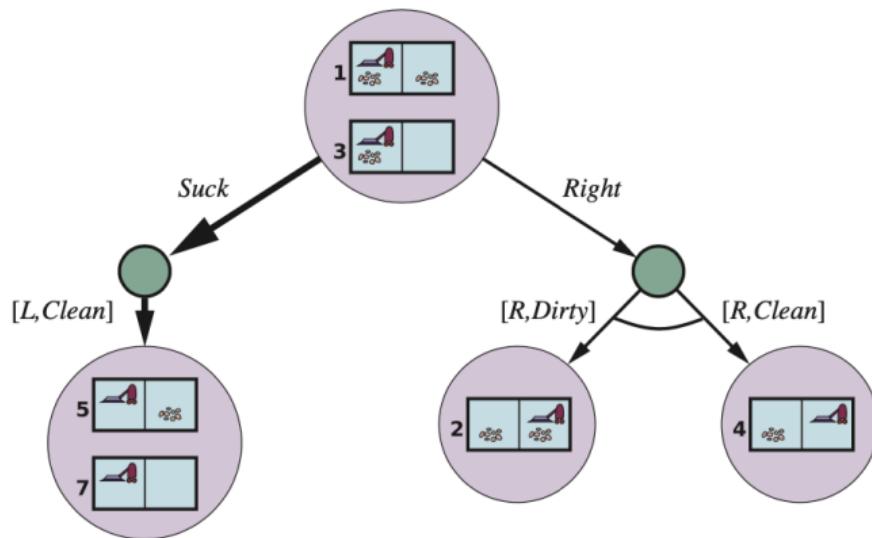
With sensors, in a nondeterministic world



- ▶ Using an **AND-OR tree**
- ▶ Notice how the nodes are now **belief states**

Searching through the belief space in partially observable environments

With sensors, in a nondeterministic world



- ▶ Using an **AND-OR tree**
- ▶ Notice how the nodes are now **belief states**
- ▶ The solution is a conditional **plan**

Constraint Satisfaction Problems

Ole C. Eidheim

September 12, 2024

Department of Computer Science

Motivation: chess

- ChatGPT plays chess

Motivation: chess

- ChatGPT plays chess
 - What is ChatGPT missing?

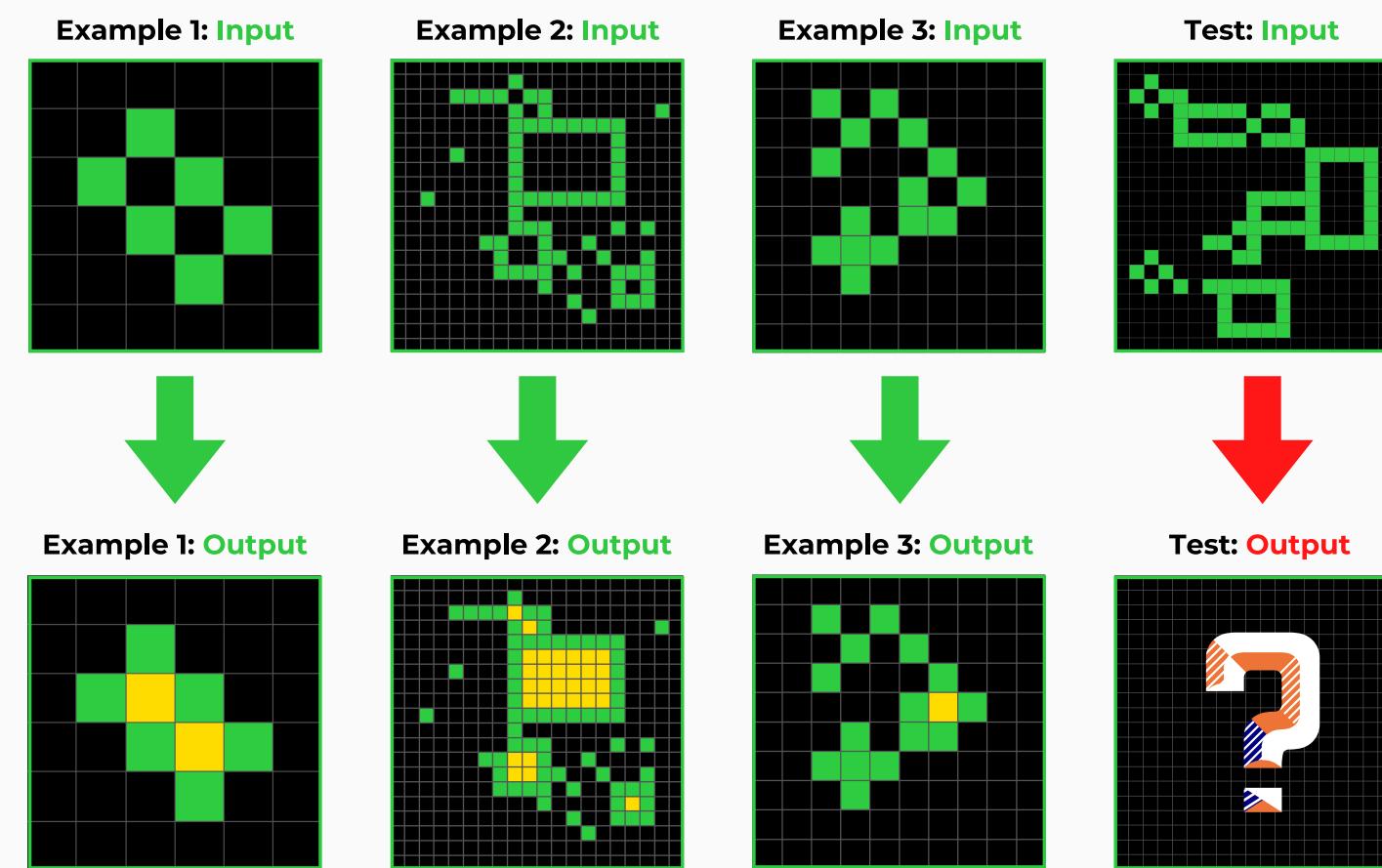
Motivation: chess

- ChatGPT plays chess
 - What is ChatGPT missing?
 - Search!

Motivation: Abstraction & Reasoning Corpus (ARC) challenge

From ARC challenge:

- ARC evaluates an AI's ability to tackle each task from scratch, using only the kind of prior knowledge about the world that humans naturally possess, known as core knowledge.
- Modern deep-learning models and large language models score near zero on ARC, highlighting the need for innovative approaches to reach human-level AI.



Motivation: the Zebra Puzzle

Five persons of different nationalities and with different jobs live in consecutive houses. The houses are painted in different colors, and the persons have different pets and favorite drinks. Additionally:

- The English lives in a red house
- The Spaniard owns a dog
- The Japanese is a painter
- The Ukrainian drinks tea
- The Norwegian lives in the first house
- The green house immediately to the right of the white one
- The photographer owns snails
- The diplomat lives in the yellow house
- Milk is drunk in the middle house
- The owner of the green house drinks coffee
- The Norwegian's house is next to the blue one
- The violinist drinks orange juice
- The fox is in a house next to that of the physician
- The horse is in a house next to that of the diplomat

Who owns a zebra? And whose favorite drink is water?

Outline

Review

Constraint Satisfaction Problems

Inference: reducing domains

Heuristics: selecting variable and ordering values

Looking backward

Local search

Tree-structured CSPs

Reducing the constraint graph to a tree

Definitions

- initialState : starting state
- $\text{actions}(s)$: possible actions at state s
- $\text{cost}(s, a)$: cost of taking action a at state s
- $\text{result}(s, a)$: next state after taking action a at state s
- $\text{isGoal}(s)$: is state s an end state?

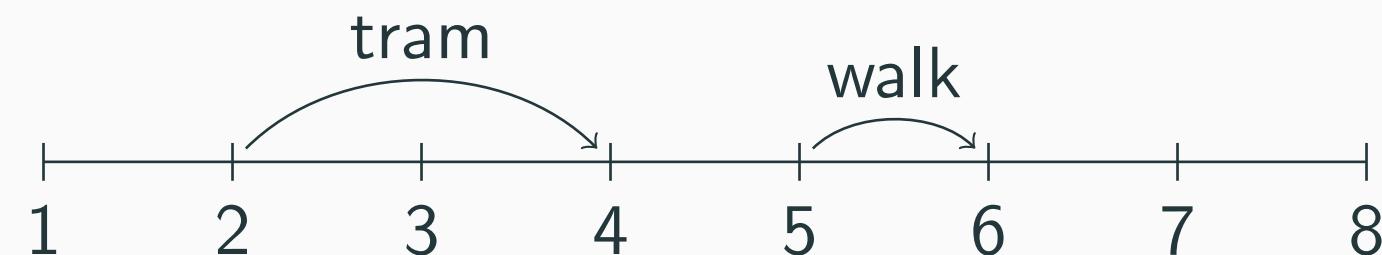
Objective: find a state path with the lowest sum of costs from initialState to a state s that satisfies $\text{isGoal}(s)$

Example: transportation problem

- Places numbered from 1 to P
- Walking from place p to $p + 1$ takes 1 minute
- Taking the magic tram from place p to $2p$ takes 3 minutes

How to travel from place 1 to P in the least amount of time?

Example actions:

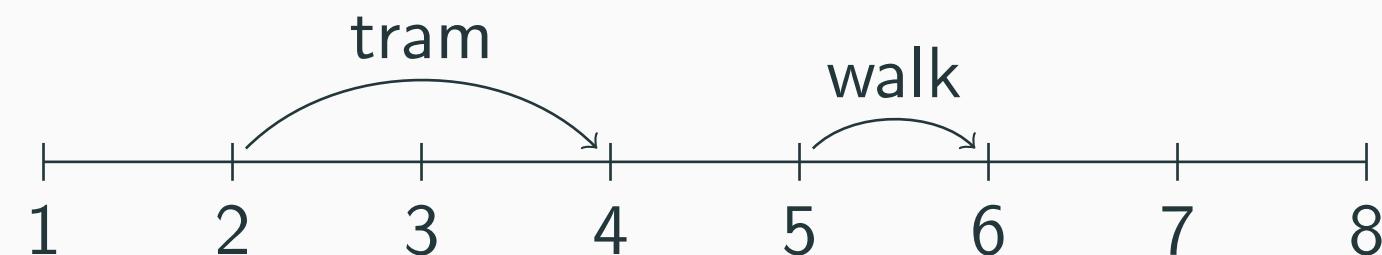


Example: transportation problem

- Places numbered from 1 to P
- Walking from place p to $p + 1$ takes 1 minute
- Taking the magic tram from place p to $2p$ takes 3 minutes

How to travel from place 1 to P in the least amount of time?

Example actions:



Example code solution using backtracking

Search Algorithms

Search Algorithm	Cost assumptions
Backtracking	None
Depth-first search	$\text{cost}(s, a) = 0$
Breadth-first search	$\text{cost}(s, a) = c, c \geq 0$
Depth-first search with iterative deepening	$\text{cost}(s, a) = c, c \geq 0$
Dynamic programming	None
Uniform-cost search	$\text{cost}(s, a) \geq 0$
A*	$\text{cost}(s, a) \geq 0$

where c is a constant

Outline

Review

Constraint Satisfaction Problems

Inference: reducing domains

Heuristics: selecting variable and ordering values

Looking backward

Local search

Tree-structured CSPs

Reducing the constraint graph to a tree

Search problems vs constraint satisfaction problems

- Search problems
 - Problem specific states, actions, next states, costs, end state, search function, and heuristics
- Constraint satisfaction problems
 - A set of constraints that must be satisfied
 - Problem specific variables, variable domains, and constraints
 - Problem independent search function, inference and heuristics
 - Example problem: Sudoku
 - Variables: the cells
 - Variable domains (values a cell may have): $\{1, \dots, 9\}$
 - Constraints: different values horizontally, vertically, and within 3x3 blocks

		3	2	6		
9		3	5			1
	1	8	6	4		
	8	1	2	9		
7						8
	6	7	8	2		
	2	6	9	5		
8		2	3			9
	5	1	3			

Constraint Satisfaction Problems

Definitions

- Variables X_1, \dots, X_n
 - The variables might have different names, e.g. A, B, \dots
- Domains D_1, \dots, D_n of each variable $X_i \in D_i$, i.e. values each variable can have
- Constraints C_1, \dots, C_m with each $C_j \in \{0, 1\}$

For example: $C_1(X_1, X_2) = \begin{cases} 1, & \text{if } X_1 \neq X_2 \\ 0, & \text{otherwise} \end{cases}$

or in book notation: $C_1 : \langle (X_1, X_2), X_1 \neq X_2 \rangle$

or simply $C_1 : X_1 \neq X_2$

Objective: find a complete assignment $\{X_1 = v_1, \dots, X_n = v_n\}$ that satisfies all constraints

Example: coloring problem

Assign red, green or blue to each region, but neighboring regions cannot have the same color

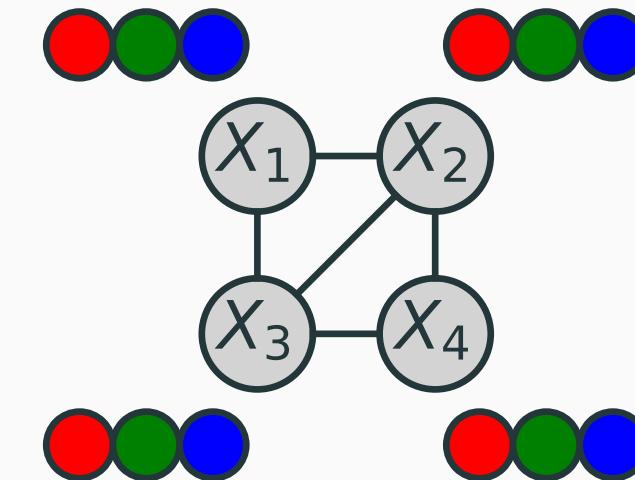


Example: coloring problem

Assign red, green or blue to each region, but neighboring regions cannot have the same color



For simplicity, ignoring NSW, V, T



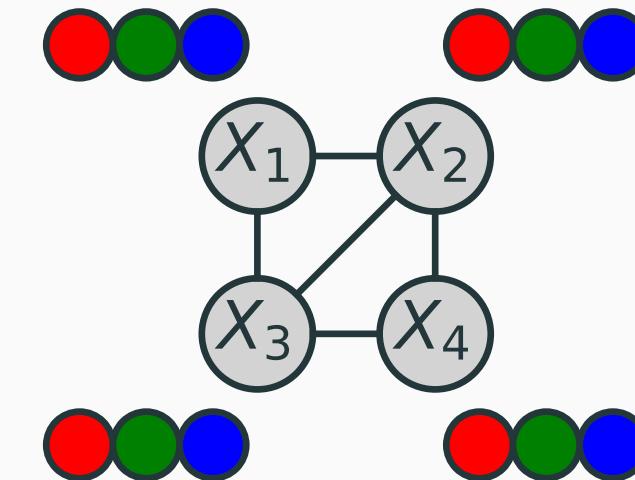
Constraint graph with domains

Example: coloring problem

Assign red, green or blue to each region, but neighboring regions cannot have the same color



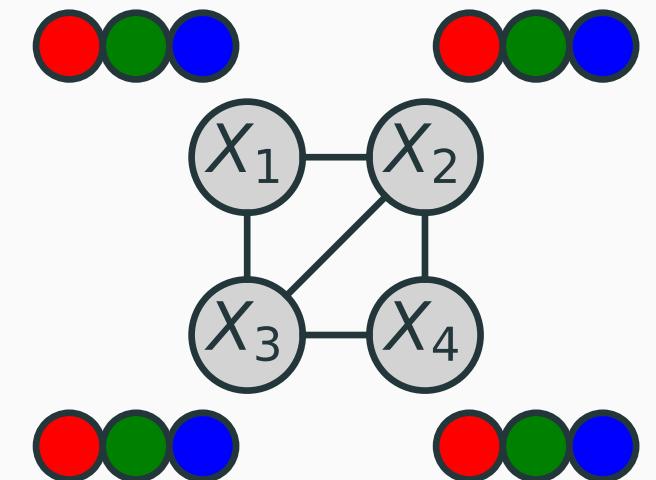
For simplicity, ignoring NSW, V, T



Constraint graph with domains

Example code solution (CSP implementation not shown)

More on constraints: coloring problem



- Variables: X_1, X_2, X_3, X_4
- Domains: $D_1 = \dots = D_4 = \{red, green, blue\}$
- Constraints: $\{X_1 \neq X_2, X_1 \neq X_3, X_2 \neq X_3, X_2 \neq X_4, X_3 \neq X_4\}$
or as tuples of values:

```
{<math>\langle (X_1, X_2), \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\} \rangle,</math>
 <math>\langle (X_1, X_3), \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\} \rangle,</math>
 <math>\langle (X_2, X_3), \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\} \rangle,</math>
 <math>\langle (X_2, X_4), \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\} \rangle,</math>
 <math>\langle (X_3, X_4), \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\} \rangle</math>}
```

More on constraints: a simpler problem

- Variables: X_1, X_2
- Domains: $D_1 = \{1, 2, 3, 4, 5\}, D_2 = \{1, 2\}$
- Constraints: $\{X_1 + X_2 = 4\}$
or as tuples of values:
 $\{\langle(X_1, X_2), \{(2, 2), (3, 1)\}\rangle\}$

Global constraints

- *alldiff(variables)*: inequality constraints between the *variables*
 - For example: $\text{alldiff}(X_1, X_2, X_3) = \{X_1 \neq X_2, X_1 \neq X_3, X_2 \neq X_3\}$

Backtracking search for Constraint Satisfaction Problems

Algorithm

```
function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value is consistent with assignment then
      add  $\{var = value\}$  to assignment
      inferences  $\leftarrow$  INFERENCE(csp, var, assignment)
      if inferences  $\neq$  failure then
        add inferences to csp
        result  $\leftarrow$  BACKTRACK(csp, assignment)
        if result  $\neq$  failure then return result
        remove inferences from csp
        remove  $\{var = value\}$  from assignment
  return failure
```

Outline

Review

Constraint Satisfaction Problems

Inference: reducing domains

Heuristics: selecting variable and ordering values

Looking backward

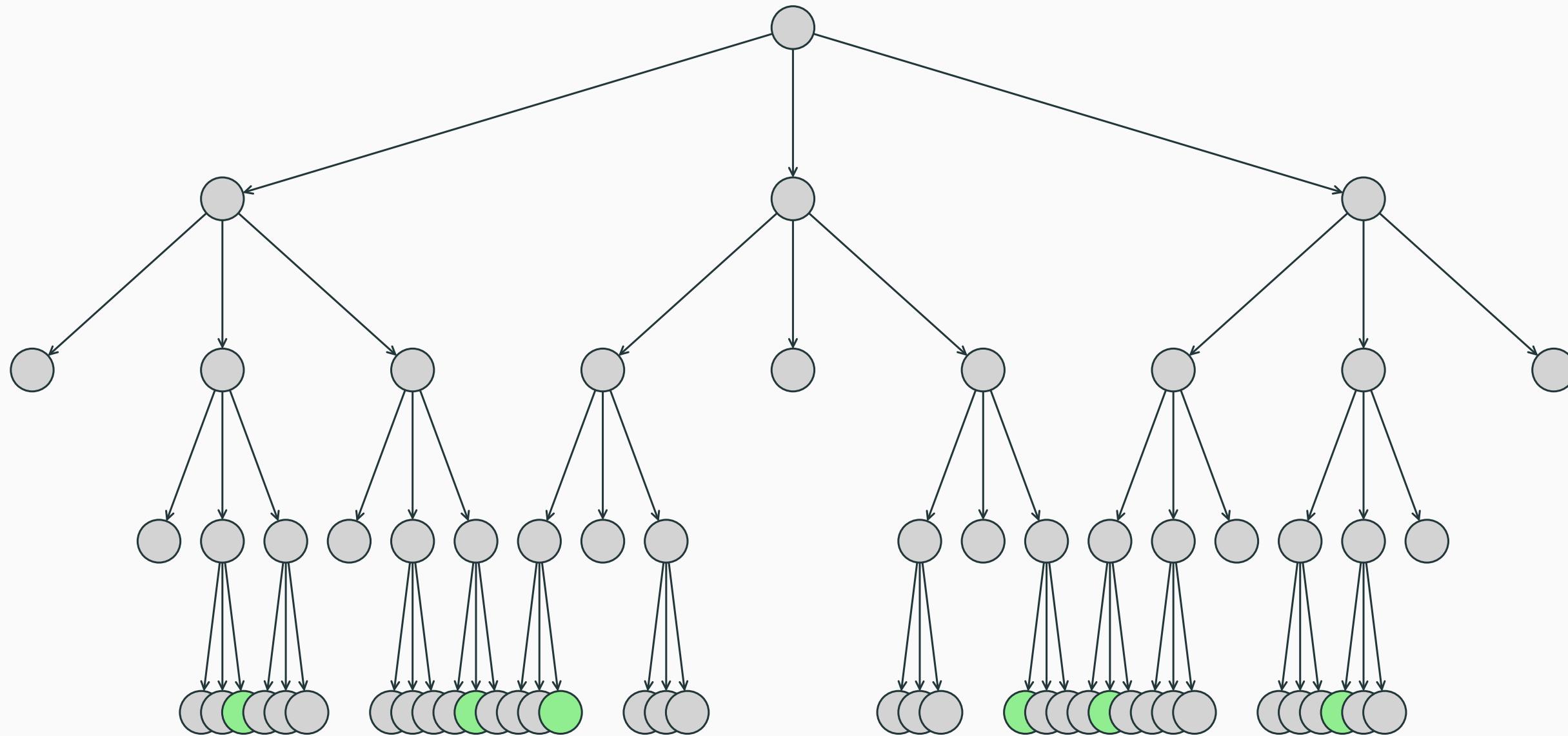
Local search

Tree-structured CSPs

Reducing the constraint graph to a tree

Motivation: simplify search by solving subnetworks

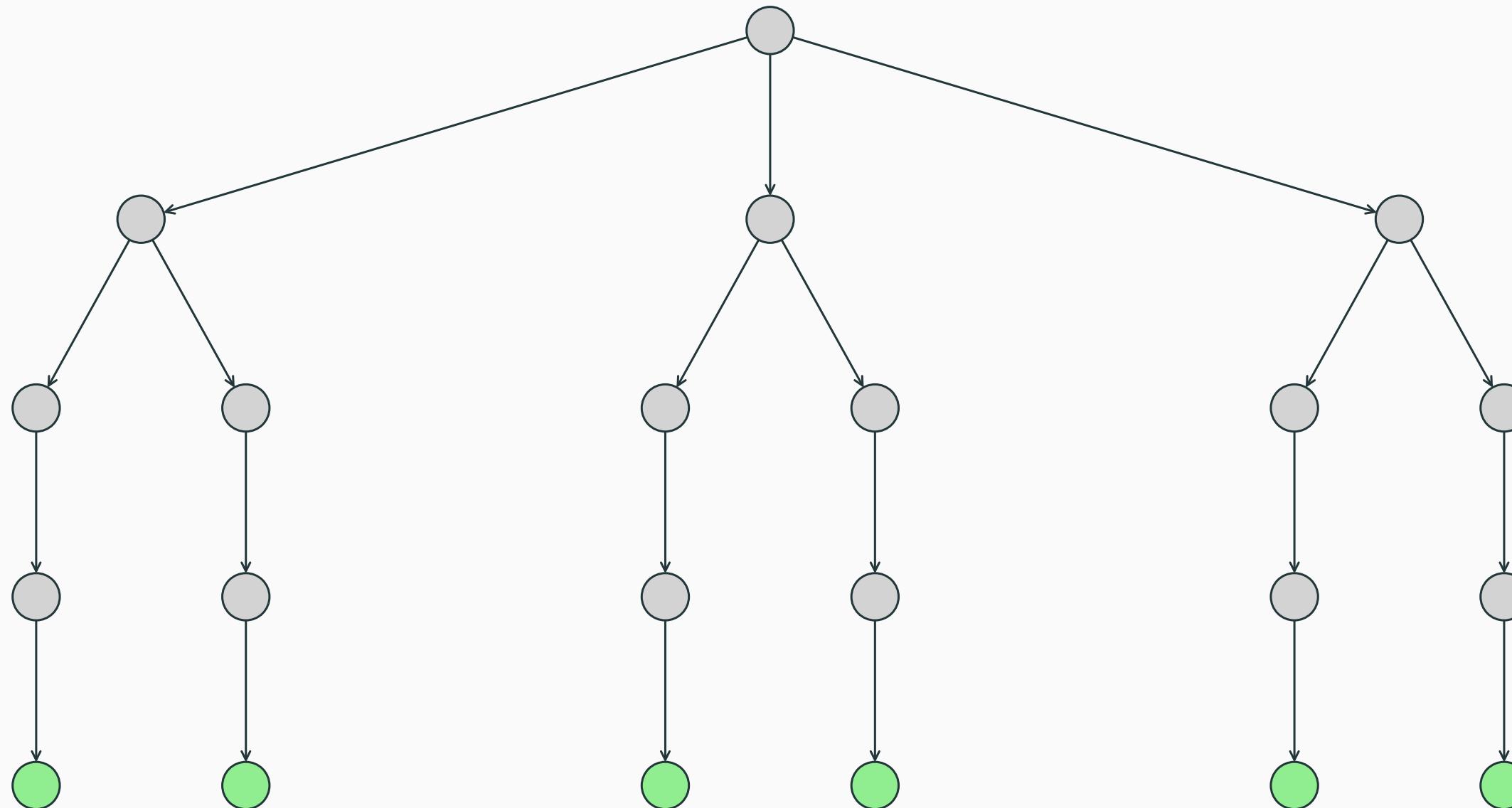
Without inference:



Example search tree with all solutions shown

Motivation: simplify search by solving subnetworks

With inference:



Example search tree with all solutions shown

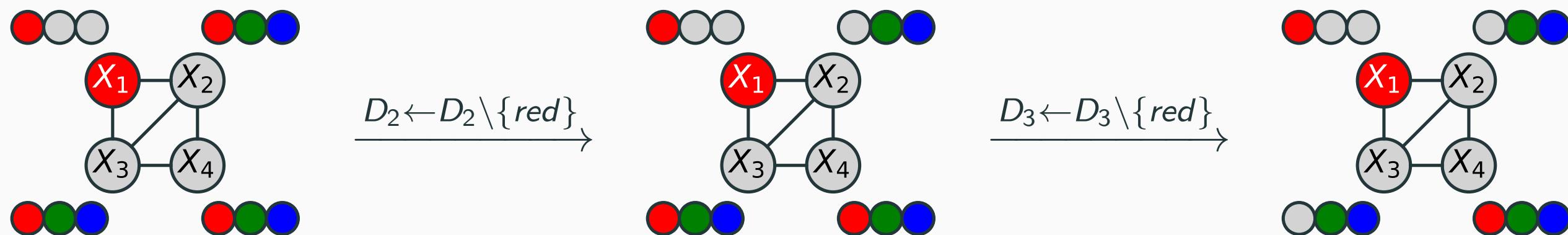
Forward checking

Algorithm

After assigning a value to a variable, explore the domains of neighboring unassigned variables, and remove values whose assignments would violate a constraint

- Neighboring variables: variables that share constraint dependencies
- During inference, domains are also set to the assigned value

For example, after assigning $\{X_1 = \text{red}\}$ (and $D_1 \leftarrow \{\text{red}\}$):



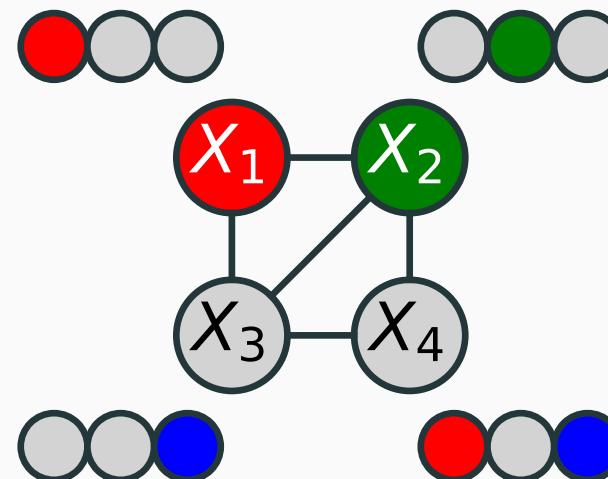
Arc Consistency Algorithm #3 (AC-3)

Arc(directed edge)-consistency

X_i is arc-consistent with X_j if for every value in D_i there exists a value in D_j whose assignments would not violate any constraint

- Only binary (two-variable) constraints considered

For example, is (X_3, X_4) arc-consistent (is X_3 arc-consistent with X_4)?



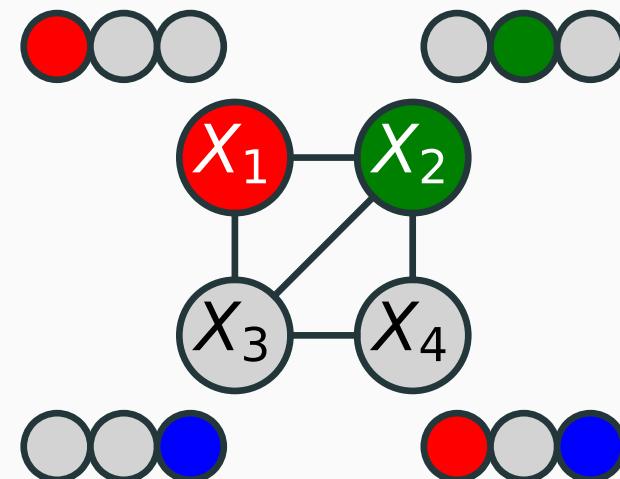
Arc Consistency Algorithm #3 (AC-3)

Arc(directed edge)-consistency

X_i is arc-consistent with X_j if for every value in D_i there exists a value in D_j whose assignments would not violate any constraint

- Only binary (two-variable) constraints considered

For example, is (X_3, X_4) arc-consistent (is X_3 arc-consistent with X_4)?



Answer: yes

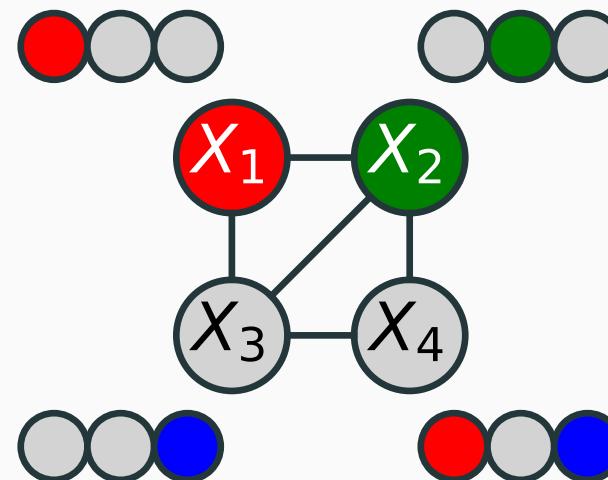
Arc Consistency Algorithm #3 (AC-3)

Arc(directed edge)-consistency

X_i is arc-consistent with X_j if for every value in D_i there exists a value in D_j whose assignments would not violate any constraint

- Only binary (two-variable) constraints considered

For example, is (X_4, X_3) arc-consistent?



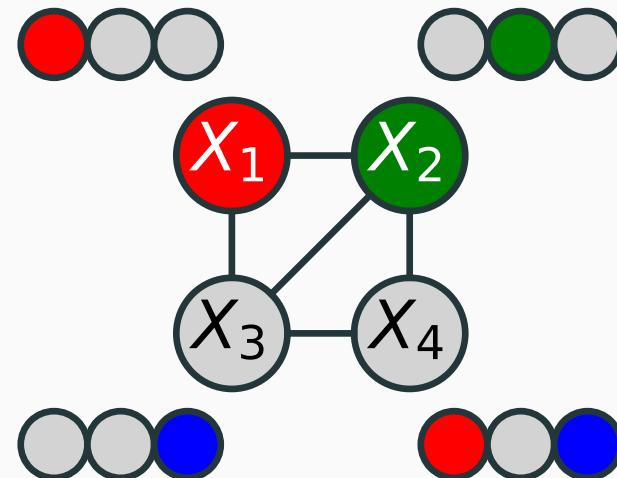
Arc Consistency Algorithm #3 (AC-3)

Arc(directed edge)-consistency

X_i is arc-consistent with X_j if for every value in D_i there exists a value in D_j whose assignments would not violate any constraint

- Only binary (two-variable) constraints considered

For example, is (X_4, X_3) arc-consistent?



Answer: no

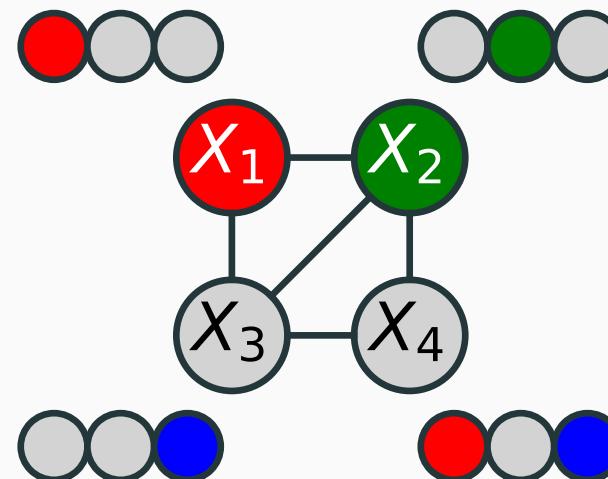
Arc Consistency Algorithm #3 (AC-3)

Arc(directed edge)-consistency

X_i is arc-consistent with X_j if for every value in D_i there exists a value in D_j whose assignments would not violate any constraint

- Only binary (two-variable) constraints considered

How do we make (X_4, X_3) arc-consistent?



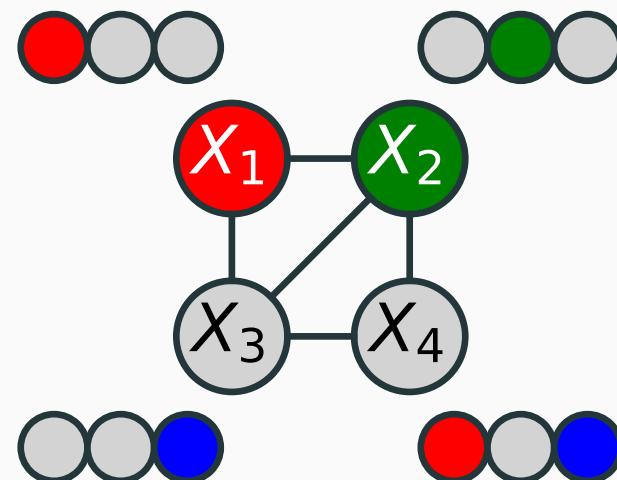
Arc Consistency Algorithm #3 (AC-3)

Arc(directed edge)-consistency

X_i is arc-consistent with X_j if for every value in D_i there exists a value in D_j whose assignments would not violate any constraint

- Only binary (two-variable) constraints considered

How do we make (X_4, X_3) arc-consistent?



Answer: $D_4 \leftarrow D_4 \setminus \text{blue}$

Arc Consistency Algorithm #3 (AC-3)

Arc(directed edge)-consistency

X_i is arc-consistent with X_j if for every value in D_i there exists a value in D_j whose assignments would not violate any constraint

- Only binary (two-variable) constraints considered

Make (X_i, X_j) arc-consistent

Remove values from D_i to make X_i arc-consistent with X_j

Algorithm

- start with a queue of initial arc(s)
- while queue is not empty
 - pop an arc and make arc-consistent
 - if the domain was reduced for a variable, add arcs between its unassigned neighbors and the variable

Arc Consistency Algorithm #3 (AC-3)

The algorithm in Figure 5.3 in the book starts with all neighboring unassigned variables in the queue (both directions initially)

This can for example solve simple Sudoku problems without search:

	4		1
3			
			4

Domains after setting the initial values:

$$D_{12} = \{4\}, D_{14} = \{1\}, D_{21} = \{3\}, D_{34} = \{4\},$$

$$D_{11} = D_{13} = D_{22} = \dots = D_{33} = D_{41} = \dots = D_{44} = \{1, 2, 3, 4\}$$

Domains after running the AC-3 algorithm:

$$D_{11} = \{2\}, D_{12} = \{4\}, D_{13} = \{3\}, D_{14} = \{1\}, D_{21} = \{3\}, D_{22} = \{1\}, D_{23} = \{4\}, D_{24} = \{2\},$$

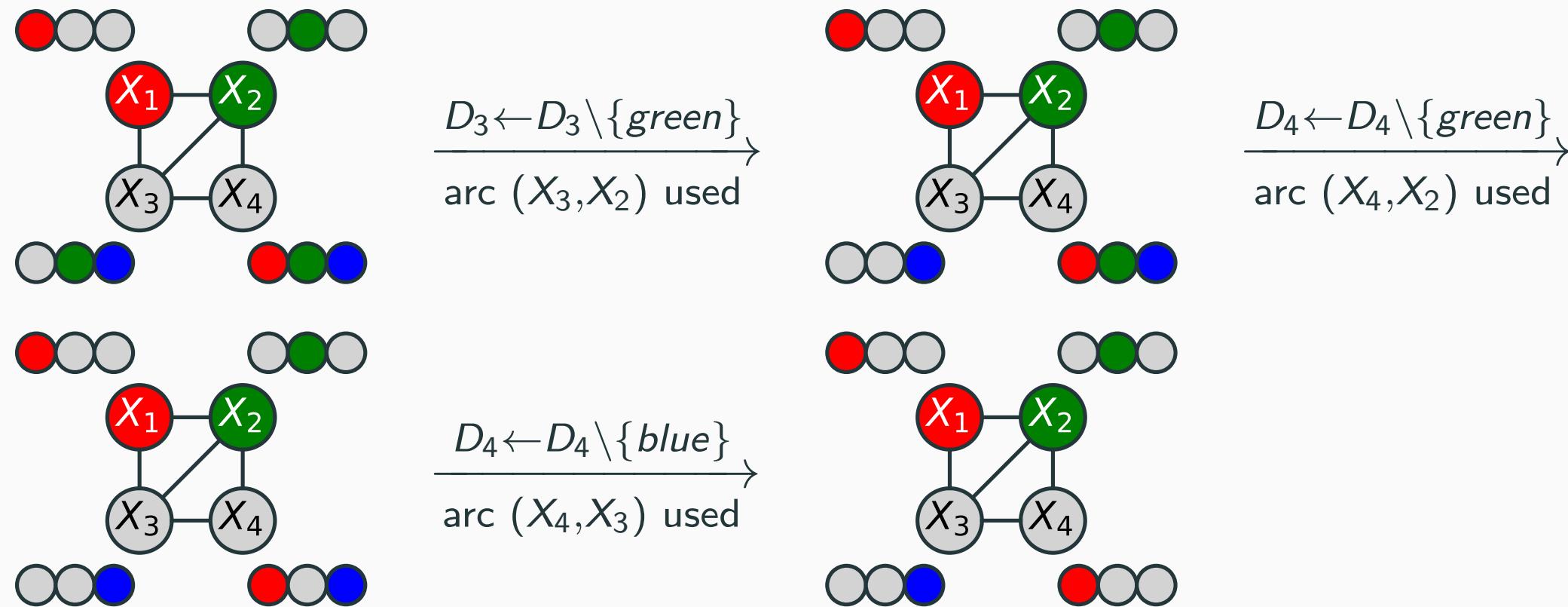
$$D_{31} = \{1\}, D_{32} = \{3\}, D_{33} = \{2\}, D_{34} = \{4\}, D_{41} = \{4\}, D_{42} = \{2\}, D_{43} = \{1\}, D_{44} = \{3\}$$

Arc Consistency Algorithm #3 (AC-3)

Another possibility is to run AC-3 for each newly assigned variable and its neighboring unassigned variables (in opposite directions)

Example, after assigning $\{X_2 = \text{green}\}$ (and $D_2 \leftarrow \{\text{green}\}$):

- Starts with two arcs in the queue: (X_3, X_2) and (X_4, X_2)
- When D_3 is updated, the arc (X_4, X_3) is added to queue
- When D_4 is updated, the arc (X_3, X_4) is added to queue



Outline

Review

Constraint Satisfaction Problems

Inference: reducing domains

Heuristics: selecting variable and ordering values

Looking backward

Local search

Tree-structured CSPs

Reducing the constraint graph to a tree

Motivation

Possibly faster search by focusing on likely choices

Backtracking search for Constraint Satisfaction Problems

Algorithm

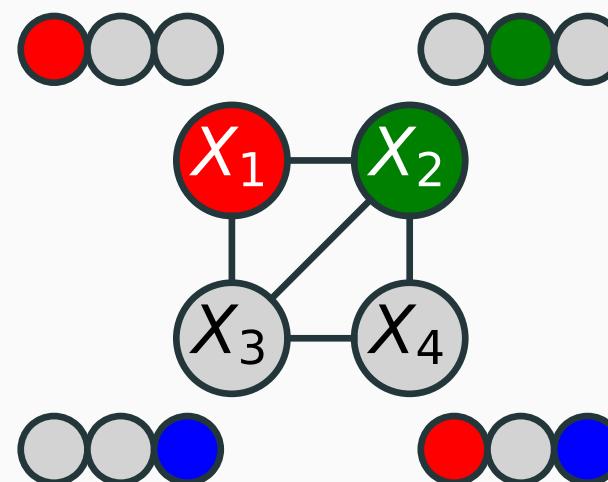
```
function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    if value is consistent with assignment then
      add  $\{var = value\}$  to assignment
      inferences  $\leftarrow$  INFERENCE(csp, var, assignment)
      if inferences  $\neq$  failure then
        add inferences to csp
        result  $\leftarrow$  BACKTRACK(csp, assignment)
        if result  $\neq$  failure then return result
        remove inferences from csp
        remove  $\{var = value\}$  from assignment
  return failure
```

Selecting variable

Choose the most constrained variable

Select the unassigned variable with the smallest remaining domain

For example, will X_3 or X_4 be chosen:

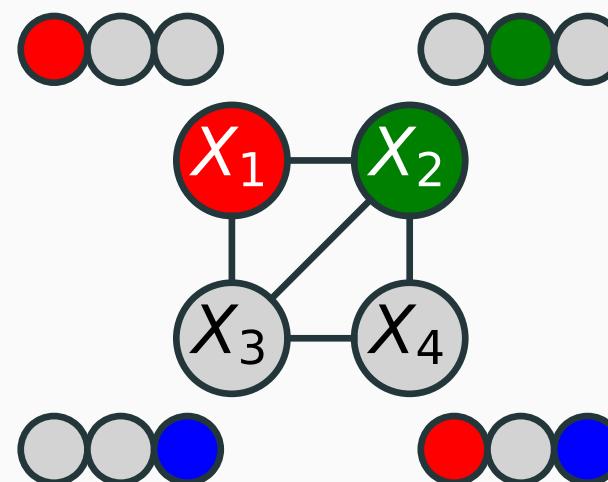


Selecting variable

Choose the most constrained variable

Select the unassigned variable with the smallest remaining domain

For example, will X_3 or X_4 be chosen:



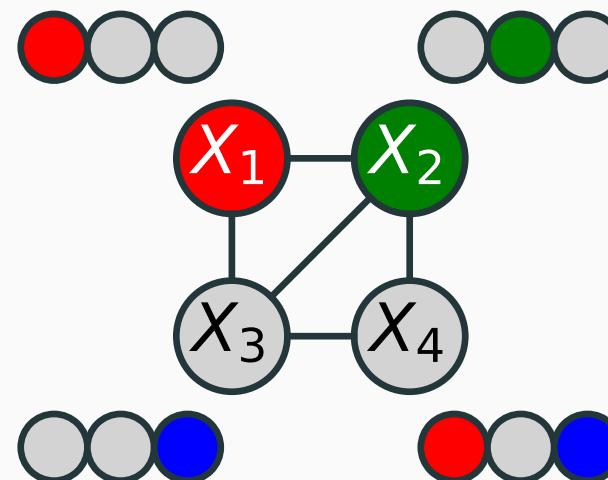
Answer: X_3

Ordering values

Choose the least constrained values first

Prefer values that would cause the smallest reduction in the unassigned variable domains during forward checking

For example, given variable X_4 was chosen, will red or blue be chosen?

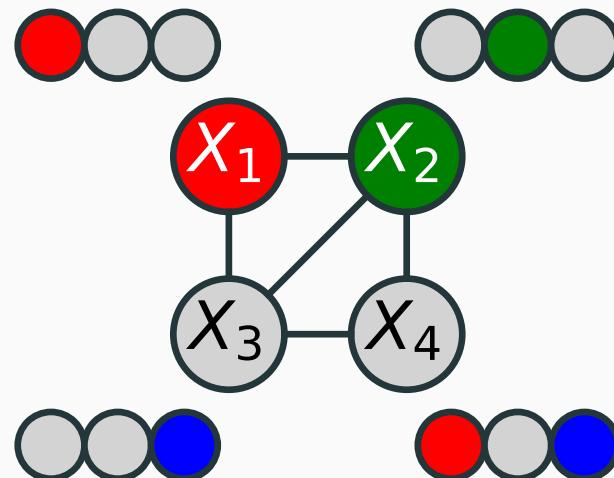


Ordering values

Choose the least constrained values first

Prefer values that would cause the smallest reduction in the unassigned variable domains during forward checking

For example, given variable X_4 was chosen, will red or blue be chosen?



Answer: red

Example: cryptarithmetic problem

Assign distinct digits to each letter such that the sum is correct, and none of the numbers have leading zeros, i.e. $T \neq 0$ and $F \neq 0$

$$\begin{array}{r} T \ W \ O \\ + \ T \ W \ O \\ \hline F \ O \ U \ R \end{array}$$

Example solution:

$$\begin{array}{r} 7 \ 6 \ 5 \\ + \ 7 \ 6 \ 5 \\ \hline 1 \ 5 \ 3 \ 0 \end{array}$$

Outline

Review

Constraint Satisfaction Problems

Inference: reducing domains

Heuristics: selecting variable and ordering values

Looking backward

Local search

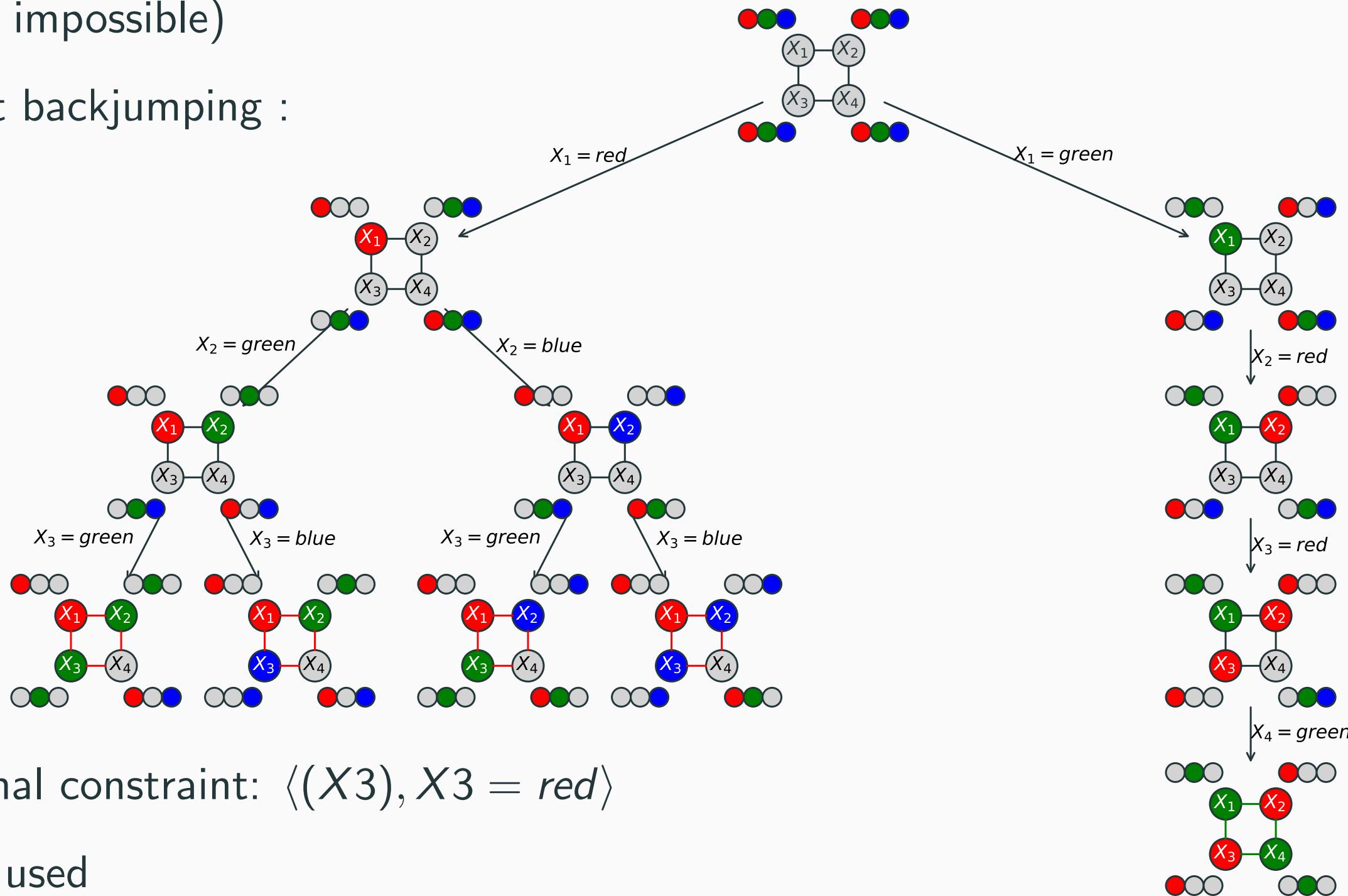
Tree-structured CSPs

Reducing the constraint graph to a tree

Backjumping

Jumps back to the most recent conflicting assignment (assignment that makes a solution impossible)

Without backjumping :



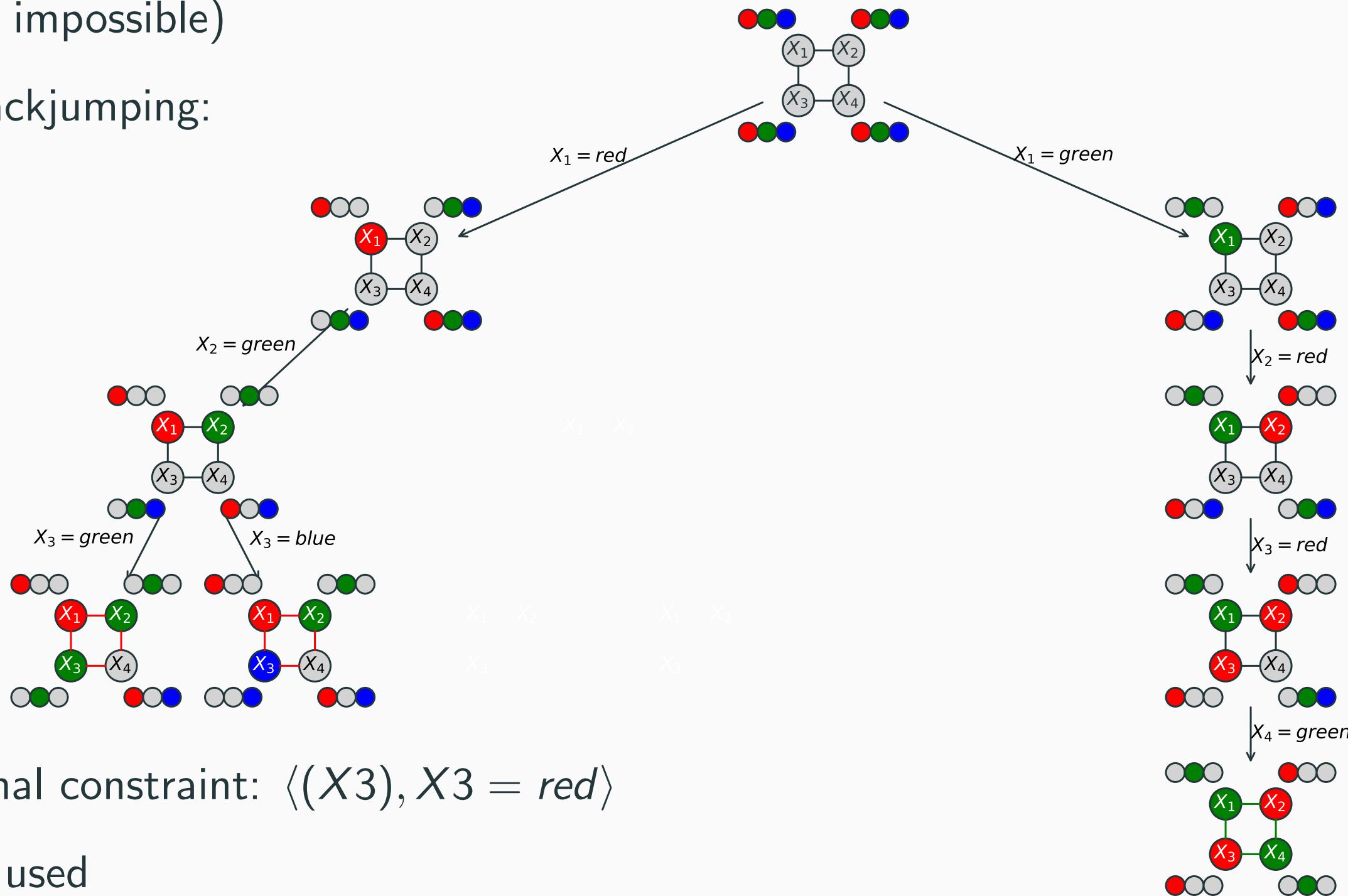
Additional constraint: $\langle (X_3), X_3 = \text{red} \rangle$

AC-3 is used

Backjumping

Jumps back to the most recent conflicting assignment (assignment that makes a solution impossible)

With backjumping:



Additional constraint: $\langle (X3), X3 = \text{red} \rangle$

AC-3 is used

Outline

Review

Constraint Satisfaction Problems

Inference: reducing domains

Heuristics: selecting variable and ordering values

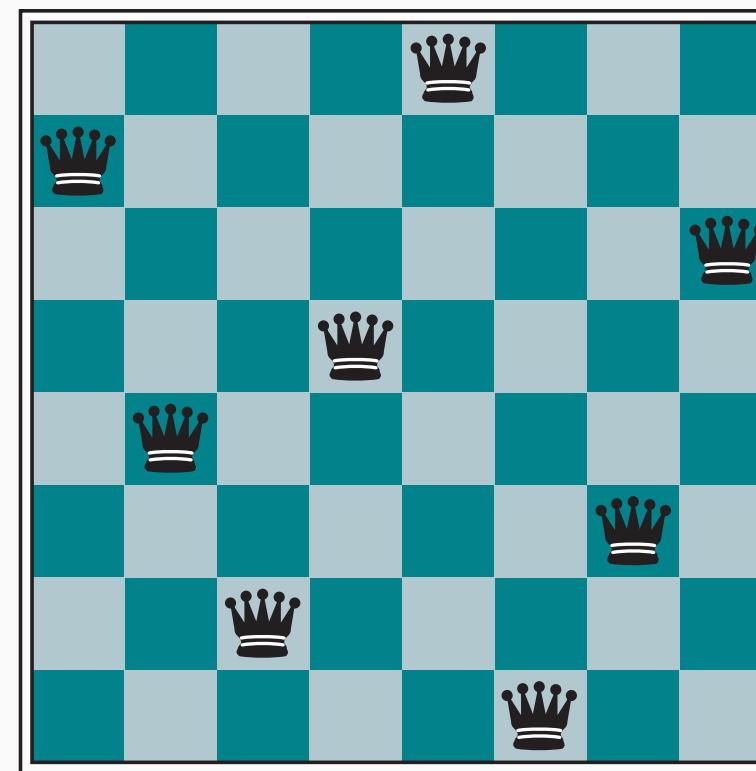
Looking backward

Local search

Tree-structured CSPs

Reducing the constraint graph to a tree

Example: 8-queens problem



Min-Conflicts

function MIN-CONFLICTS(*csp*, *max_steps*) **returns** a solution or *failure*

inputs: *csp*, a constraint satisfaction problem

max_steps, the number of steps allowed before giving up

current \leftarrow an initial complete assignment for *csp*

for *i* = 1 to *max_steps* **do**

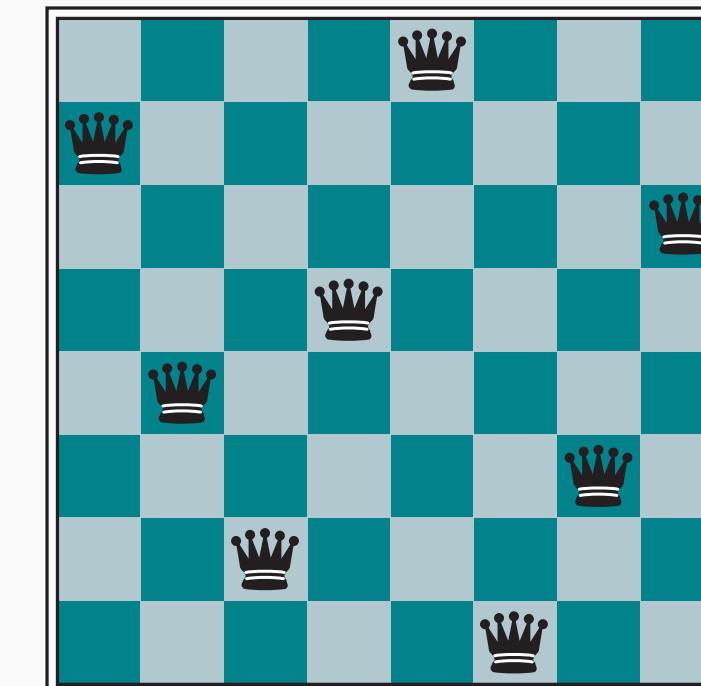
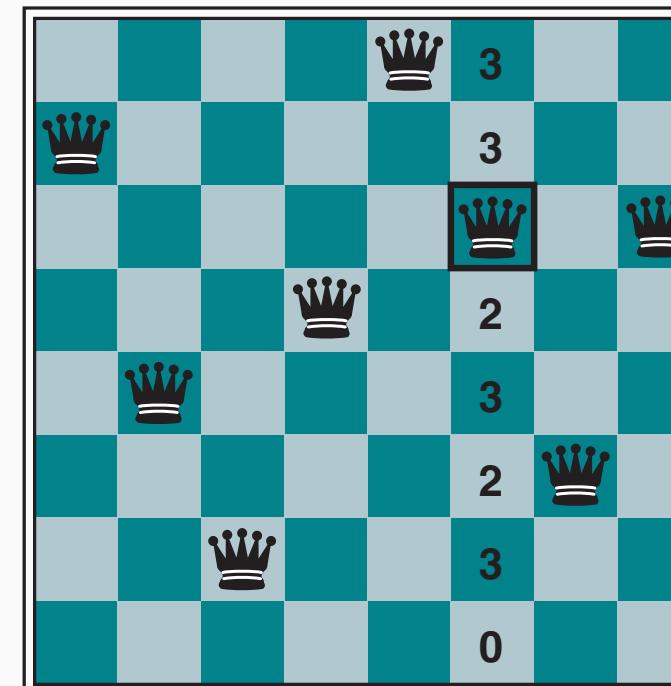
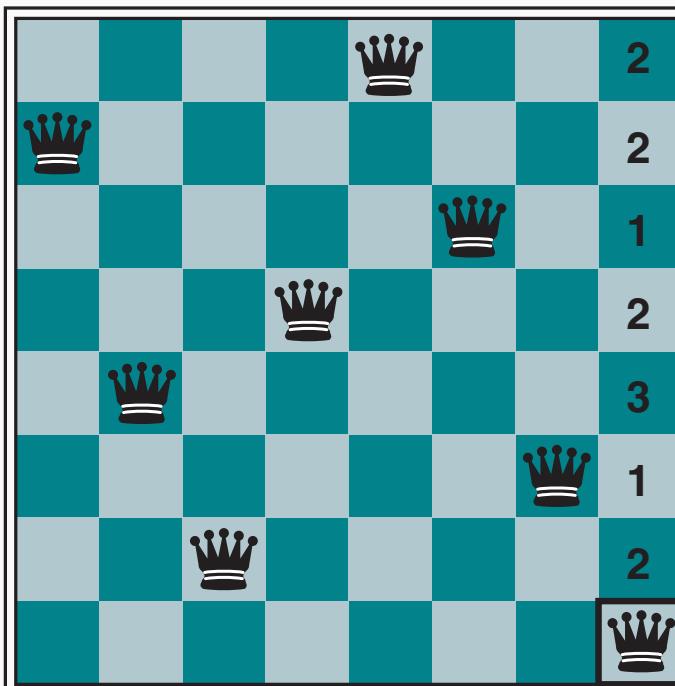
if *current* is a solution for *csp* **then return** *current*

var \leftarrow a randomly chosen conflicted variable from *csp.VARIABLES*

value \leftarrow the value *v* for *var* that minimizes CONFLICTS(*csp*, *var*, *v*, *current*)

 set *var* = *value* in *current*

return *failure*



Outline

Review

Constraint Satisfaction Problems

Inference: reducing domains

Heuristics: selecting variable and ordering values

Looking backward

Local search

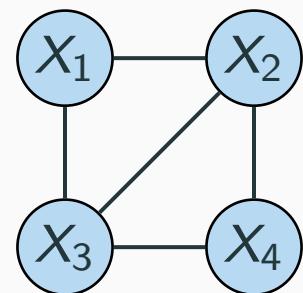
Tree-structured CSPs

Reducing the constraint graph to a tree

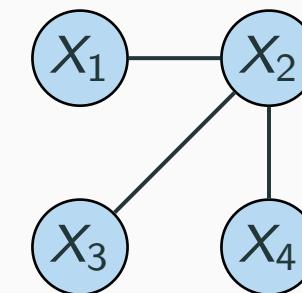
Tree-structured CSPs

Definition

Any two variables are connected by exactly one path



Not tree-structured



Tree-structured

The Tree-CSP-Solver algorithm

function TREE-CSP-SOLVER(*csp*) **returns** a solution, or *failure*

inputs: *csp*, a CSP with components X , D , C

$n \leftarrow$ number of variables in X

assignment \leftarrow an empty assignment

root \leftarrow any variable in X

$X \leftarrow \text{TOPOLOGICALSORT}(X, \text{root})$

for $j = n$ **down to** 2 **do**

 MAKE-ARC-CONSISTENT(PARENT(X_j), X_j)

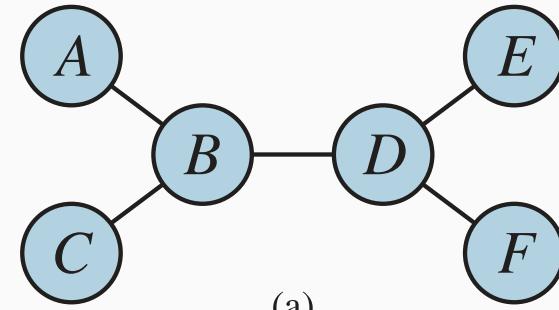
if it cannot be made consistent **then return** *failure*

for $i = 1$ **to** n **do**

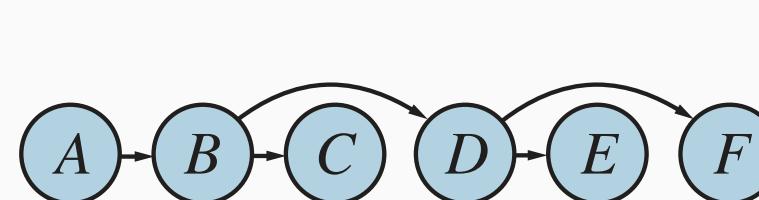
assignment[X_i] \leftarrow any consistent value from D_i

if there is no consistent value **then return** *failure*

return *assignment*



(a)



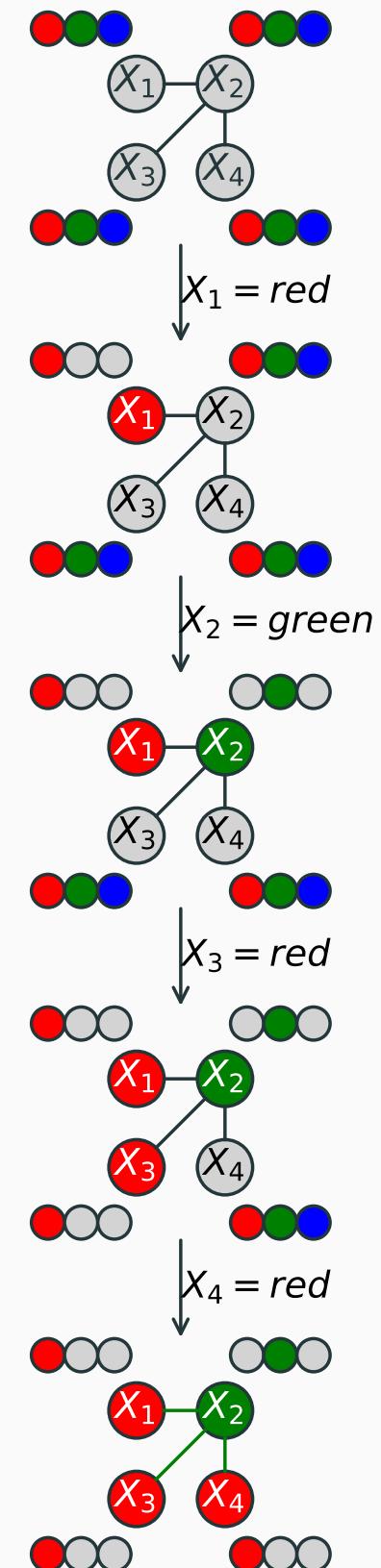
(b)

Topological sort of a) is shown in b)

If a tree-structured CSP has a solution, it will be found in linear time

The Tree-CSP-Solver algorithm

- For example:
 - $D_1 = \dots = D_4 = \{\text{red, green, blue}\}$
 - $\text{root} = X_1$
 - $\text{parent}(X_2) = X_1$
 - $\text{parent}(X_3) = X_2$
 - $\text{parent}(X_4) = X_2$
- No changes in the first for-loop of the algorithm
- The second for-loop of the algorithm →

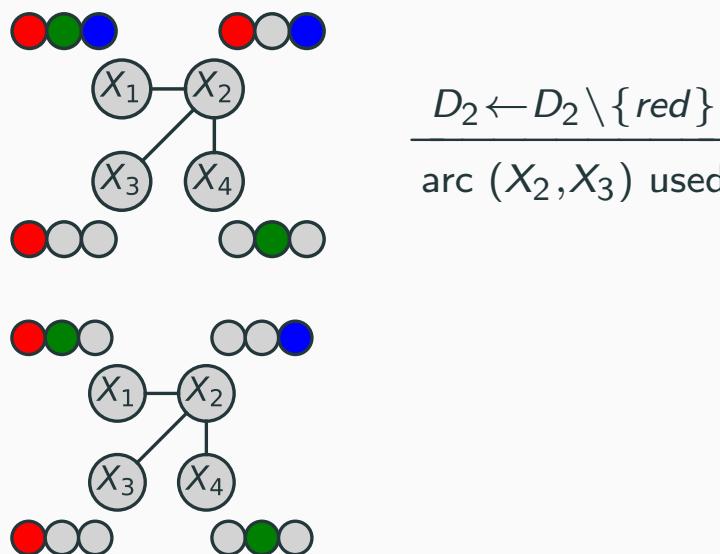
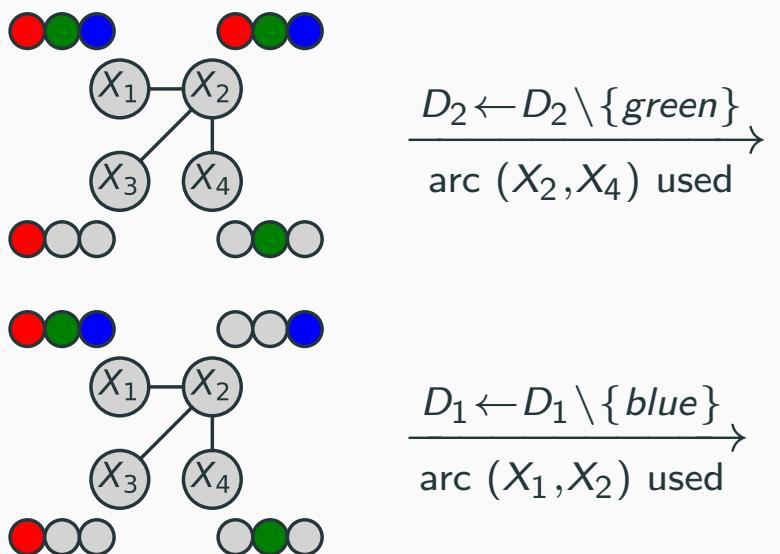


The Tree-CSP-Solver algorithm

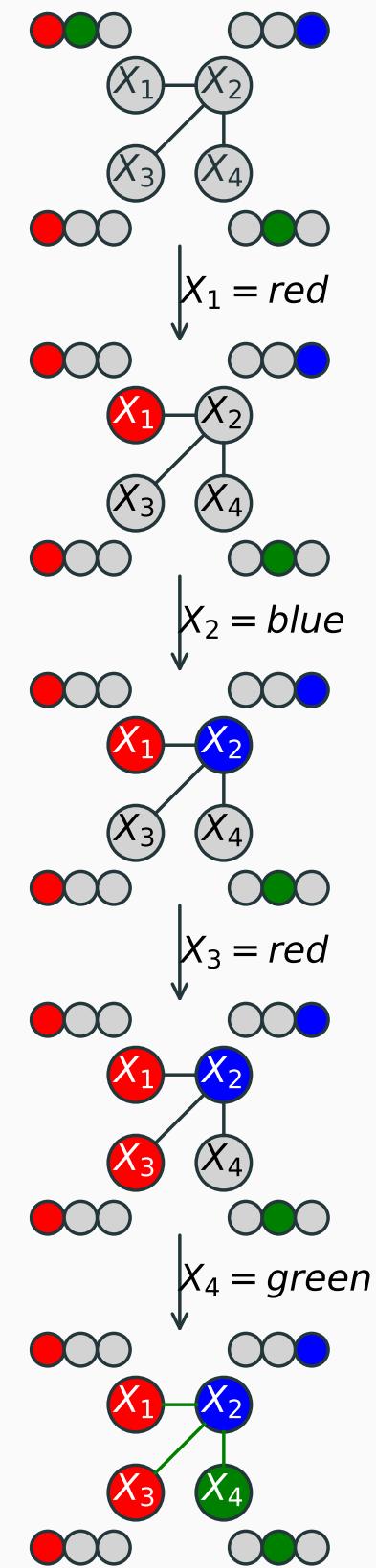
- For example:

- $D_1 = D_2 = \{red, green, blue\}$
- $D_3 = \{red\}, D_4 = \{green\}$
- $root = X_1$
- $parent(X_2) = X_1$
- $parent(X_3) = X_2$
- $parent(X_4) = X_2$

- The first for-loop of the algorithm:



- The second for-loop of the algorithm →



Outline

Review

Constraint Satisfaction Problems

Inference: reducing domains

Heuristics: selecting variable and ordering values

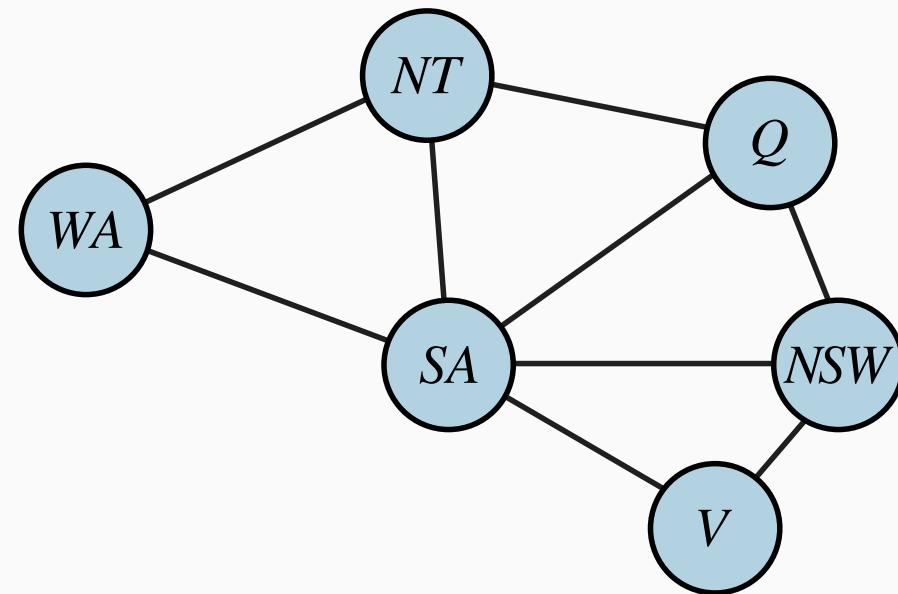
Looking backward

Local search

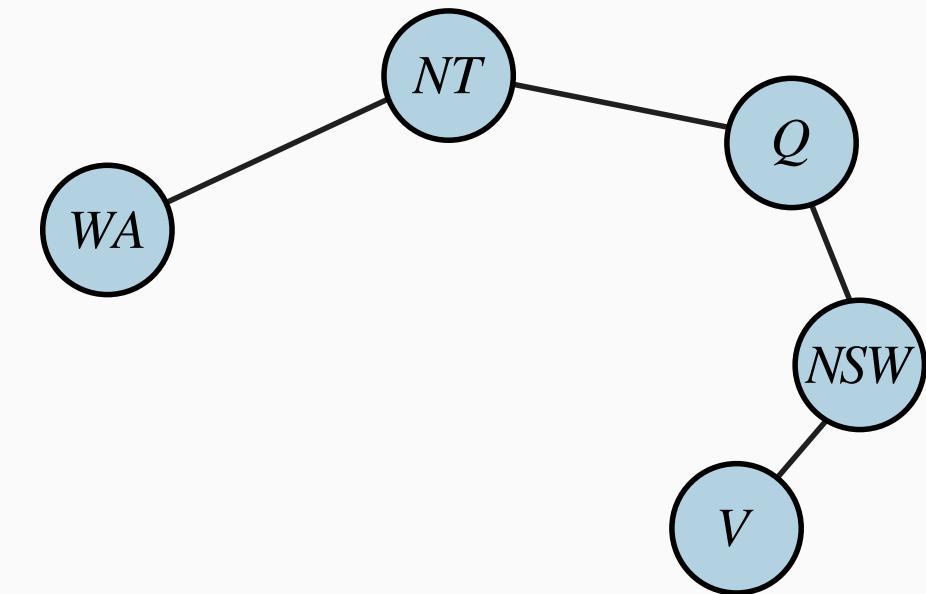
Tree-structured CSPs

Reducing the constraint graph to a tree

Cutset conditioning



The original graph



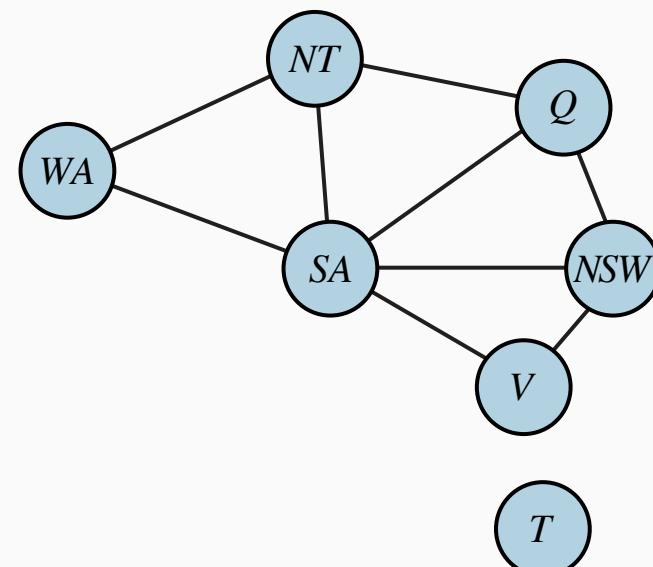
After removing the **cycle cutset** = {SA},
the graph becomes a tree

Algorithm

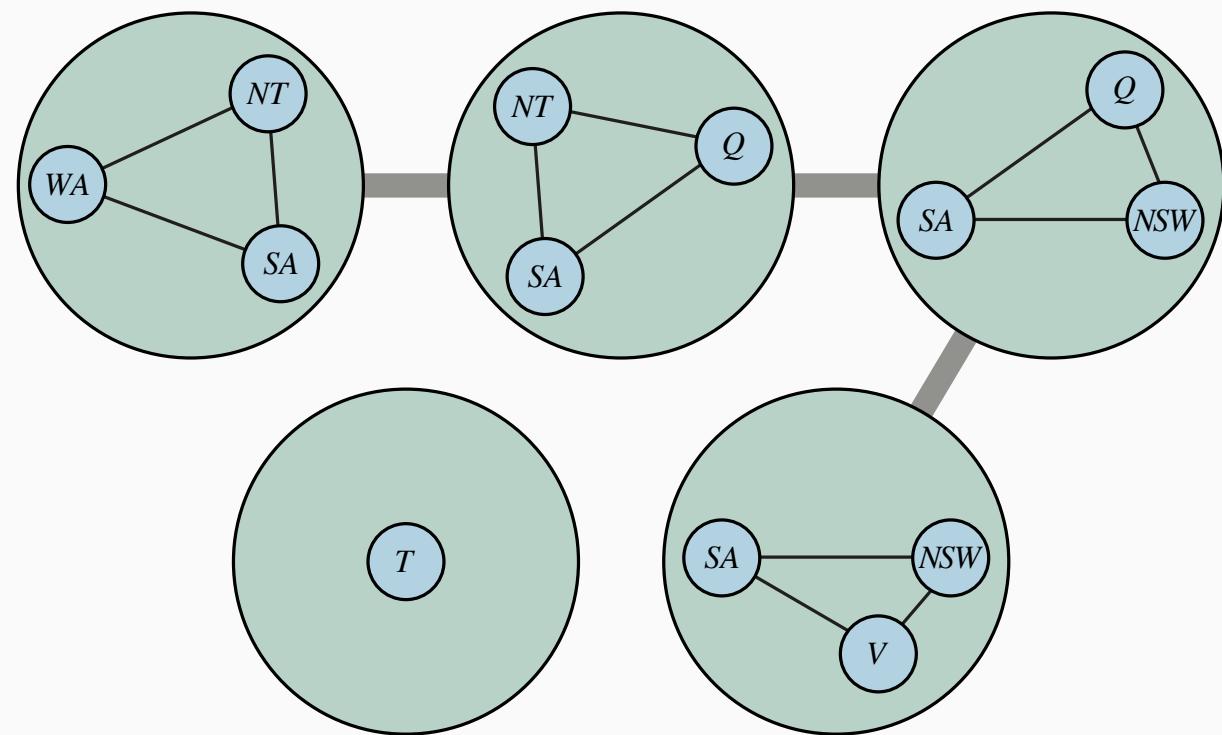
For each possible assignment to the **cycle cutset**:

- remove values from the remaining domains that would violate a constraint with the **cycle cutset**
- if the remaining tree-structured CSP has a solution, return the solution together with the assignment to the **cycle cutset**

Tree decomposition



The original graph



Decomposition of the graph

Algorithm

- Decompose the original graph into a tree where each node consists of overlapping subproblems that are solved independently
- Solve the tree-structured CSP

TDT4136 Introduction to Artificial Intelligence

Lecture 5: Logical Agents, Propositional Logic

Chapter 7 in the textbook

Keith Downing

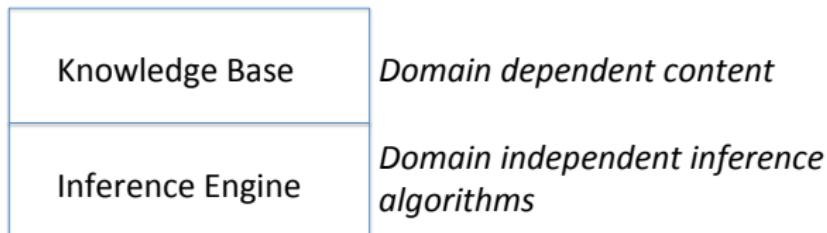
Norwegian University of Science and Technology

2024

Outline

Knowledge-based agents

- **Knowledge base (KB)**: A set of sentences that describe facts about the world in some formal (representational) language
- **Inference engine**: A set of procedures that use the representational language to infer new facts from known ones as well as to answer a variety of KB queries.



Operation on the Knowledge Base

Two important operations on the KB:

- add new knowledge to KB
- ask questions about the knowledge in the KB. Questions are "asked" /triggered in two ways
 - a direct question from the user that doesn't require reasoning but just **retrieval** from the KB
 - a question representing a lack of knowledge required to solve a problem. This knowledge is implicit in the KB and need to be **inferred**.

Inference and Retrieval

- What happens in your mind when you are asked the following question?
 - Question: Is Stockholm in Sweden?
- What happens in your mind when you are asked the following question?
 - Question: Does a vegan person eat snail?
- Different?

A simple knowledge-based agent

- Knowledge base = set of **sentences** in a **formal** language
- Declarative approach to building an agent:
 - **Tell** it what it needs to know
 - Then it can **Ask** itself what to do—answers should "follow from" the KB

```
function KB-AGENT(percept) returns an action
```

static: *KB*, a knowledge base

t, a counter, initially 0, indicating time

```
    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
```

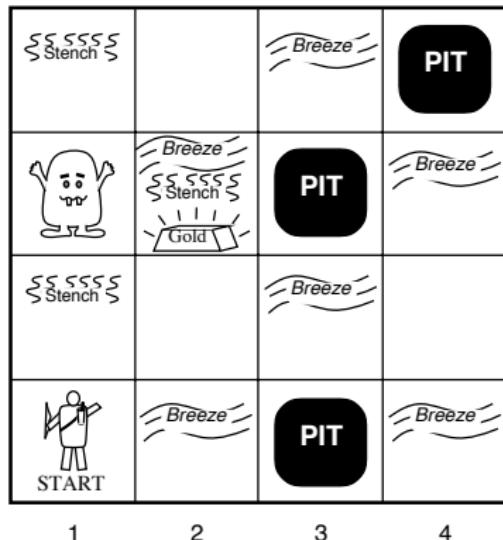
```
    action  $\leftarrow$  ASK(KB, MAKE-ACTION-QUERY(t))
```

```
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
```

```
    t  $\leftarrow$  t + 1
```

```
return action
```

An "informal" look into Wumpus World



Environment

Squares adjacent to wumpus are smelly

Squares adjacent to pit are breezy

One room contains a gold

Glitter iff gold is in the same square

Shooting kills wumpus (screams) if agent is facing it

When an agent walks into a wall, it perceives a bump.

The goal of the agent is to grab the gold and bring it to square [1,1]

Wumpus World PEAS description

Performance measure

+100 points for walk out w/gold

-100 points for dying

-1 point for each action

-10 points for using arrow

Actions and percepts

Actions: Left turn by 90°, Right turn by 90°, Forward, Grab, Climb, Shoot

Percept: [Stench?, Breeze?, Glitter?, Bump?, Scream?]

First movement

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1	A OK	2,1 OK	3,1 4,1

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

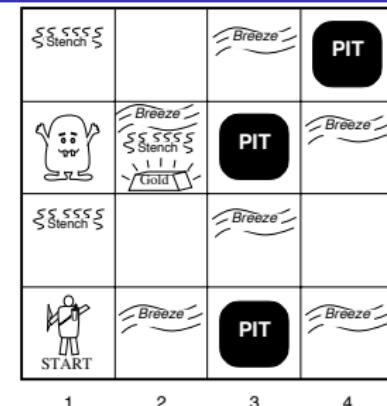
1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2 P?	3,2	4,2
OK			
1,1	V OK	2,1 A B OK	3,1 P? 4,1

$[\neg \text{Stench}, \neg \text{Breeze}, \neg \text{Glitter}, \neg \text{Bump}, \neg \text{Scream}]$ $[\neg \text{Stench}, \text{Breeze}, \neg \text{Glitter}, \neg \text{Bump}, \neg \text{Scream}]$

- Knowledge Base: Rules of game/environment
- Location (always starts at): [1,1]
- Percept: $[\neg \text{Stench}, \neg \text{Breeze}, \neg \text{Glitter}, \neg \text{Bump}, \neg \text{Scream}]$
- Action: Move forward to cell [2,1]. Outcome: Location [2,1]
- New Percept: $[\neg \text{Stench}, \text{Breeze}, \neg \text{Glitter}, \neg \text{Bump}, \neg \text{Scream}]$
- Infer: There must be a pit in [2,2] or [3,1]
- Knowledge Base: Rules of game/environment + inference, i.e., There must be pit in [2,2] or [3,1]
- Action: Return to [1,1] to try next safe cell

Next move

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1



- Knowledge Base: Rules of game/environment + inference, i.e., There must be pit in [2,2] or [3,1]
- Action: Move to cell [1,2]. Outcome: Location [1,2]
- Percept: [Stench, \neg Breeze, \neg Glitter, \neg Bump, \neg Scream]
- Infer: No pit in [2,2]
- Knowledge Base: Rules of game/environment + inference, i.e., There is pit in [3,1]

Wumpus World PEAS description

Environment characteristics

Deterministic?: Yes. Outcomes are exactly specified

Static?: Yes. Wumpus and pits do not move

Discrete?: Yes

Single agent?: Yes - assuming wumpus as a natural phenomena

Fully observable?: No, only local perception

Episodic?: No, previous actions affect the current and next actions.

Remembering what was observed (e.g., pit) is very useful

How to tell a machine to play the Wumpus game?

How to represent the knowledge into the agent ?

How does the agent do the inference based on built knowledge ?

We need a knowledge representation language.

The objective of knowledge representation is to express the knowledge about the world in a computer-tractable form

Various Knowledge Representation Languages

Fundamental concepts in logic

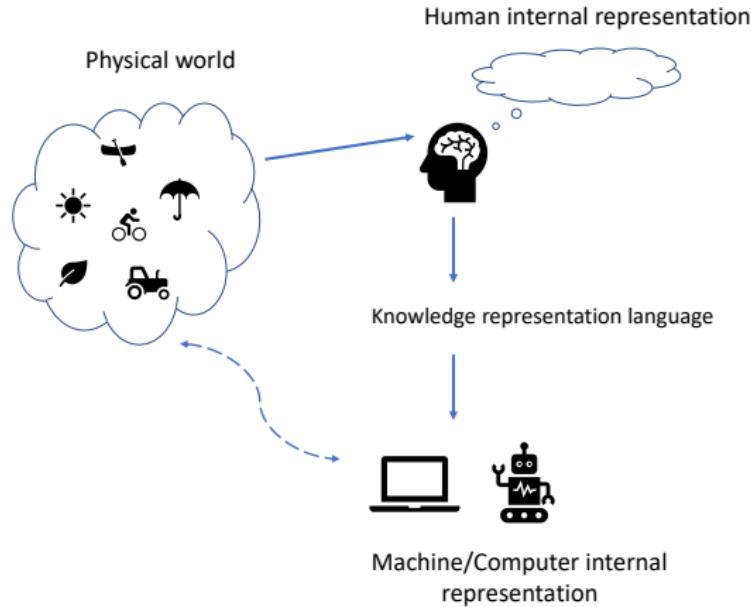
- Syntax and semantics
- Possible worlds and Model
 - an assignment of a symbol is either true or false in boolean systems
 - a *possible world* is an assignment of all the Symbols in the "system"
 - a possible world for each assignment, hence a set of possible worlds representing all possible assignment combinations
 - a *model* (of sentence S): a symbol assignment(i.e, a possible world) that makes S true
 - a *model (of KB)* is a possible world where the KB(i.e., each sentence in it) is true
- *Entailment.* $\langle \text{Sentence1} \rangle \models \langle \text{Sentence2} \rangle$
 - A *entails* B
 - B *follows from* A
 - B is true whenever A is true

Computers and logic

- Computers can use logic in order to, e.g., prove mathematical theorems and to diagnose failures
- But, first what is logic and how it works.

Human - layman	Human- computer scientist	Computer
Natural language  Sissel is a student. Sophie is a teacher. Sissel plays piano Sophie plays piano if she is happy	Knowledge Representation Language  student(Sissel) teacher (Sophie) playsPiano(Sissel) happy(Sophie) → playsPiano(Sophie)	Programming language (e.g., Prolog)  student(Sissel). teacher (Sophie). playsPiano(Sissel). playsPiano(Sophie):- happy(Sophie).
Does Sissel play piano? Does Sophie play piano?	playPiano(Sissel) playPiano(Sophie) (answer these, i.e., find if these are True or False)	?- playPiano(Sissel). ?- playPiano(Sophie).

Physical World and Internal Representations



What is Logic

- One of the oldest disciplines in history
- Dates back to Aristoteles



"EVERYONE'S USING YOUR THEOREM, PYTHAGORAS.
I TOLD YOU YOU SHOULD HAVE PATENTED IT."

If there had been Facebook®
in Ancient Greece, we would
never have had philosophy.

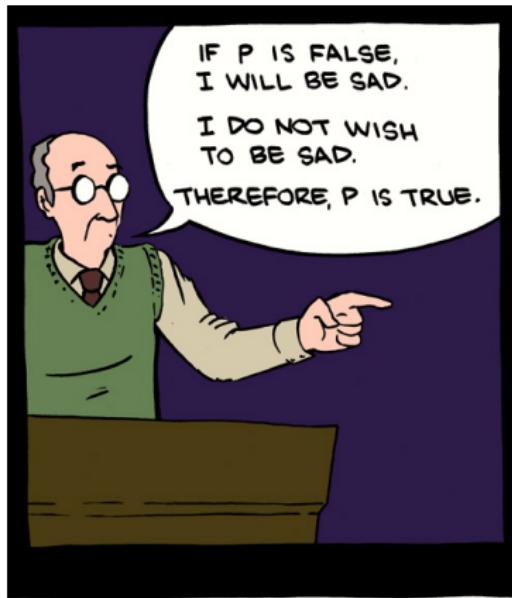


People use logic all the time

- People use logic in order to talk about observations, to define concepts, and to formalize theories

People use logic all the time -cont.

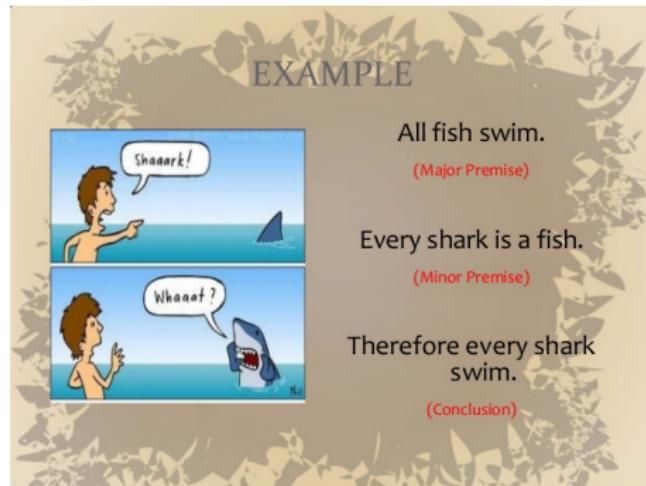
- People use logic to prove something



There. Now you can skip 99% of philosophical debates.

People use logic all the time -cont.

- Using logical reasoning we can derive new information from what we already know.

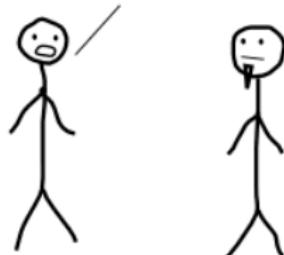


People use logic all the time -cont.

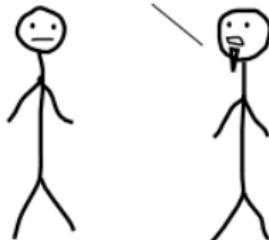
Aristotle is said to be the creator of "modal logic," a wonderful logical tool that allows one to create unsound arguments that most people will go along with.



Give me some random fact and I will show you how to use this amazing tool.



Off the southern coast of South America there is an island called Snow Island. It is named after all the snow on it.



Ok...

- 1) It is possible for islands to be named after that which is most common on them.
- 2) There is an island named "Prince Edward Island."
- 3) Therefore, Prince Edward Augustus (father of Queen Victoria) is the most common entity on Prince Edward Island.



ThadGuy.com

People use logic all the time -cont.

- We use logical proofs to convince others of our conclusions..



"BOY IS HE GOOD ! HE'S EVEN GOT ME
CONVINCED YOU'RE GUILTY !"

People use logic all the time -cont.



Propositional Logic

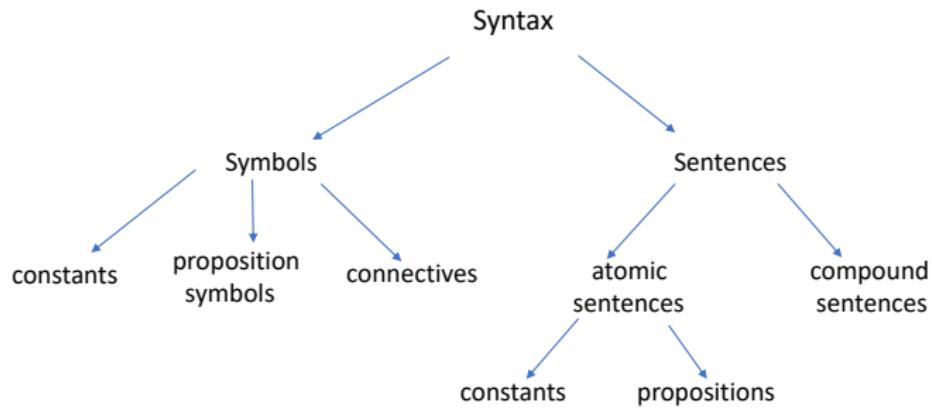
In this lecture we deal with **Propositional Logic**.

Syntax of Propositional Logic

Syntax - symbols, sentences

- Symbols (alphabet) consists of:
 - Constants: True, False
 - Proposition symbols : P, Q, . . .
 - Connectives: \neg , \wedge , \vee , \rightarrow , \Leftrightarrow
- Sentences can be atomic (constants and propositions) or compound sentences

Syntax



Atomic Sentences

- **Proposition:** a declarative statement about the world that is either true or false
- Which of the followings are propositions:
 - Norway is in Europe.
 - Stockholm is capital city of Norway.
 - What is your name?
 - Do your homework.
 - This sentence is false.

Which of the followings are propositions?

- Norway is in Europe (true).
- What is your name? (not declarative)
- Do your homework (not declarative)
- This sentence is false (neither true nor false)

Compound Sentences

- Compound sentences: constructed from atomic and/or other compound sentences via connectives:
 - If S is a sentence, $\neg S$ is a sentence (negation)
 - If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)
 - If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)
 - If S_1 and S_2 are sentences, $S_1 \implies S_2$ is a sentence (implication)
 - If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Example - Compound Sentences

Assume the following propositions:

P : It is sunny this afternoon

Q : it is colder than yesterday

Lg : the traffic light is green

Cg: the cars will go

How are the following compound sentences be represented in terms of the proposition symbols above?

- ① It is not sunny this afternoon and it is colder than yesterday.
- ② If the traffic light is green then the cars will go.
- ③ The cars will go only if traffic light is green

Example - Compound Sentences, cont.

How are the following compound sentences be represented in terms of the proposition symbols above, p, q, lg, cg?

- ① It is not sunny this afternoon and it is colder than yesterday.
- ② If the traffic light is green then the cars will go.
- ③ The cars will go only if traffic light is green

Representation of these sentences in propositional logic:

- ① $\neg P \wedge Q$
- ② $Lg \implies Cg$
- ③ $Cg \implies Lg$

Semantics

Semantics of **atomic** sentences are determined according to their truth values wrt interpretations.

An **interpretation** maps symbols to one of the two values:
True (T), or False (F), depending on whether the symbol is **satisfied** in the "world".

- P: Light in the room is on (*True in Interpretation I*) then $\text{Value}(P, I) = \text{True}$,
- Q: It rains outside (False) then $\text{Value}(Q, I) = \text{False}$
- If P: Light in the room is on (False in I') then $\text{Value}(P, I') = \text{False}$

Semantics of Connectives

Semantics of **compositional** sentences are determined using the standard rules of logic for connectives:

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>
<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>
<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>

Example - Compound Sentences with implication

Assume the following propositions:

Lg : the traffic light is green

Cg : the cars will go

Draw the truth table for implication and check if the logic representation of the following sentences are correct.

- ① If the traffic light is green then the cars will go: $Lg \implies Cg$
- ② The cars will go only if traffic light is green: $Cg \implies Lg$

Entailment

- Intuitively, when we read in the newspaper that “RBK’ and “Brann” won’, we can immediately say “RBK won”
- **Note that \models is NOT a part of the logical knowledge representation(KR) language, it is not a connective in any logic KR language)**
- \models belongs to a/the metalanguage that is used a level above the knowledge representation
- Entailment means that the truth of one sentence (α) follows from the truth of another (e.g., set of all sentences in KB).

Semantics of Inferring new information

Inference may be needed in order to answer a question, based on what the agent knows (i.e, the sentences in the KB).

Logically, this means whether KB "**entails**" the sentence (e.g. α) of which truth is asked in the question, i.e., $\text{KB} \models \alpha$?

In other saying, the logical agent can use the sentences in the Knowledge Base to draw conclusions that are *logically entailed* by those sentences.

How to design the reasoning procedure(s) that answers $\text{KB} \models \alpha$?

Entailment

Entailment may be obtained in various ways:

- Model checking/enumeration
- Resolution and Proof by Contradiction
- Forward and Backward chaining

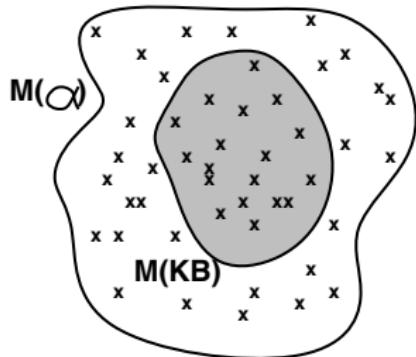
Model checking

Entailment through checking models:

Necessary for entailment of α : α is true in every model that KB is true.

$\text{KB} \models \alpha$ iff $M(\text{KB}) \subseteq M(\alpha)$.

We say m is a model of a sentence α if α is true in m . $M(\alpha)$ is the set of all models of α



Example: Wumpus gridworld

Propositional representation of Wumpus gridworld.

- Rules of Wumpus game and representation of a game.
- KB = wumpus-world rules + observations
- Let $P_{i,j}$ be true if there is a pit in $[i,j]$.
- Let $B_{i,j}$ be true if there is a breeze in $[i,j]$.
- “Pits cause breezes in adjacent squares” = “A square is breezy **if and only if** there is an adjacent pit”

Wumpus world after the first 2 moves

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1 A	2,1	3,1	4,1
OK	OK		

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

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

- Situation after detecting nothing in [1,1] and moving right, breeze in [2,1]:

$$R_1 : \neg P_{1,1}$$

$$R_2 : B_{1,1} \leftrightarrow (P_{1,2} \vee P_{2,1})$$

$$R_3 : B_{2,1} \leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

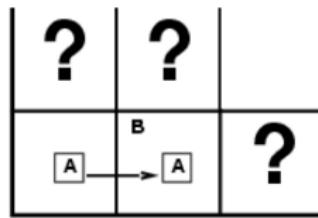
$$R_4 : \neg B_{1,1}$$

$$R_5 : B_{2,1}$$

- $\text{KB} \equiv R1 \wedge R2 \wedge R3 \wedge R4 \wedge R5$

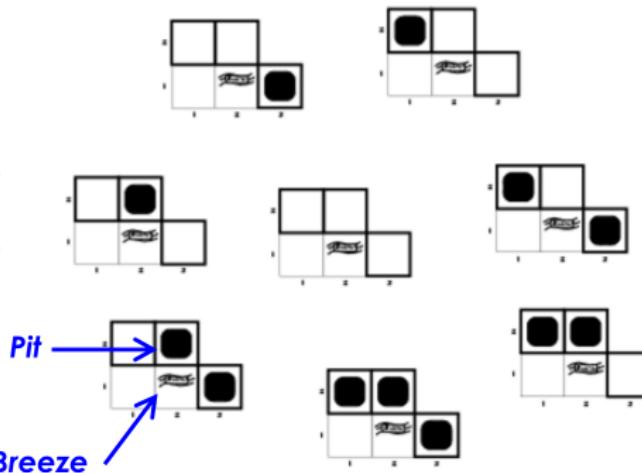
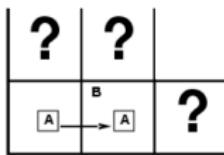
Models in reduced Wumpus gridworld

Consider possible models for KB assuming only pits and a **reduced** Wumpus world with only 5 squares and pits:



Wumpus Models

The reduced Wumpus World. All 8 possible worlds/models are:



Decision in Wumpus world by Model Checking

Still reduced Wumpus gridworld example.

- Goal: Decide whether KB says "no pit in [1,2]"
- Let $\alpha = \neg P_{1,2}$
- Does $\text{KB} \models \neg P_{1,2}$?

Models within the red line are consistent with our KB:

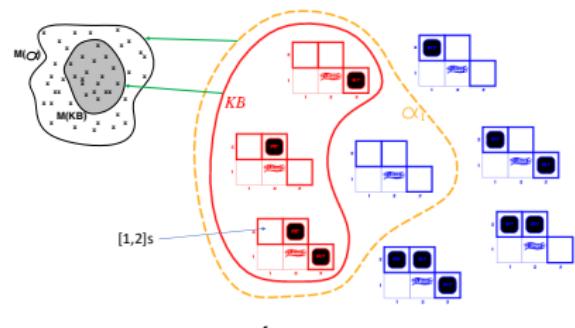
$$R_1 : \neg P_{1,1}$$

$$R_2 : B_{1,1} \leftrightarrow (P_{1,2} \vee P_{2,1})$$

$$R_3 : B_{2,1} \leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

$$R_4 : \neg B_{1,1}$$

$$R_5 : B_{2,1}$$



By model checking: $\text{KB} \models \alpha$
Hence [1,2] is safe!

Model Checking through Truth Tables

Truth Table is a simple method for model enumeration and checking.

KB: $A \wedge B \rightarrow C$, $A \wedge B$
 α : C

Does KB $\models \alpha$, i.e.,
 $[(A \wedge B \rightarrow C) \wedge (A \wedge B)] \models C$??

World	A	B	C	$A \wedge B$	$A \wedge B \rightarrow C$
0	0	0	0	0	1
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	1	0	1
4	1	0	0	0	1
5	1	0	1	0	1
6	1	1	0	1	0
7	1	1	1	1	1

- $M[(A \wedge B \rightarrow C) \wedge (A \wedge B)] = \{7\} \subseteq \{1, 3, 5, 7\} = M(C)$.
- Yes

Example Model enumeration in Wumpus world

- After visiting (1,1) and (2,1)

$$R_1 : \neg P_{1,1}$$

$$R_2 : B_{1,1} \leftrightarrow (P_{1,2} \vee P_{2,1})$$

$$R_3 : B_{2,1} \leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

$$R_4 : \neg B_{1,1}$$

$$R_5 : B_{2,1}$$

- KB = $R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5$
- In this Wumpus world: 7 symbols (in KB). We can get $2^7 = 128$ models

Example Model Enumeration in Wumpus world- cont.

if KB is true in a row, check whether α is too.

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	true	true	true	true	false	false						
false	false	false	false	false	false	true	true	true	false	true	false	false
:	:	:	:	:	:	:	:	:	:	:	:	:
false	true	false	false	false	false	false	true	true	false	true	true	false
false	true	false	false	false	false	true	true	true	true	true	true	true
false	true	false	false	false	false	true	true	true	true	true	true	true
false	true	false	false	true	false	false	true	false	false	true	true	false
:	:	:	:	:	:	:	:	:	:	:	:	:
true	false	true	true	false	true	false						

Answer: $KB \models \neg P_{1,2}$. No pit in $P_{1,2}$.

Truth-table enumeration algorithm

```
function TT-ENTAILS?( $KB, \alpha$ ) returns true or false
  inputs:  $KB$ , the knowledge base, a sentence in propositional logic
            $\alpha$ , the query, a sentence in propositional logic
   $symbols \leftarrow$  a list of the proposition symbols in  $KB$  and  $\alpha$ 
  return TT-CHECK-ALL( $KB, \alpha, symbols, \{ \}$ )

function TT-CHECK-ALL( $KB, \alpha, symbols, model$ ) returns true or false
  if EMPTY?( $symbols$ ) then
    if PL-TRUE?( $KB, model$ ) then return PL-TRUE?( $\alpha, model$ )
    else return true // when KB is false, always return true
  else
     $P \leftarrow$  FIRST( $symbols$ )
     $rest \leftarrow$  REST( $symbols$ )
    return (TT-CHECK-ALL( $KB, \alpha, rest, model \cup \{P = true\}$ )
          and
          TT-CHECK-ALL( $KB, \alpha, rest, model \cup \{P = false\}$ ))
```

Inference by Model Enumeration

- Enumeration is sound and complete
- The truth table is exponential in the number of propositional symbols (we checked all rows/assignments)
- Model checking complexity:
If KB and α contain n symbols:
 - Time complexity: $O(2^n)$
 - Space complexity: $O(n)$
- **We need effective/smarter ways of doing inference**

Theorem Proving

- To build a proof of the desired/goal/question sentence without dealing with model enumeration/truth table
- A proof is a chain of application of **inference rules** - and logical equivalences.
- We'll see what these inference rules are - soon.

First, some concepts related to Entailment

- Logical Equivalence. Two sentences α and β are logically equivalent if $\alpha \models \beta$ and $\beta \models \alpha$
- Validity. A sentence is valid if it is true in all models.
 - How is validity relevant to Entailment?
 - Answer: This relates to the relationship between entailment and implication.
 - Decision of whether $\alpha \models \beta$: answer is "yes" iff the sentence $\alpha \implies \beta$ is valid - true in all models. This is called **Deduction Theorem**
- Satisfiability is about a specific relationship between a sentence and a(some) model.
 - a model *satisfies* a sentence if the sentence is true for this model
- How is satisfiability relevant to entailment?
- Decision of whether $\alpha \models \beta$: answer is "yes" iff the sentence $\alpha \wedge \neg\beta$ is unsatisfiable - not true in any model.
- This is the basis of an inference procedure called **proof by refutation (or contradiction)**

Some/standard Logical equivalences

Two sentences are **logically equivalent** iff true in same models:

$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \text{ commutativity of } \wedge$$

$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \text{ commutativity of } \vee$$

$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \text{ associativity of } \wedge$$

$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \text{ associativity of } \vee$$

$$\neg(\neg \alpha) \equiv \alpha \text{ double-negation elimination}$$

$$(\alpha \implies \beta) \equiv (\neg \beta \implies \neg \alpha) \text{ contraposition}$$

$$(\alpha \implies \beta) \equiv (\neg \alpha \vee \beta) \text{ implication elimination}$$

$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \implies \beta) \wedge (\beta \implies \alpha)) \text{ biconditional elimination}$$

$$\neg(\alpha \wedge \beta) \equiv (\neg \alpha \vee \neg \beta) \text{ De Morgan}$$

$$\neg(\alpha \vee \beta) \equiv (\neg \alpha \wedge \neg \beta) \text{ De Morgan}$$

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

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

Inference Rules approach

- How to make the process more efficient?
- KB is true on only a smaller subset
- **Solution:** check only entries for which KB is True.
- That is, infer new logical sentences from the knowledge base and see if they match a query
- This is the idea behind the inference rules approach
- *Inference rules* represent sound inference patterns repeated in inferences

Properties of an Inference Procedure and connection to Entailment

- Assume an **inference procedure** i that
 - derives a sentence α from the KB, i.e., $KB \vdash_i \alpha$
- **Soundness:** An inference procedure is sound
If whenever $KB \vdash_i \alpha$, then it is also true that $KB \models \alpha$



- **Completeness:** An inference procedure is complete
If whenever $KB \models \alpha$ then it is also true that $KB \vdash_i \alpha$
- Sound and complete inference procedures are desirable

Inference rules

Syllogism rules

- Modus ponens (method of affirming)

$$\begin{array}{c} A \rightarrow B \\ A \\ \hline B \end{array}$$

- Modus Tollens (method of denying)

$$\begin{array}{c} A \rightarrow B \\ \neg B \\ \hline \neg A \end{array}$$

- Hypothetical Syllogism

$$\begin{array}{c} A \rightarrow B \\ B \rightarrow C \\ \hline A \rightarrow C \end{array}$$

- Disjunctive syllogism (Unit resolution)

$$\begin{array}{c} A \vee B \\ \neg A \\ \hline B \end{array}$$

Some other inference rules

- And-elimination

$$\frac{A_1 \wedge A_2 \wedge \dots \wedge A_n}{A_i}$$

- And-introduction

$$\frac{A_1, A_2, \dots, A_n}{A_1 \wedge A_2 \wedge \dots \wedge A_n}$$

- Or-introduction

$$\frac{A_i}{A_1 \vee A_2 \vee \dots \vee A_n}$$

- Implication Creation

$$\frac{A}{B \implies A}$$

- Implication Distribution

$$\frac{A \implies (B \implies C)}{(A \implies B) \implies (A \implies C)}$$

Inference rules approach

- Inference rule approach: Apply an inference rule that matches with the knowledge in KB. Do this until satisfying the query, e.g., P1.
- Starting with a KB.
 - ASK(P1): is P1 true given what is in KB?
- Derive P1 from the KB
 1. Use inference rules to add new statements
 2. Use **logical equivalence** to rewrite existing statements

Inference rules approach - Example.

KB: $P \Rightarrow Q, Q \Rightarrow R$

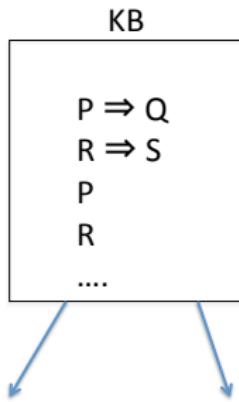
Question: Does $\text{KB} \models (P \Rightarrow R)$?

Using Inference rules:

- ① $P \Rightarrow Q$ (Premise)
- ② $Q \Rightarrow R$ (Premise)
- ③ $P \Rightarrow (Q \Rightarrow R)$ (Implication Creation: 2)
- ④ $(P \Rightarrow Q) \Rightarrow (P \Rightarrow R)$ (Implication Distribution: 3)
- ⑤ $P \Rightarrow R$ (Modus Ponens: 4, 1)

Problems with Inference rules approach

- There may be more than one rule that can apply at a certain stage.



- One solution: **Resolution** is a single inference rule that yields a complete inference algorithm

Unsatisfiability and proof by Inference rule Resolution

- A sentence is **satisfiable** if it is true in **some** models
- A sentence is **unsatisfiable** if it is true in **no** models
 - e.g., $A \wedge \neg A$
- **Unsatisfiability** is connected to inference via the following:
 $KB \models \alpha$ if and only if $(KB \wedge \neg \alpha)$ is unsatisfiable
 - i.e., **proof** α by contradiction

Resolution rule and "Proof by Contradiction"

- Instead of showing $\text{KB} \models \alpha$, we show that $\text{KB} \wedge \neg\alpha$ is not satisfiable.
- Disproving $\text{KB} \wedge \neg\alpha$
proves the entailment $\text{KB} \models \alpha$
- The inference rule called *Resolution Rule* is used for this purpose

Resolution Rule

- Example:

$$\frac{(A \vee B) , (\neg A)}{B}$$

- Resolution inference rule

$$\frac{l_1 \vee \dots \vee l_{i-1} \vee l_i \vee l_{i+1} \vee \dots \vee l_k , m_1 \vee \dots \vee m_{j-1} \vee m_j \vee m_{j+1} \vee \dots \vee m_n}{l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

where l_i and m_j are complementary literals (e.g., $l_i = \neg m_j$)

Resolution algorithm

Proof by contradiction, i.e., show $KB \wedge \neg\alpha$ unsatisfiable

```
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  $\leftarrow \{ \}$ 
  while true 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
```

Conjunctive Normal Form

However to apply resolution technique its requires to represent KB as well as any sentence α that we wish to derive in a special format known as **Conjunctive Normal Form** (CNF): conjunction of clauses.

- A **clause** is an expression of the form $I_1 \vee I_2 \vee \dots \vee I_k$ where each I_i is a literal - a disjunction of literals.
- Example CNF: $(A \vee B) \wedge (\neg A \vee \neg C \vee D)$ - conjunction of disjunctions
- A **literal** is either a propositional symbol or the negation of a symbol.
- Every propositional sentence is equivalent to a conjunction of clauses.

Conversion to CNF

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

1. Eliminate \Leftrightarrow , replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$$

2. Eliminate \Rightarrow , replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$$

3. Move \neg inwards using de Morgan's rule:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$$

4. Apply distributivity law (\vee over \wedge) and flatten:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$

Resolution procedure for Wumpus problem

Our knowledge base: $R2 \wedge R4$
 $(B_{1,1} \leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$

Now, we want to verify that there is no pit in [1,2]. $\alpha = \neg P_{1,2}$

$KB \models \alpha ?$

For this, we need to show that $KB \wedge \neg \alpha$ is unsatisfiable.

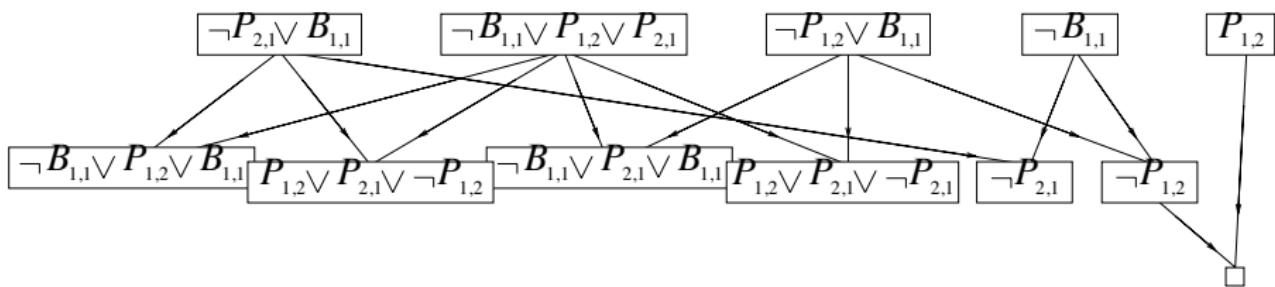
4	 Stench		Breeze
3	 Breeze	 Stench	PIT
2	 Stench		Breeze
1	 START	Breeze	PIT
	1	2	3
			4

RR process on Wumpus example

We have converted the KB into CNF:

$$(\neg P_{2,1} \vee B_{1,1}) \wedge (\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee \neg B_{1,1}) \wedge (\neg B_{1,1})$$

$\alpha = \neg P_{1,2}$, we take also its negation.



Problem with Resolution Refutation

- Resolution is complete but can be exponential in space and time.
- If we can reduce all clauses to a special forms called **Horn Clauses**, deciding entailment becomes linear in the size of the knowledge base (KB)
- Inference with Horn clauses can be done through the **forward chaining** and **backward chaining** algorithms

Horn Clauses

- Definite clause: Disjunction of literals of which exactly one is positive; the rest are negative
- Horn Clause: Disjunction of literals of which at most one is positive

$$\neg A \vee \neg B \vee \neg C \vee D$$

$$\equiv [\neg A \vee \neg B \vee \neg C] \vee D \text{ (Associativity)}$$

$$\equiv \neg [A \wedge B \wedge C] \vee D \text{ (De Morgan's Law)}$$

$$\equiv [A \wedge B \wedge C] \rightarrow D \text{ (Implication Introduction)}$$

Premise \rightarrow Consequent

Modus Ponens in Forward and Backward Chaining

- Modus ponens is perfect for **Definite Clause** KBs.
- Forward and Backward algorithms rely on Modus Ponens:

$$\frac{\alpha_1, \dots, \alpha_n, \quad \alpha_1 \wedge \dots \wedge \alpha_n \implies \beta}{\beta}$$

Forward and backward chaining

- Forward chaining (data driven)

Idea: Whenever the premises of a rule are satisfied, infer the conclusion.

- Backward chaining (goal driven)

Idea: To prove the fact that appears in the conclusion of a rule prove the premises of the rule.

Both procedures are complete for KBs in the Definite clause form.

Forward chaining

Idea: fire any rule whose premises are satisfied in the *KB*,
add its conclusion to the *KB*, until query is found
Avoid loops.

$P \Rightarrow Q$ (Query)

$L \wedge M \Rightarrow P$

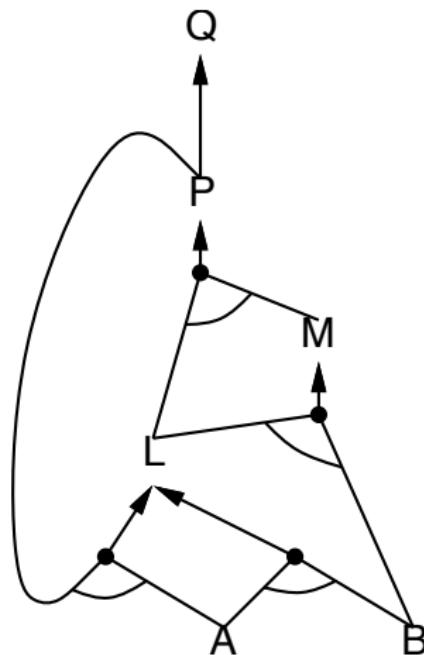
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

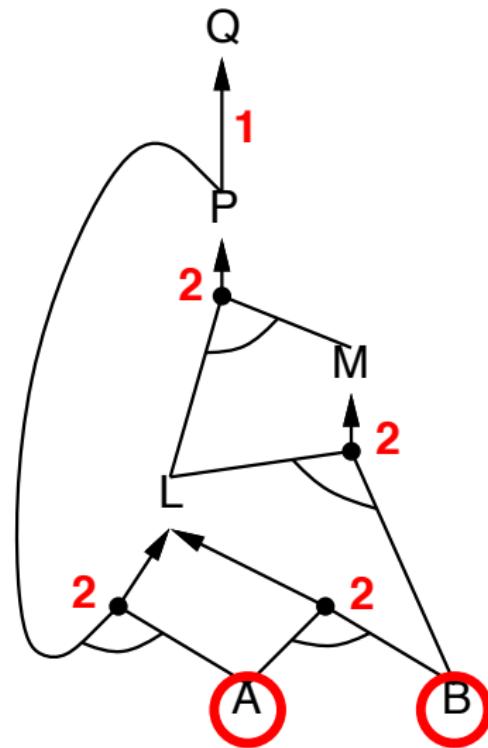


Forward chaining algorithm

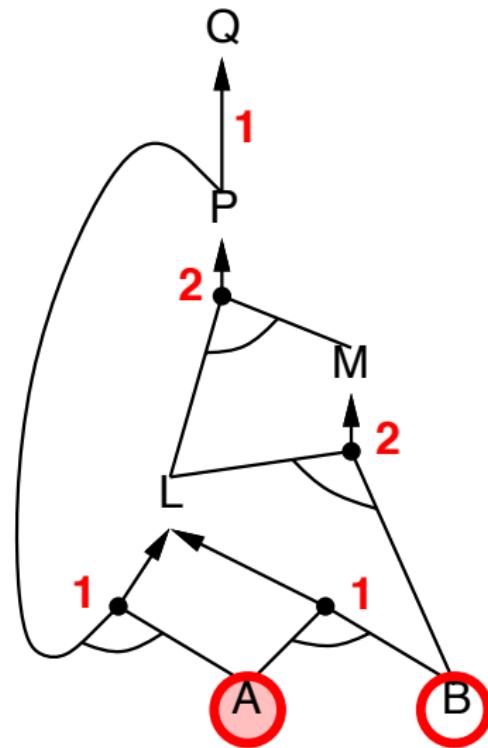
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
           q, the query, a proposition symbol
  count  $\leftarrow$  a table, where count[c] is initially the number of symbols in clause c's premise
  inferred  $\leftarrow$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

  while queue is not empty do
    p  $\leftarrow$  POP(queue)
    if p = q then return true
    if inferred[p] = false then
      inferred[p]  $\leftarrow$  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
```

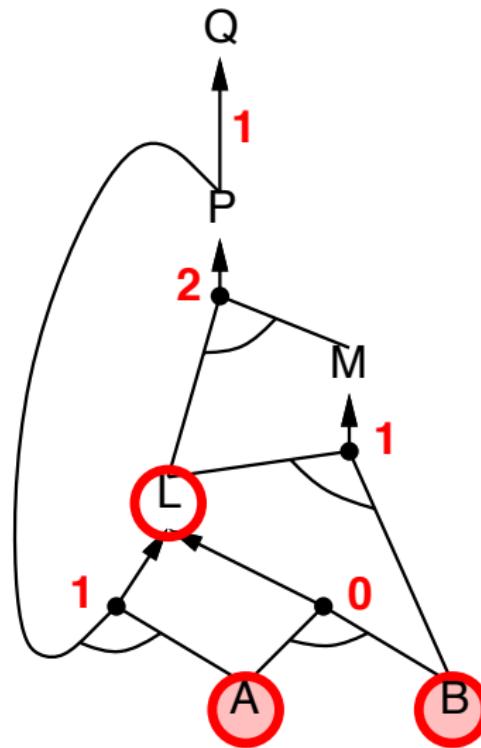
Forward chaining example



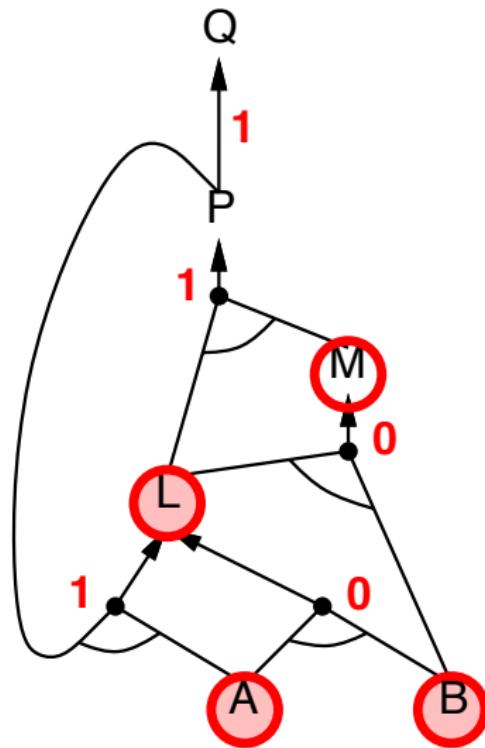
Forward chaining example



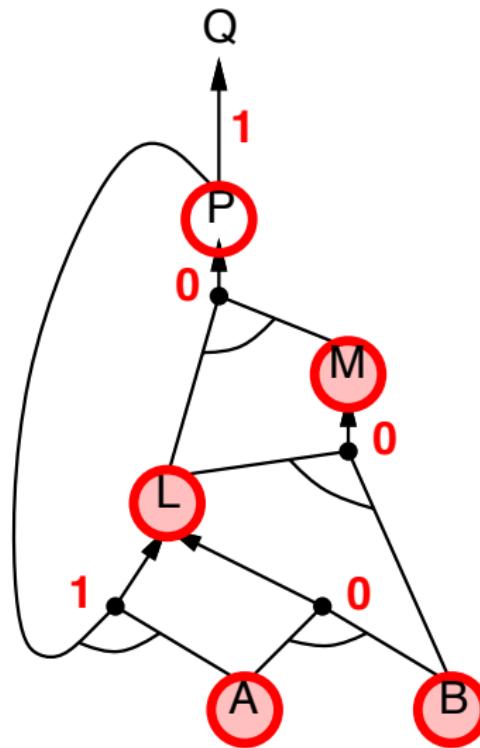
Forward chaining example



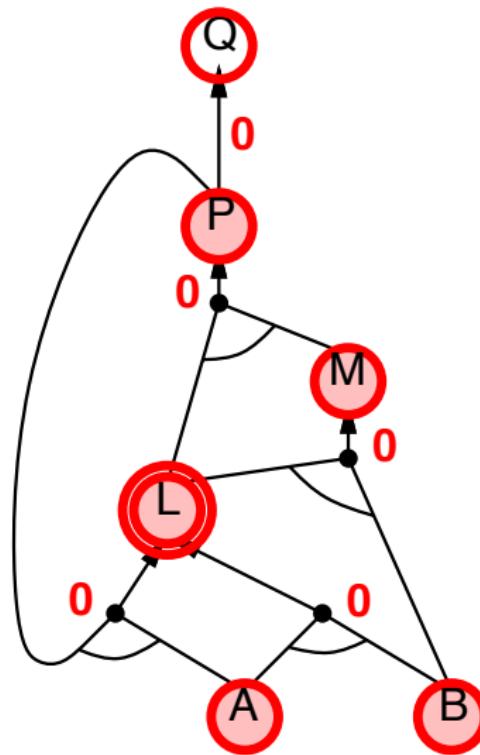
Forward chaining example



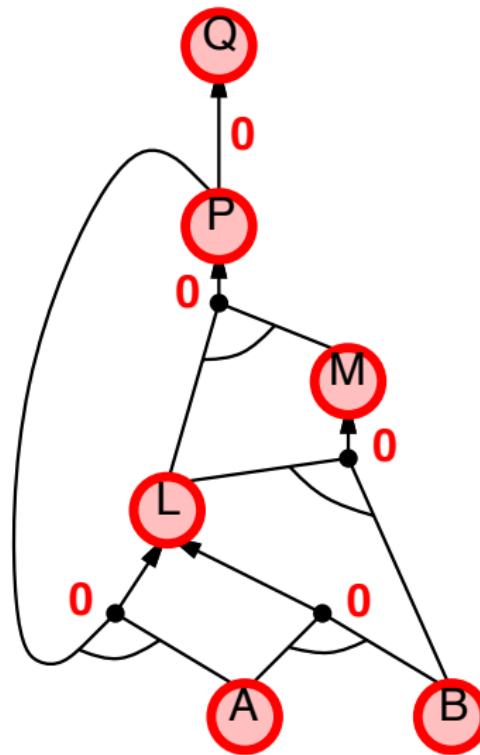
Forward chaining example



Forward chaining example



Forward chaining example



Backward chaining

Idea: work backwards from the query q :

to prove q by BC,

check if q is known already, or

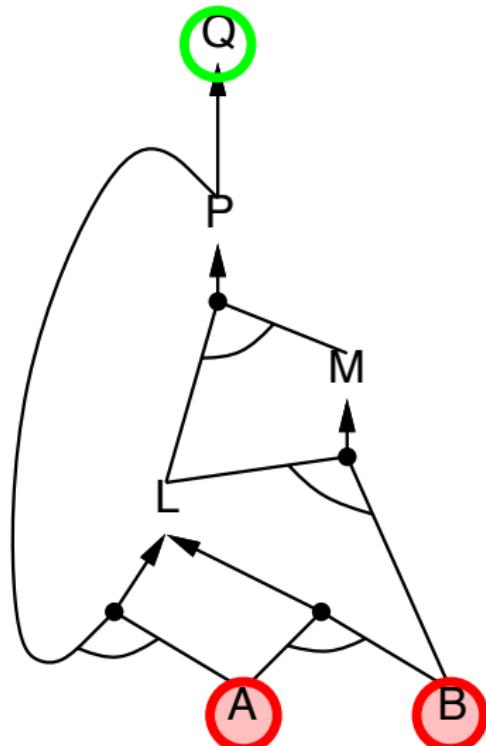
prove by BC all premises of some rule concluding q

Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal

- 1) has already been proved true, or
- 2) has already failed

Backward chaining example



$P \implies Q$ (Query)

$L \wedge M \implies P$

$B \wedge L \implies M$

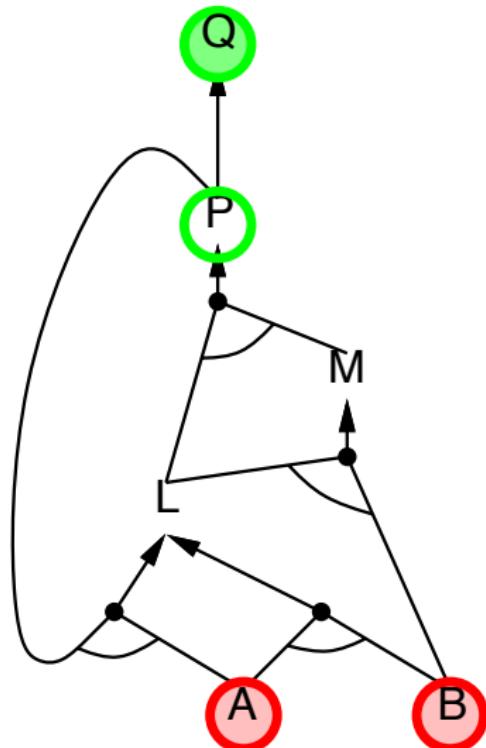
$A \wedge P \implies L$

$A \wedge B \implies L$

A

B

Backward chaining example



$P \implies Q$ (Query)

$L \wedge M \implies P$

$B \wedge L \implies M$

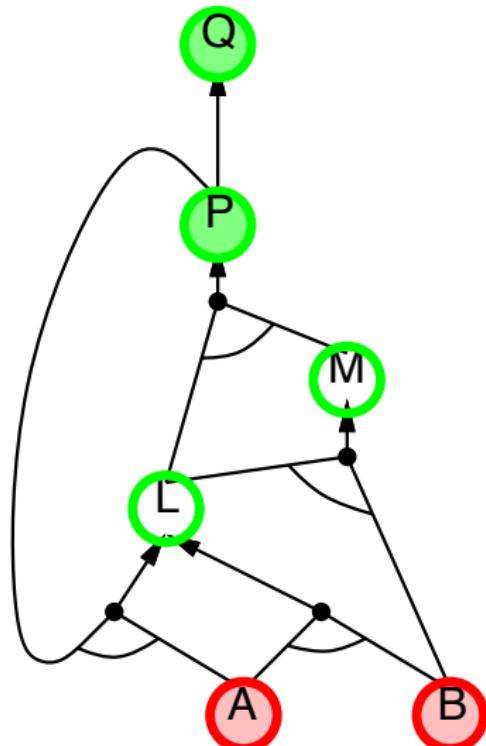
$A \wedge P \implies L$

$A \wedge B \implies L$

A

B

Backward chaining example



$P \implies Q$ (Query)

$L \wedge M \implies P$

$B \wedge L \implies M$

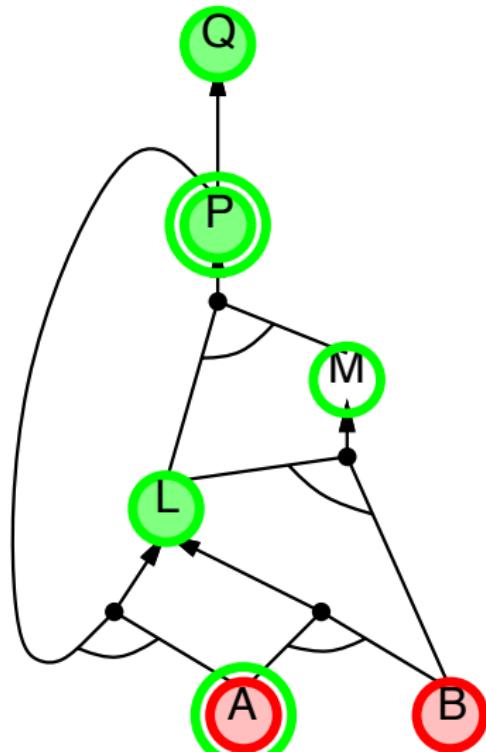
$A \wedge P \implies L$

$A \wedge B \implies L$

A

B

Backward chaining example



$P \implies Q$ (Query)

$L \wedge M \implies P$

$B \wedge L \implies M$

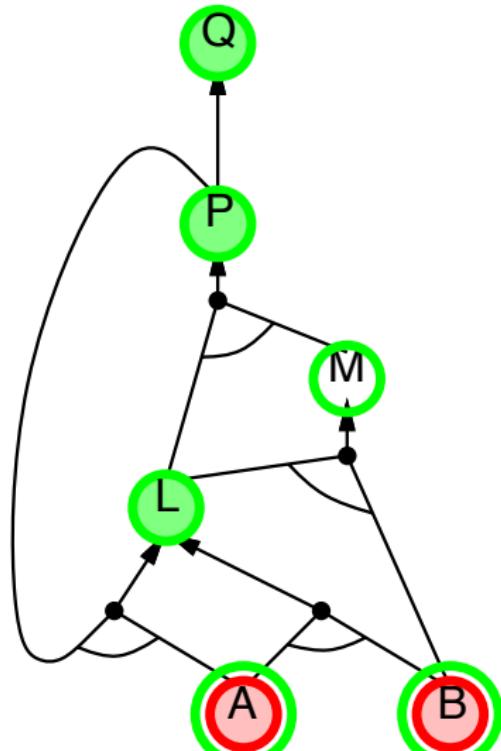
$A \wedge P \implies L$

$A \wedge B \implies L$

A

B

Backward chaining example



$P \implies Q$ (Query)

$L \wedge M \implies P$

$B \wedge L \implies M$

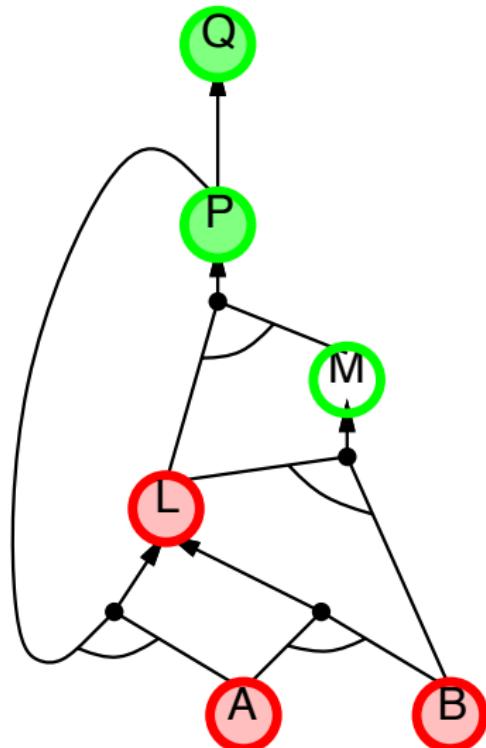
$A \wedge P \implies L$

$A \wedge B \implies L$

A

B

Backward chaining example



$P \implies Q$ (Query)

$L \wedge M \implies P$

$B \wedge L \implies M$

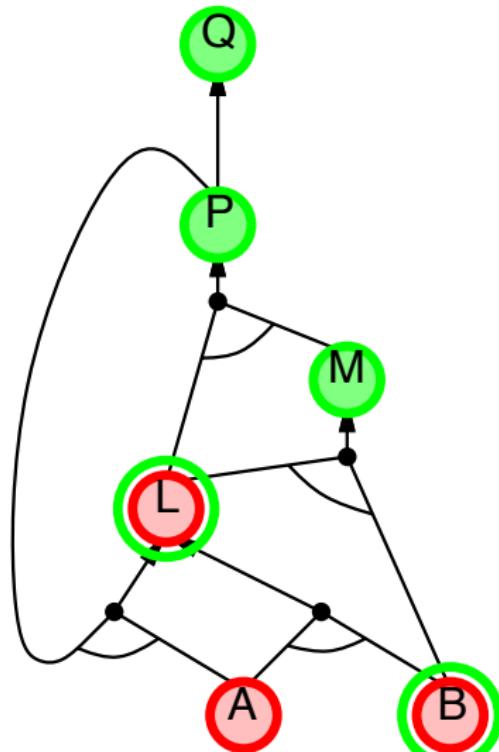
$A \wedge P \implies L$

$A \wedge B \implies L$

A

B

Backward chaining example



$P \implies Q$ (Query)

$L \wedge M \implies P$

$B \wedge L \implies M$

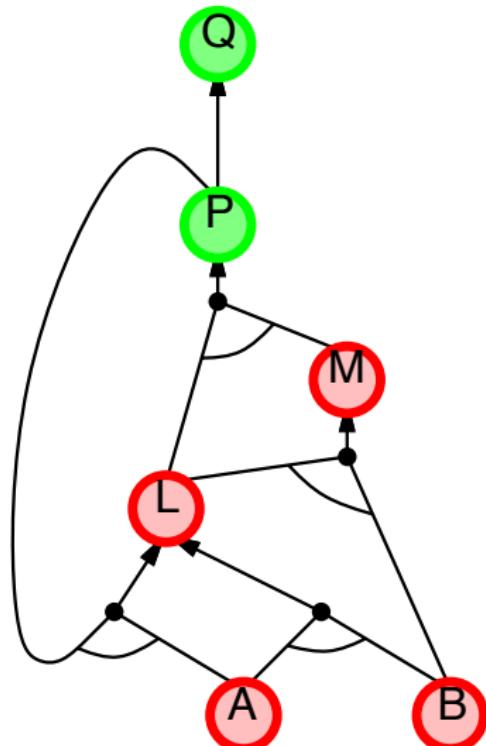
$A \wedge P \implies L$

$A \wedge B \implies L$

A

B

Backward chaining example



$P \implies Q$ (Query)

$L \wedge M \implies P$

$B \wedge L \implies M$

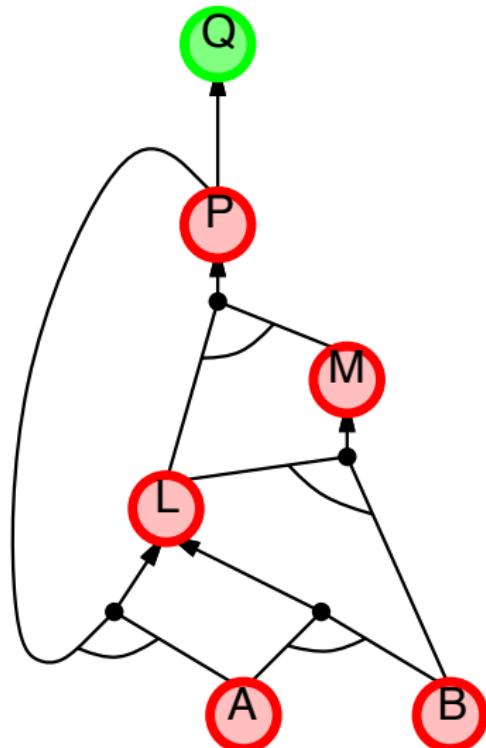
$A \wedge P \implies L$

$A \wedge B \implies L$

A

B

Backward chaining example



$P \implies Q$ (Query)

$L \wedge M \implies P$

$B \wedge L \implies M$

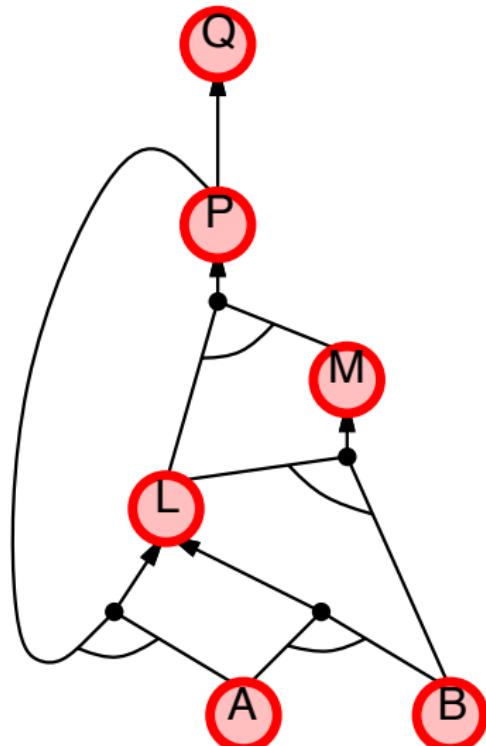
$A \wedge P \implies L$

$A \wedge B \implies L$

A

B

Backward chaining example



$P \implies Q$ (Query)

$L \wedge M \implies P$

$B \wedge L \implies M$

$A \wedge P \implies L$

$A \wedge B \implies L$

A

B

Forward vs.backward chaining

FC is **data-driven**, cf. automatic, unconscious processing,
e.g., object recognition, routine decisions

May do lots of work that is irrelevant to the goal

BC is **goal-driven**, appropriate for problem-solving,
e.g., Where are my keys? How do I get into a PhD program?

Complexity of BC can be **much less** than linear in size of KB

TDT4136 Introduction to Artificial Intelligence

Lecture 6: First Order Logic

Chapter 8 in the textbook

Keith L. Downing

The Norwegian University of Science and Technology
Trondheim, Norway
keithd@idi.ntnu.no

September 25, 2024

Outline

- Recap - Propositional logic
- FOL (Predicate) logic
 - Syntax and semantics
- Next week: Inference in FOL (chapter 9)

- Logic is the scientific study of validity - allows us to test validity related to
 - answering questions
 - making decisions about plans, verifying designs, solving problems in general

- Logic is the scientific study of validity - allows us to test validity related to
 - answering questions
 - making decisions about plans, verifying designs, solving problems in general
- Logic provides a knowledge representation language and an inference mechanism

- Logic is the scientific study of validity - allows us to test validity related to
 - answering questions
 - making decisions about plans, verifying designs, solving problems in general
- Logic provides a knowledge representation language and an inference mechanism
- There are various logical languages with different expressive power

Problem solving in Logic

- Problem description:
 - If it is autumn, then it is lamb-meat season in Norway.
 - If it is lamb-meat season in Norway, then "får i kål" is delicious.
 - "Får i kål" is not delicious.
- Question: Is it autumn?
- How to translate this to Propositional Logic?

Propositional Logic - "Får i kål" example

- Propositions:

Propositional Logic - "Får i kål" example

- Propositions:

P: It is autumn

Q: It is lamb-meat season

R: "Får i kål" is delicious

- Premises?

Propositional Logic - "Får i kål" example

- Propositions:

P: It is autumn

Q: It is lamb-meat season

R: "Får i kål" is delicious

- Premises?

$$P \implies Q$$

$$Q \implies R$$

$$\neg R$$

Propositional Logic - "Får i kål" example

- Propositions:

P: It is autumn

Q: It is lamb-meat season

R: "Får i kål" is delicious

- Premises?

$$P \implies Q$$

$$Q \implies R$$

$$\neg R$$

- How to answer the Question "Is it autumn", i.e., whether the proposition **P** is true or false?

Is it Autumn?

Two main methods for testing/checking if P is true - i.e., if KB entails P :

- by Model Checking with Truth Table
- by Theorem Proving

Model checking with a Truth table - "Får i kål" example

P	Q	R	$P \rightarrow Q$	$Q \rightarrow R$	$\neg R$	P	$\neg P$
T	T	T	T	T	F	T	F
T	T	F	T	F	T	T	F
T	F	T	F	T	F	T	F
T	F	F	F	T	T	T	F
F	T	T	T	T	F	F	T
F	T	F	T	F	T	F	T
F	F	T	T	T	F	F	T
<hr/>							
F	F	F	T	T	T	F	T

Theorem Proving. - "Får i kål" example

Proof: a sequence of sentences, where each is a premise (i.e., given) or is derived from earlier sentences in the proof by an inference rule.

At the end of the sequence of the sentences there will be the target/goal/query if it can be proven true.

Theorem Proving. - "Får i kål" example

Proof: a sequence of sentences, where each is a premise (i.e., given) or is derived from earlier sentences in the proof by an inference rule.

At the end of the sequence of the sentences there will be the target/goal/query if it can be proven true.

To answer P? (Is it autumn?) we'll use the *resolution rule* twice:

Theorem Proving. - "Får i kål" example

Proof: a sequence of sentences, where each is a premise (i.e., given) or is derived from earlier sentences in the proof by an inference rule.

At the end of the sequence of the sentences there will be the target/goal/query if it can be proven true.

To answer P? (Is it autumn?) we'll use the *resolution rule* twice:

Resolution rule:

$$\begin{array}{c} A \vee B \\ \neg A \\ \hline B \end{array}$$

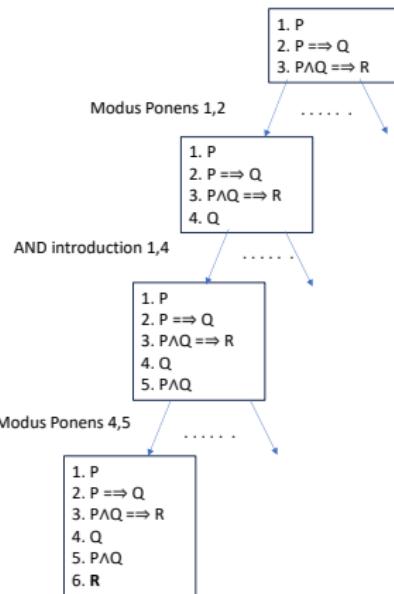
Theorem Proving - "Får i kål" example

- The KB has these sentences (premises), and we ASK the logic system if it is autumn?
 - ① $P \implies Q$ (If it is autumn, then it is lamb-meat season)
 - ② $Q \implies R$. (If it is lamb-meat season then "får i kål" is delicious)
 - ③ $\neg R$ ("Får i kål" is not delicious)
- Proof:
 - ④ **Resolution rule** on 2 and 3: $\neg Q$
 - ⑤ **Resolution rule** on 1 and 4: $\neg P$ - Goal sentence.

Proof as Search

- At any step in the proof process, there may be several inferences that can be applied to the KB.
- We can therefore imagine the proof like a search process.
- Problem definition
 - What is a state?
 - what is an action?
 - what is goal?

Proof as search- example



State: KB

Initial State: $P, P \Rightarrow Q, P \wedge Q \Rightarrow R$.

Actions: Inference rules

Goal: KB including the goal sentence (i.e., R)

More efficient Inference methods

- Searching on the state space (shown as above) is exponential in nr. of propositions
- More efficient methods, using specific forms of sentences
 - Conjunctive Normal Form -CNF
 - Horn Clauses

CNF and Resolution Refutation

- CNF: Conjunction of Clauses (i.e., disjunctive literals)
- Used in Resolution Refutation
 - To prove P , given KB, show that KB and $\neg P$ is unsatisfiable/false/empty clause.
 - Repeatedly use only one inference rule: Resolution rule.

Horn Clauses and Forward/backward Chaining)

- Horn clause: A clause with at most one positive literal
- Used in Forward/Backward chaining
- To prove P , given KB, repeatedly apply Modus Ponens in KB.

Propositional logic

- Less expressive than First order/Predicate logic
- PL assumes that the world contains *facts* that are true or false
- Propositional constants refer to atomic propositions:
 - R: It is raining
 - S: It is snowing
 - W: It is wet
- Compound sentences capture *relationships* among propositions:
$$R \vee S \implies W$$
- Ontological commitment of PL: does not include objects, properties of objects, and relationships between objects.

Limitations of propositional logic

- Assume we want to express *Every student likes vacation:*

John likes vacation \wedge

Mary likes vacation \wedge

Ann likes vacation \wedge

....

Limitations of propositional logic

- Assume we want to express *Every student likes vacation:*
John likes vacation \wedge
Mary likes vacation \wedge
Ann likes vacation \wedge
....
- Problem: KB grows large
- Possible solution: ?

Limitations of propositional logic

- Assume we want to express *Every student likes vacation:*

John likes vacation \wedge

Mary likes vacation \wedge

Ann likes vacation \wedge

....

- Problem: KB grows large
- Possible solution: ?
All students like vacation.

Limitations of propositional logic

- Assume the KB has:
 - Stig is older than Sissel
 - Sissel is older than Paul
 - Stig is older than Sissel \wedge Sissel is older than Paul \implies Stig is older than Paul

Limitations of propositional logic

- Assume the KB has:
 - Stig is older than Sissel
 - Sissel is older than Paul
 - Stig is older than Sissel \wedge Sissel is older than Paul \implies Stig is older than Paul
- We can derive *Stig is older than Paul*

Example on Limitations of propositional logic

- Assume we add *Hanne is older than Sissel* into the KB
- The current KB now:
 - Stig is older than Sissel
 - Sissel is older than Paul
 - $\text{Stig is older than Sissel} \wedge \text{Sissel is older than Paul} \implies \text{Stig is older than Paul}$
 - *Hanne is older than Sissel*

Example on Limitations of propositional logic

- Assume we add *Hanne is older than Sissel* into the KB
- The current KB now:
 - Stig is older than Sissel
 - Sissel is older than Paul
 - $\text{Stig is older than Sissel} \wedge \text{Sissel is older than Paul} \implies \text{Stig is older than Paul}$
 - *Hanne is older than Sissel*
- What else do we need to have in the KB in order to derive *Hanne is older than Paul?*

Example on Limitations of propositional logic

- Assume we add *Hanne is older than Sissel* into the KB
- The current KB now:
 - Stig is older than Sissel
 - Sissel is older than Paul
 - $\text{Stig is older than Sissel} \wedge \text{Sissel is older than Paul} \implies \text{Stig is older than Paul}$
 - *Hanne is older than Sissel*
- What else do we need to have in the KB in order to derive *Hanne is older than Paul?*
- We need:
Hanne is older than Sissel \wedge Sissel is older than Paul \implies Hanne is older than Paul

Limitations of propositional logic -Example 2

- KB grows large
- Possible solution: ?

PersA is older than PersB \wedge PersB is older than PersC \implies
PersA is older than PersC

Limitations of Propositional Logic -example 3: Wumpus

- Consider the statement "If there is breeze in a square, there must be pit in an adjacent square"
- In propositional logic we need 16 sentences (one for each square) to represent this statement (for 4x4 grid):
 - $B_{1,1} \implies P_{1,2} \vee P_{2,1}$
 - $B_{1,2} \implies P_{1,2} \vee P_{1,3} \vee P_{2,2}$
 - ...
 - ...
- **We want to be able to say this in one single sentence.**

How to say it in one sentence

- Our statement above refers to 2 types objects (*pit* and *square*). The square has the property to be *breezy*. The relationship between a square and pit is *adjacency*, i.e., neighbourhood.
- In FOL, this statement is represented by means of the following formula - instead of 16 sentences in propositional logic,:
$$\forall \text{ square}, \text{adjacent}(\text{square},\text{pit}) \implies \text{breezy}(\text{square})$$

First Order Logic (FOL)

- More expressive than propositional logic. While PL assumes that the world contains facts, FOL assumes the world contains
 - Objects: trees, people, numbers, movies, Trump, maps, colours, hypotheses, Wumpus....

First Order Logic (FOL)

- More expressive than propositional logic. While PL assumes that the world contains facts, FOL assumes the world contains
 - Objects: trees, people, numbers, movies, Trump, maps, colours, hypotheses, Wumpus....
 - Relations: square, smelly, brother, older than, owns, has colour, adjacent to....

First Order Logic (FOL)

- More expressive than propositional logic. While PL assumes that the world contains facts, FOL assumes the world contains
 - Objects: trees, people, numbers, movies, Trump, maps, colours, hypotheses, Wumpus....
 - Relations: square, smelly, brother, older than, owns, has colour, adjacent to....
 - Functions: brother of, colour-of, adjacent to,....

Syntax of FOL - elements

- Constants represents objects. NTNU, KingHarald, 5, ...
- Predicates represents relations. Brother, $>$, $=$, ...
- Functions represents functions: Sqrt, LeftLegOf
- Variables: x , y , a , b , ...
- Connectives: \neg , \wedge , \vee , \rightarrow , \leftrightarrow
- Quantifiers: \forall , \exists

Functions versus Relations

- Functions are a way of referring to individuals indirectly, e.g., $\text{BrotherOf}(\text{Janne})$ and Edvard would refer to the same individual if Janne 's brother is the person named Edvard .

Functions versus Relations

- Functions are a way of referring to individuals indirectly, e.g.,
 $\text{BrotherOf}(\text{Janne})$ and Edvard would refer to the same individual if
Janne's brother is the person named Edvard.
- Relations hold among objects - unary, binary, n-ary relations
 $\text{Brother}(\text{Janne}, \text{Edvard})$ is true if Edvard is Janne's brother

Syntax

- Atomic sentence: $\text{predicate}(\text{term}_1, \dots, \text{term}_n)$
Term: constant, variable, or $\text{function}(\text{term}_1, \dots, \text{term}_m)$
- Complex sentences: Composed of atomic sentences using connectives

$\neg S, S_1 \vee S_2, S_1 \wedge S_2, S_1 \implies S_2, S_1 \Leftrightarrow S_2$

Examples:

- $\text{Brother}(\text{JonSnow}, \text{AryaStark}) \implies \text{Sister}(\text{AryaStark}, \text{JonSnow})$
- $\text{Dreaming}(\text{Garcia}) \vee \text{Today}(\text{Monday})$

Quantifiers

- Quantifications express properties of collections of objects.
- Universal Quantifier, \forall : "For all"
- $\forall \langle \text{variable} \rangle \langle \text{sentence} \rangle$

Quantifiers

- Quantifications express properties of collections of objects.
- Universal Quantifier, \forall : "For all"
- $\forall \langle \text{variable} \rangle \langle \text{sentence} \rangle$
- We can state $\forall x P(x)$
 - English translation: "for all values of x , $P(x)$ is true"
 - Example: $P(x) : x+1 \geq x$
 - English translation: "for all values of x , $x+1 \geq x$ is true"

Universal quantification

- Everyone at NTNU is smart:
 $\forall x \text{ } At(x, \textit{NTNU}) \implies \textit{Smart}(x)$
- $\forall x \text{ } P$ is true in a model m iff P is true with x being each possible object in the model

Universal quantification

- Everyone at NTNU is smart:
 $\forall x \text{ } At(x, \text{NTNU}) \implies Smart(x)$
- $\forall x \text{ } P$ is true in a model m iff P is true with x being each possible object in the model
- Equivalent to the conjunction of instantiations of P

$$\begin{aligned} & (At(\text{KingJohn}, \text{NTNU}) \implies Smart(\text{KingJohn})) \\ \wedge \quad & (At(\text{Richard}, \text{NTNU}) \implies Smart(\text{Richard})) \\ \wedge \quad & (At(\text{NTNU}, \text{NTNU}) \implies Smart(\text{NTNU})) \\ \wedge \quad & \dots \end{aligned}$$

Existential quantification

- \exists : "There exist a/some"
- $\exists \langle \text{variables} \rangle \ \langle \text{sentence} \rangle$
- Someone at NTNU is smart:
 $\exists x \ At(x, \text{NTNU}) \wedge Smart(x)$

Existential quantification

- \exists : "There exist a/some"
- $\exists \langle \text{variables} \rangle \ \langle \text{sentence} \rangle$
- Someone at NTNU is smart:
 $\exists x \ At(x, \text{NTNU}) \wedge Smart(x)$
- $\exists x \ P$ is true in a model m iff P is true with x being **some** possible object in the model
- Equivalent to the **disjunction** of instantiations of P
 - ($At(\text{KingJohn}, \text{NTNU}) \wedge Smart(\text{KingJohn})$)
 - \vee ($At(\text{Richard}, \text{NTNU}) \wedge Smart(\text{Richard})$)
 - \vee ($At(\text{NTNU}, \text{NTNU}) \wedge Smart(\text{NTNU})$)
 - \vee ...

Exercise: A Few First-Order Logic Translations

Convert these sentences to First Order Logic

- I am happy only if my spouse is happy.
- Claudia is happy if every one of her friends clicks *like* for all of her posts.
- Claudia is happy only if at least one of her friends *shares* all of her posts.
- Wherever you go, Big Brother is watching you.
- Joe never works past 5 pm.

Answers

These work. Others are possible as well.

- $\text{Happy}(\text{me}) \Rightarrow \text{Happy}(\text{spouse}(\text{me}))$:: In most countries, spouse is a function:
single output for any input

...alternatively....

- $\exists Y : \text{spouse}(\text{me}, Y) \wedge [\text{Happy}(\text{me}) \rightarrow \text{Happy}(Y)]$
-

$$\begin{aligned} [\forall F, P : (\text{post}(\text{Claudia}, P) \wedge \text{friend}(\text{Claudia}, F)) \Rightarrow \text{likes}(F, P)] \\ \Rightarrow \text{Happy}(\text{Claudia}) \end{aligned}$$

-

$$\begin{aligned} \text{Happy}(\text{Claudia}) \Rightarrow \\ [\exists F \forall P : (\text{friend}(\text{Claudia}, F) \wedge \text{post}(\text{Claudia}, P)) \Rightarrow \text{shares}(F, P)] \end{aligned}$$

- $\forall X : [\text{location}(X) \wedge \text{at}(\text{You}, X)] \Rightarrow \text{watching}(\text{BigBro}, \text{You}, X)$
- $\forall T : [\text{time}(T) \wedge \text{works}(\text{Joe}, T)] \Rightarrow \sim \text{later}(T, 5pm)$

Example for comparison of Propositional Logic and FOL

Primitives in Propositional Logic. Ski-race example.¹

Objects

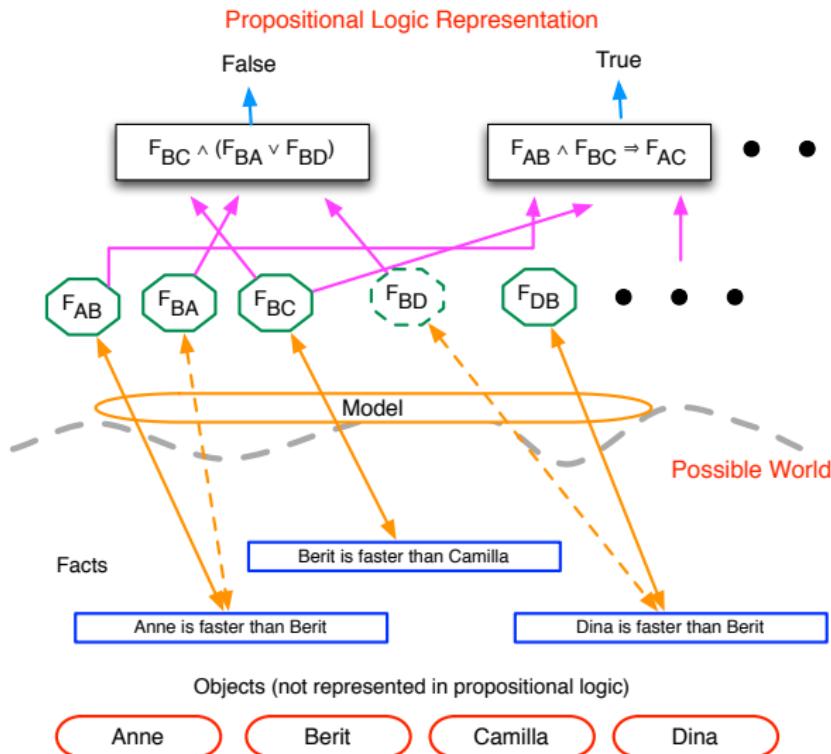
- Anne, Berit , Camilla, Dina
- These are not actually represented in propositional logic.
- Only True-or-False facts **about** them are represented.
- Objects alone do not have a truth value, whereas all primitives in propositional logic do.

Propositional Symbols

- F_{AB} : (Anne is faster than Berit) ; F_{BA} : (Berit is faster than Anne), ..
 F_{AC}
- These have truth values and are atomic.
- Their logical combinations into sentences (representing specific facts or general rules) also have truth values.

¹Thanks to Keith Downing for this example

Example- Ski-race in Propositional Logic



Example- Ski Race in FOL

Objects

- Anne, Berit, Camilla, Dina
- These are now represented in the logic, even though they still have no truth value.

Functions

- `best10K(person) → time`. Mapping from athlete to their best 10K time.
- `rank(person) → integer`. Mapping from athlete to their seeding in the competition.
- `start(person) → integer`. Mapping from athlete to start order in the race, where slowest start first.
- These have no truth value and map one primitive object (person) to another (number).

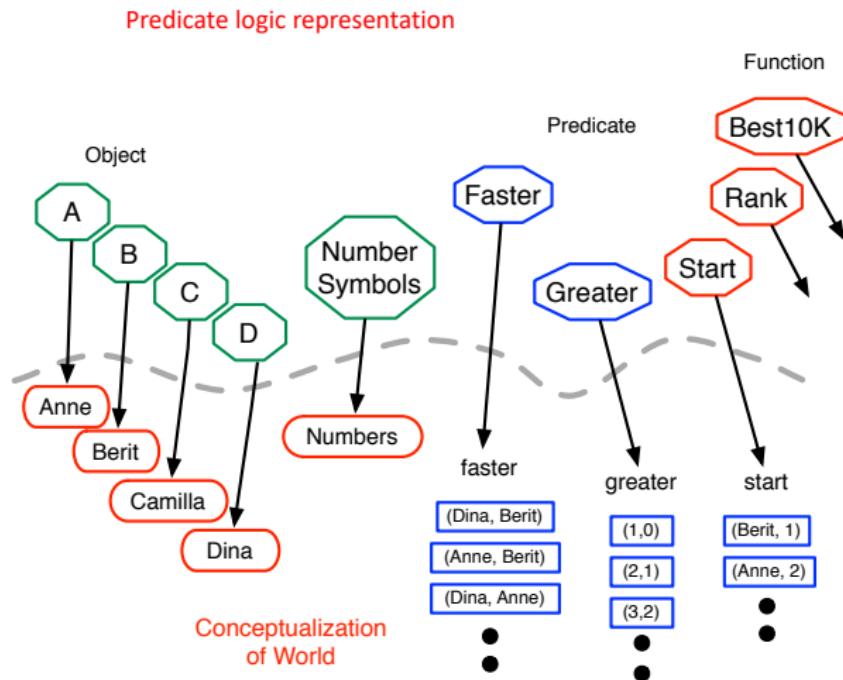
Example- Ski Race in FOL

Relations

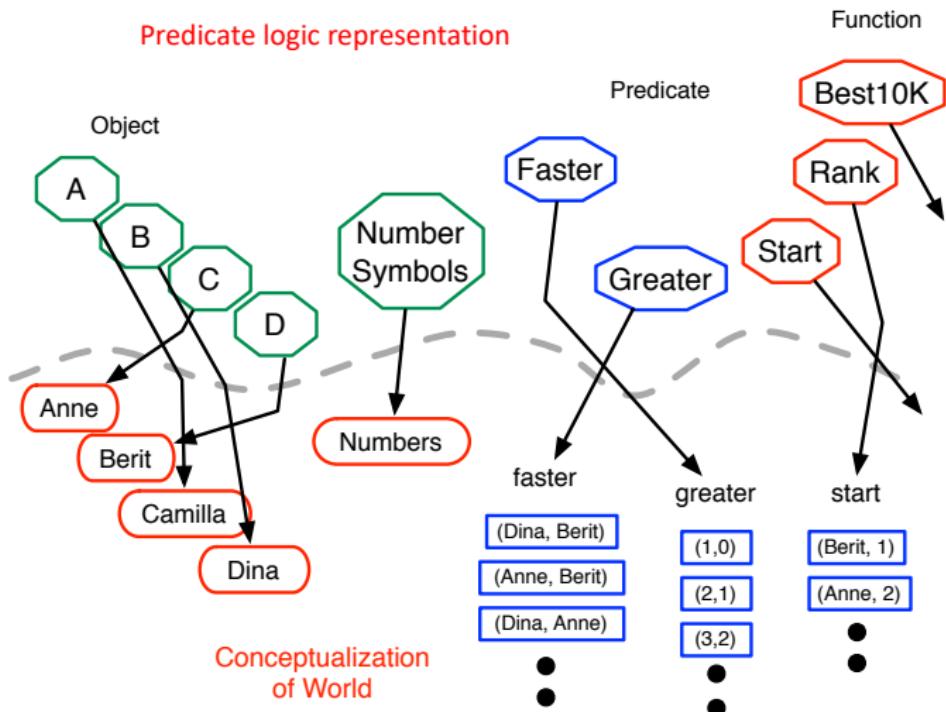
- $\text{greater}(X,Y) \rightarrow \{\text{True}, \text{False}\}$. Is number X greater than number Y?
- $\text{faster}(X,Y) \rightarrow \{\text{True}, \text{False}\}$. Is athlete X faster than athlete Y?
- These always have a truth value.
- These are often viewed as explicit lists of tuples, one list for each TRUE relation. So in one possible world, **faster** is represented by:
 $\{ (\text{anne}, \text{berit}), (\text{anne}, \text{camilla}), (\text{dina}, \text{anne}), (\text{dina}, \text{camilla}), (\text{camilla}, \text{berit}), (\text{dina}, \text{berit}) \}$

Interpretations in First-Order Logic

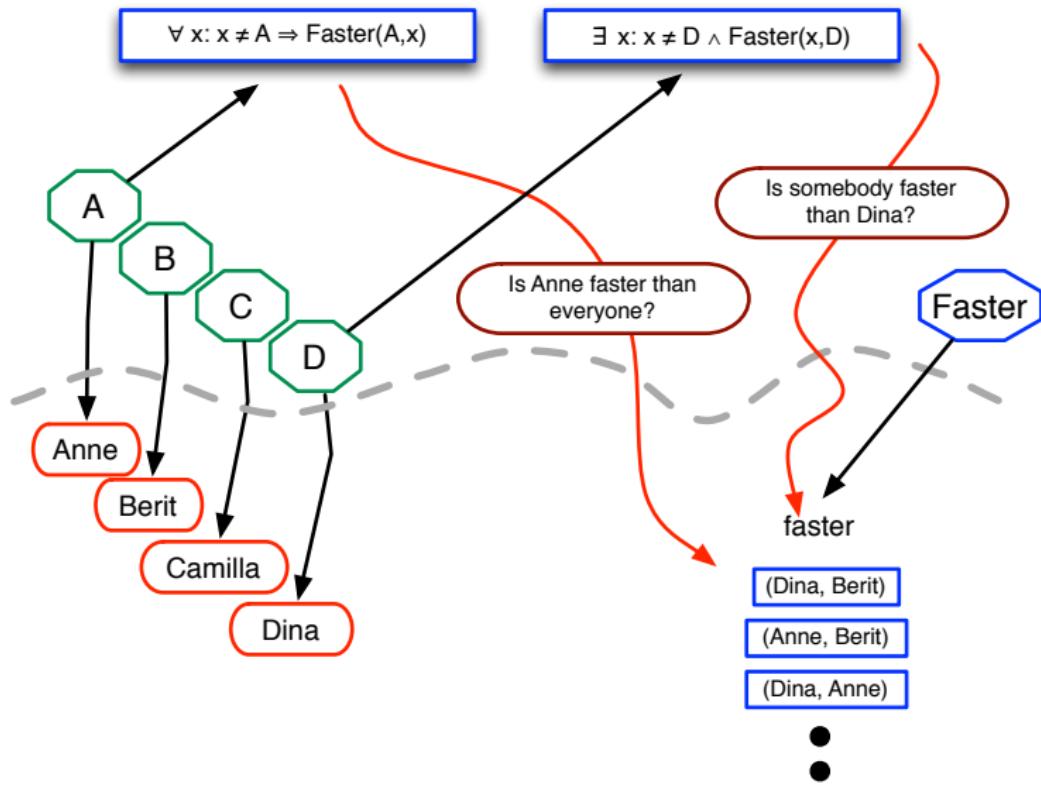
Interpretation = Mapping from constant, function and predicate symbols of the representation to the conceptualization.



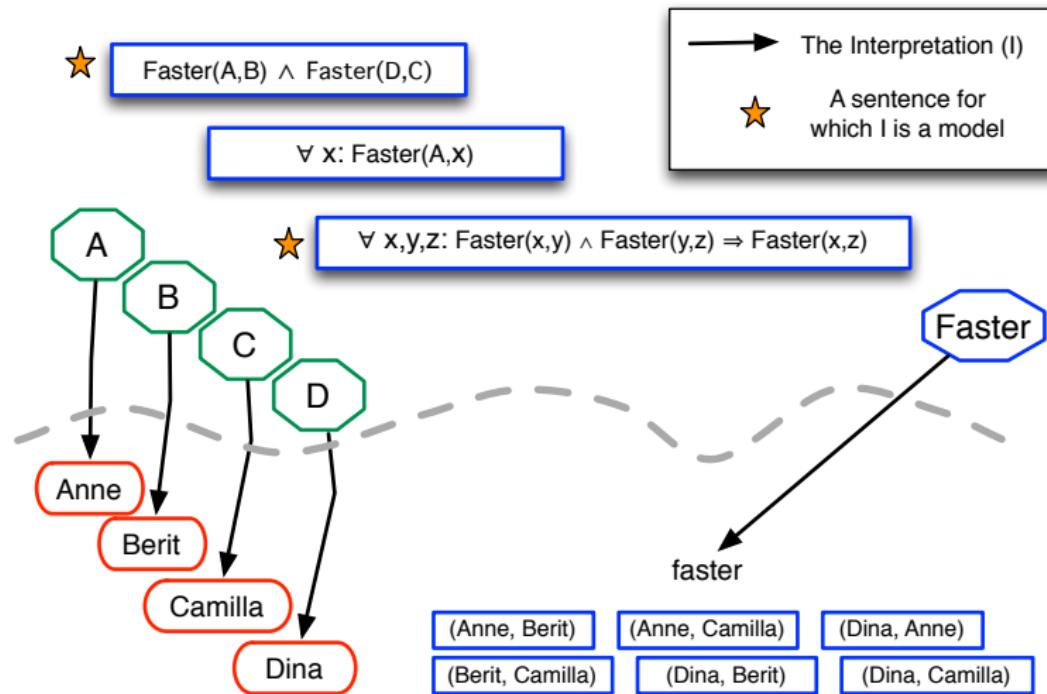
Another Legal Interpretation



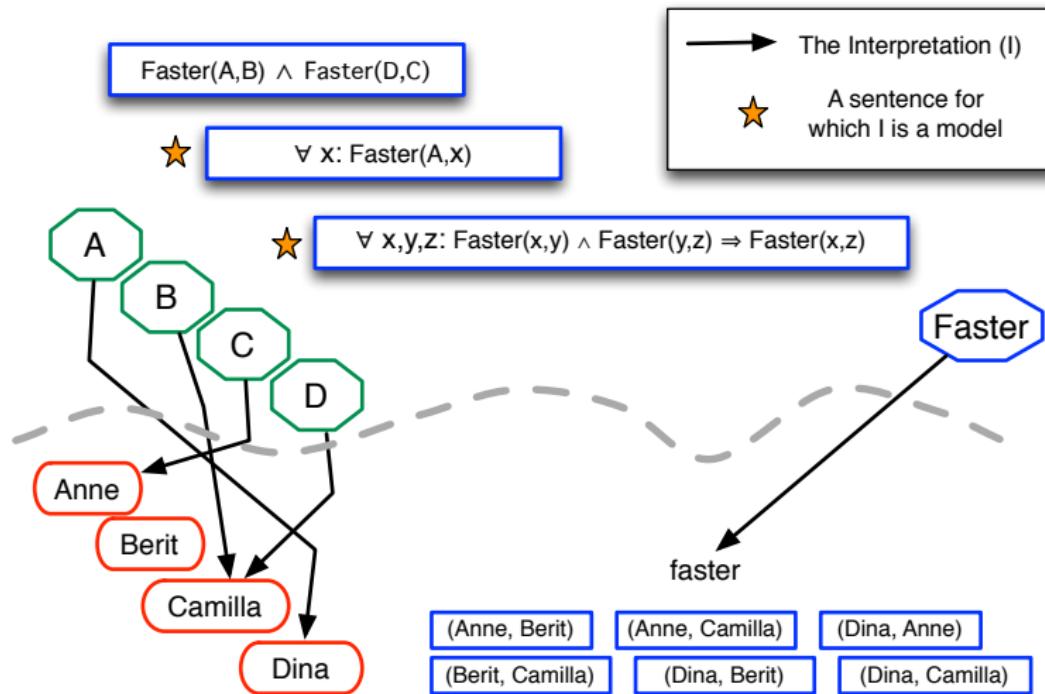
Evaluating Sentences with Quantified Variables



Example - Ski Models, Interpretation 1



Example - Ski Models for Interpretation 2



Common mistakes with Quantifiers

- Typically, \Rightarrow is the main connective with \forall
- Common mistake: using \wedge as the main connective with \forall :
- Let us represent this sentence: "Everybody at NTNU is smart"

Common mistakes with Quantifiers

- Typically, \Rightarrow is the main connective with \forall
- Common mistake: using \wedge as the main connective with \forall :
- Let us represent this sentence: "Everybody at NTNU is smart"

$\forall x \ At(x, NTNU) \wedge Smart(x)$

– Correct?

Common mistakes with Quantifiers

- Typically, \Rightarrow is the main connective with \forall
- Common mistake: using \wedge as the main connective with \forall :
- Let us represent this sentence: "Everybody at NTNU is smart"

$$\forall x \ At(x, \text{NTNU}) \wedge \text{Smart}(x)$$

- Correct?
- No, it means "Everyone is at NTNU and everyone is smart"

Another common mistake to avoid

- Typically, \wedge is the main connective with \exists
- Common mistake: using \Rightarrow as the main connective with \exists :
- Let us represent this sentence: "There is a smart person at NTNU"

Another common mistake to avoid

- Typically, \wedge is the main connective with \exists
- Common mistake: using \implies as the main connective with \exists :
- Let us represent this sentence: "There is a smart person at NTNU"

$\exists x \ At(x, NTNU) \implies Smart(x)$

– Correct?

Another common mistake to avoid

- Typically, \wedge is the main connective with \exists
- Common mistake: using \implies as the main connective with \exists :
- Let us represent this sentence: "There is a smart person at NTNU"

$\exists x \ At(x, NTNU) \implies Smart(x)$

- Correct?
- No, because it becomes true if there is anyone who is not at NTNU!

Connections between \forall and \exists

- All statements made with one quantifier can be converted into equivalent statements with the other quantifier by using negation.
- Negation rules/laws:
 - $\neg \exists \equiv \forall \neg$

Connections between \forall and \exists

- All statements made with one quantifier can be converted into equivalent statements with the other quantifier by using negation.
- Negation rules/laws:
 - $\neg A \equiv \exists \neg$
 - $\neg \exists \equiv \forall \neg$

Connections between \forall and \exists

- All statements made with one quantifier can be converted into equivalent statements with the other quantifier by using negation.
- Negation rules/laws:
 - $\neg A \equiv \exists \neg$
 - $\neg \exists \equiv \forall \neg$
 - $\exists \neg \equiv \neg \forall$

Connections between \forall and \exists

- All statements made with one quantifier can be converted into equivalent statements with the other quantifier by using negation.
- Negation rules/laws:
 - $\neg A \equiv \exists \neg$
 - $\neg \exists \equiv \forall \neg$
 - $\exists \equiv \neg \forall \neg$
 - $A \equiv \neg \exists \neg$

Connections between \forall and \exists

- All statements made with one quantifier can be converted into equivalent statements with the other quantifier by using negation.
- Negation rules/laws:
 - $\neg A \equiv \exists \neg$
 - $\neg \exists \equiv \forall \neg$
 - $\exists \equiv \neg \forall \neg$
 - $A \equiv \neg \exists \neg$

Connections between \forall and \exists

- Remember De Morgan's Rules
 - $P \wedge Q \equiv (\neg(\neg P \vee \neg Q))$
 - $P \vee Q \equiv (\neg(\neg P \wedge \neg Q))$
 - $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$
 - $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
- Generalized De Morgan's rules
 - $\forall x P(x) \equiv \neg \exists x (\neg P(x))$
 - $\exists x P(x) \equiv \neg \forall x (\neg P(x))$
 - $\neg \forall x P(x) \equiv \exists x (\neg P(x))$
 - $\neg \exists x P(x) \equiv \forall x (\neg P(x))$

Multiple Quantifiers

More complex sentences can be formulated by multiple variables and by nesting quantifiers

- "For all x , there exists a y such that $P(x,y)$ "
 - $\forall x \exists y P(x,y)$
 - Example: $\forall x \exists y (x + y = 0)$

Multiple Quantifiers

More complex sentences can be formulated by multiple variables and by nesting quantifiers

- "For all x , there exists a y such that $P(x,y)$ "
 - $\forall x \exists y P(x,y)$
 - Example: $\forall x \exists y (x + y = 0)$
- "There exists an x such that for all y $P(x,y)$ is true"
 - $\exists x \forall y P(x,y)$
 - Example: $\exists x \forall y (x * y = 0)$

Order of Multiple Quantifiers

- Reversing the order of the same type of consecutive quantifiers does not change the truth value of a sentence

$\forall x \forall y P(x, y)$ is the same as $\forall y \forall x P(x, y)$

- $\forall x, \forall y \text{ Parent}(x, y) \implies \text{Child}(y, x)$ can be written as
 $\forall y \forall x \text{Parent}(x, y) \implies \text{Child}(y, x)$

Order of Multiple Quantifiers

Order of consecutive quantifiers of opposite type cannot be changed.

- $\forall x \exists y \text{ Loves}(x, y)$
 - Everyone in the world loves someone.
 - With parentheses: $\forall x (\exists y \text{ Loves}(x, y))$
 - y is inside the scope of x

Order of Multiple Quantifiers

Order of consecutive quantifiers of opposite type cannot be changed.

- $\forall x \exists y \text{ Loves}(x, y)$

- Everyone in the world loves someone.
- With parentheses: $\forall x (\exists y \text{ Loves}(x, y))$
- y is inside the scope of x

- $\exists x \forall y \text{ Loves}(x, y)$

- There is a person who loves everyone in the world.
- The same person x loves everybody.
- y is inside the scope of x

Order of Multiple Quantifiers

Order of consecutive quantifiers of opposite type cannot be changed.

- $\forall x \exists y \text{ Loves}(x, y)$

- Everyone in the world loves someone.
- With parentheses: $\forall x (\exists y \text{ Loves}(x, y))$
- y is inside the scope of x

- $\exists x \forall y \text{ Loves}(x, y)$

- There is a person who loves everyone in the world.
- The same person x loves everybody.
- y is inside the scope of x

- $\exists y \forall x \text{ Loves}(x, y)$

- There is someone whom everybody likes
- Everybody likes the same y.
- x is inside the scope of y.

Order of Multiple Quantifiers

Order of consecutive quantifiers of opposite type cannot be changed.

- $\forall x \exists y \text{ Loves}(x, y)$

- Everyone in the world loves someone.
- With parentheses: $\forall x (\exists y \text{ Loves}(x, y))$
- y is inside the scope of x

- $\exists x \forall y \text{ Loves}(x, y)$

- There is a person who loves everyone in the world.
- The same person x loves everybody.
- y is inside the scope of x

- $\exists y \forall x \text{ Loves}(x, y)$

- There is someone whom everybody likes
- Everybody likes the same y.
- x is inside the scope of y.

- What about "There is a puppy that likes every woman."

Order of Multiple Quantifiers

Order of consecutive quantifiers of opposite type cannot be changed.

- $\forall x \exists y \text{ Loves}(x, y)$

- Everyone in the world loves someone.
- With parentheses: $\forall x (\exists y \text{ Loves}(x, y))$
- y is inside the scope of x

- $\exists x \forall y \text{ Loves}(x, y)$

- There is a person who loves everyone in the world.
- The same person x loves everybody.
- y is inside the scope of x

- $\exists y \forall x \text{ Loves}(x, y)$

- There is someone whom everybody likes
- Everybody likes the same y.
- x is inside the scope of y.

- What about "There is a puppy that likes every woman."

$$\exists p \text{Puppy}(p) \wedge (\forall w \text{Woman}(w) \implies \text{Loves}(p, w))$$

Negating multiple Quantifiers

- Recall negation rules for single quantifiers:

$$\neg \forall x P(x) = \exists x \neg P(x)$$

$$\neg \exists x P(x) = \forall x \neg P(x)$$

- You change the quantifier(s), and negate what it's quantifying:

Negating multiple Quantifiers

- Recall negation rules for single quantifiers:

$$\neg \forall x P(x) = \exists x \neg P(x)$$

$$\neg \exists x P(x) = \forall x \neg P(x)$$

- You change the quantifier(s), and negate what it's quantifying:

$$\neg(\forall x \exists y P(x, y)) \equiv \exists x \neg \exists y P(x, y) \equiv \exists x \forall y \neg P(x, y)$$

Higher-Order Logic

- FOL is called first-order because it allows quantifiers to range over objects but not properties, relations, or functions applied to those objects.
- Second-order logic allows quantifiers to range over predicates and functions as well:
 - $\forall x \forall y [(x = y) \leftrightarrow (\forall P P(x) \leftrightarrow P(y))]$
Means that two objects are equal if and only if they have exactly the same properties.

Higher-Order Logic

- FOL is called first-order because it allows quantifiers to range over objects but not properties, relations, or functions applied to those objects.
- Second-order logic allows quantifiers to range over predicates and functions as well:

- $\forall x \forall y [(x = y) \leftrightarrow (\forall P P(x) \leftrightarrow P(y))]$

Means that two objects are equal if and only if they have exactly the same properties.

- $\forall F \forall G [(F = G) \leftrightarrow (\forall x F(x) = G(x))]$

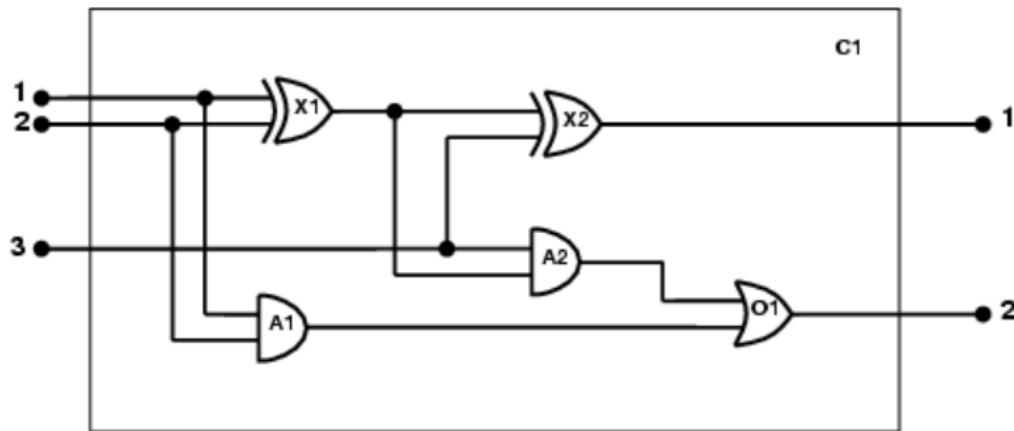
Means that two functions are equal if and only if they have the same value for all possible arguments.

Knowledge engineering in FOL

- ① Identify the problem/task you want to solve
- ② Assemble the relevant knowledge
- ③ Decide on a vocabulary of predicates, functions, and constants
- ④ Encode general knowledge about the domain
- ⑤ Encode a description of the specific problem instance
- ⑥ Pose queries to the inference procedure and get answers
- ⑦ Debug the knowledge base

The electronic circuits domain

One-bit full adder



Knowledge engineering in FOL

1. Identify the task
 - Does the circuit actually add properly? (circuit verification)
2. Assemble the relevant knowledge
 - Composed of wires and gates; Types of gates (AND, OR, XOR, NOT)
 - Irrelevant attributes: size, shape, colour, cost of gates

Knowledge engineering in FOL

Decide on a vocabulary. Gate, Circuit, Terminal, Out, Connected,. Type....

Alternatives:

Type(X1) = XOR

Type(X1, XOR)

XOR(X1)

Knowledge engineering in FOL

Encode general knowledge of the domain.

- $\forall t_1, t_2 \text{ Connected}(t_1, t_2) \implies \text{Signal}(t_1) = \text{Signal}(t_2)$
- $\forall t \text{ Signal}(t) = 1 \vee \text{Signal}(t) = 0$
- $1 \neq 0$
- $\forall t_1, t_2 \text{ Connected}(t_1, t_2) \implies \text{Connected}(t_2, t_1)$
- $\forall g \text{ Type}(g) = \text{OR} \implies \text{Signal}(\text{Out}(1,g)) = 1 \iff \exists n \text{ Signal}(\text{In}(n,g)) = 1$
- $\forall g \text{ Type}(g) = \text{AND} \implies \text{Signal}(\text{Out}(1,g)) = 0 \iff \exists n \text{ Signal}(\text{In}(n,g)) = 0$
- $\forall g \text{ Type}(g) = \text{XOR} \implies \text{Signal}(\text{Out}(1,g)) = 1 \iff \text{Signal}(\text{In}(1,g)) \neq \text{Signal}(\text{In}(2,g))$
- $\forall g \text{ Type}(g) = \text{NOT} \implies \text{Signal}(\text{Out}(1,g)) \neq \text{Signal}(\text{In}(1,g))$

Knowledge engineering in FOL

Encode the specific problem instance

- Type(X1) = XOR Type(X2) = XOR
- Type(A1) = AND Type(A2) = AND
- Type(O1) = OR

- Connected(Out(1,X1),In(1,X2))
- Connected(Out(1,X1),In(2,A2))
- Connected(Out(1,A2),In(1,O1))
- Connected(Out(1,A1),In(2,O1))
- Connected(Out(1,X2),Out(1,C1))
- Connected(Out(1,O1),Out(2,C1))

- Connected(In(1,C1),In(1,X1))
- Connected(In(1,C1),In(1,A1))
- Connected(In(2,C1),In(2,X1))
- Connected(In(2,C1),In(2,A1))
- Connected(In(3,C1),In(2,X2))
- Connected(In(3,C1),In(1,A2))

Knowledge engineering in FOL

Pose queries to the inference procedure.

What are the possible sets of values of all the terminals for the adder circuit?

$$\exists i1,i2,i3,o1,o2 \text{ Signal}(In(1,C1)) = i1 \wedge \text{Signal}(In(2,C1)) = i2 \wedge \\ \text{Signal}(In(3,C1)) = i3 \wedge \text{Signal}(Out(1,C1)) = o1 \wedge \\ \text{Signal}(Out(2,C1)) = o2$$

Debug the KB

- Perturb the KB to see what kinds of erroneous. Behaviors emerge.

Knowledge Engineering for Euclidean Geometry



**Can we give a First-Order Logic Theorem Prover
enough Knowledge to do Euclidean Geometry?**

The Knowledge Engineering Process

- ① Identify questions to be asked of the Knowledge Base.
- ② Knowledge Acquisition - gather the proper kwg *
- ③ Ontology Design - determine constants, predicates and functions
- ④ Encode general kwg about the domain – e.g. axioms that define the main concepts.
- ⑤ Encode an instance in the problem domain.**
- ⑥ Pose queries and get answers.
- ⑦ Debug

* Kwg is sometimes used as an abbreviation for Knowledge

** Steps 4 and 5 will often reveal faults in the ontology and encoded domain kwg, thus forcing a return to step 3. Many such returns were necessary during the making of these slides !

Step 1: Identify Relevant Knowledge-Base Queries

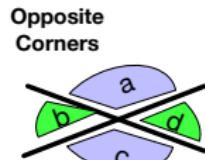
- Are angles ABC and ABD adjacent? Are they supplementary?
- Are angles ABC and XYZ congruent (same degrees, radians)?
- Do angles ABC and DBE refer to the same location (a.k.a. corner)?
- Are points P and Q on the same side of line L?
- Are lines L and M parallel? Perpendicular?
... and eventually....
- Are triangles ABC and ADE congruent? Similar?
- Are the ratios of the lengths of two pairs of segments equal?

Step 2: Identify Relevant Knowledge

- Two lines can intersect at at most one point.
- Intersecting lines form two pairs of opposite angles.
- Opposite angles are congruent.
- Parallel lines form corresponding angles.
- Corresponding angles are congruent.
- Many identical angles can be formed when two intersecting lines contain many points.
- Adjacent angles sum to a larger angle with the same corner point.
- Supplementary and Complementary angles are important.
- For any point (P) and line (L), P is either on L or on exactly one of L 's two sides.
- The side of line on which a point lies is important for determining angles and their adjacencies and equalities.
- Angles refer to unique locations (*corners*), but it's a many-to-one mapping. This could be tricky to represent!

Step 3: Design the Ontology

Basic Objects	
Points	
Lines	
Segments	
Corners	
Integers	{0,+1,-1, 90, 180}
Reals	(-inf, +inf)



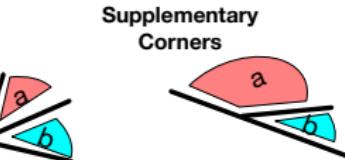
opposite(a,c)
opposite(b,d)

Opposite
Corners



adjacent(a,b)

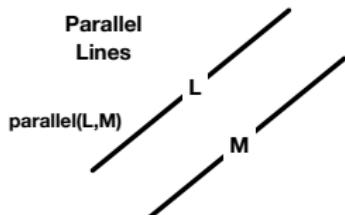
Adjacent
Corners



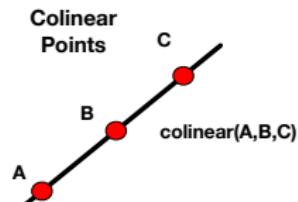
supplementary(a,b)

Supplementary
Corners

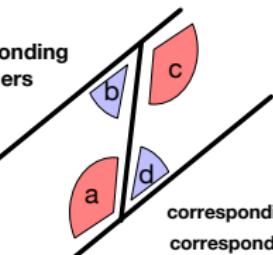
Relationships (Predicates)



parallel(L,M)



colinear(A,B,C)



corresponding(a,c)
corresponding(b,d)

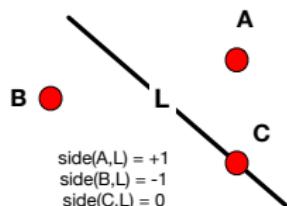
Corresponding
Corners

* These are predicates, since they map to a truth value: {True, False}

Step 3: Design the Ontology (continued)

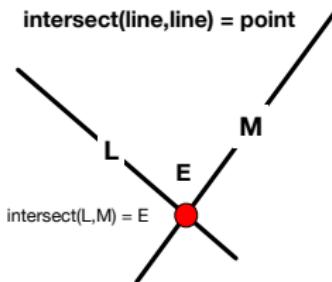
Functions

$\text{side(point, line)} = \{0, +1, -1\}$

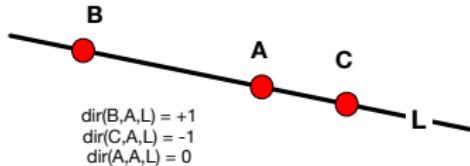


* These are functions, since every set of inputs maps to a single object.

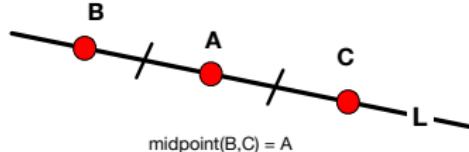
$\text{intersect(line, line)} = \text{point}$



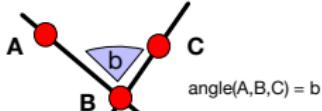
$\text{dir(point, point, line)} = \{0, +1, -1\}$



$\text{midpoint(point, point)} = \text{point}$



$\text{angle(point, point, point)} = \text{corner}$



$\text{measure(corner)} = (-180, 180]$

$\text{measure}(b) = 85^\circ$

Step 4: Encode General Domain Knowledge

- Defining *on*

$$\forall L, P : [line(L) \wedge pt(P) \wedge side(P, L) = 0] \Leftrightarrow on(P, L)$$

- Defining *colinear*

$$\begin{aligned} \forall A, B, C, L : pt(A) \wedge pt(B) \wedge pt(C) \wedge line(L) \Rightarrow \\ \{(on(A, L) \wedge on(B, L) \wedge on(C, L)) \Leftrightarrow colinear(A, B, C)\} \end{aligned}$$

- Defining *corresponding corners* (Note: $\text{angle}(A, B, C) \rightarrow \text{corner}$)

$$\begin{aligned} \forall L, M, N, A, B, C, D : line(L) \wedge line(M) \wedge line(N) \wedge \\ pt(A) \wedge pt(B) \wedge pt(C) \wedge pt(D) \Rightarrow \\ \{[\text{parallel}(L, M) \wedge \text{intersect}(L, N) = B \wedge \text{intersect}(M, N) = \\ C \wedge on(A, L) \wedge on(D, M) \wedge \text{side}(A, N) \neq \text{side}(D, N)] \\ \Leftrightarrow \text{corresponding}(\text{angle}(A, B, C), \text{angle}(B, C, D))\} \end{aligned}$$

- Equivalence of corresponding corners

$$\forall F, G : \text{corner}(F) \wedge \text{corner}(G) \Rightarrow \{\text{corresponding}(F, G) \Leftrightarrow (\text{measure}(F) = \text{measure}(G))\}$$

- Two lines can intersect at at most one point.

$$\forall L, M : [line(L) \wedge line(M) \wedge \text{intersect}(L, M, A) \wedge \text{intersect}(L, M, B)] \Rightarrow (A = B)$$

- Symmetry of angle function

$$\forall A, B, C : pt(A) \wedge pt(B) \wedge pt(C) \Rightarrow \text{angle}(A, B, C) = \text{angle}(C, B, A)$$

Step 4: Encode General Domain Knowledge (cont.)

- Defining separation

$$\begin{aligned} \forall L, A, B : & line(L) \wedge pt(A) \wedge pt(B) \Rightarrow \\ & \{(side(A, L) \neq side(B, L) \wedge side(A, L) \neq 0 \wedge side(B, L) \neq 0) \Leftrightarrow \\ & separates(L, A, B)\} \end{aligned}$$

- Defining opposite corners

$$\begin{aligned} \forall L, M, E : & line(L) \wedge line(M) \wedge pt(E) \wedge intersect(L, M) = E \Rightarrow \\ & \{(on(A, L) \wedge on(C, L) \wedge separates(M, A, C) \wedge on(B, M) \wedge on(D, M) \wedge \\ & separates(L, B, D)) \Leftrightarrow \\ & (opposite(angle(A, E, B), angle(D, E, C)) \wedge \\ & opposite(angle(B, E, C), angle(A, E, D)))\} \end{aligned}$$

- Equivalence of opposite corners

$$\forall F, G : corner(F) \wedge corner(G) \Rightarrow \{opposite(F, G) \Leftrightarrow (measure(F) = measure(G))\}$$

- Many angles can map to the same corner

$$\begin{aligned} \forall L, M, A, B, C, D : & line(L) \wedge line(M) \wedge pt(A) \wedge pt(B) \wedge pt(C) \wedge pt(D) \Rightarrow \\ & \{(intersect(L, M) = \\ & D \wedge on(A, L) \wedge on(B, M) \wedge on(C, M) \wedge side(B, L) = side(C, L)) \Rightarrow \\ & angle(A, D, B) = angle(A, D, C)\} \end{aligned}$$

Step 4: Encode General Domain Knowledge (cont.)

Defining adjacent angles/corners*

$\forall L, M, N \in \text{lines}, A, B, C, D \in \text{points} :$

$[D = \text{intersect}(L, M) \wedge D = \text{intersect}(M, N) \wedge \text{on}(A, M) \wedge \text{on}(B, L) \wedge \text{on}(C, N) \wedge \text{side}(C, M) \neq \text{side}(B, M)]$
 $\Rightarrow \text{adjacent}(\text{angle}(B, D, A), \text{angle}(C, D, A))$

Summing adjacent angles

$\forall A, B, C, D \in \text{points} : \text{adjacent}(\text{angle}(B, D, A), \text{angle}(C, D, A)) \rightarrow$
 $\text{measure}(\text{angle}(B, D, A)) + \text{measure}(\text{angle}(C, D, A)) =$
 $\text{measure}(\text{angle}(B, D, C))$

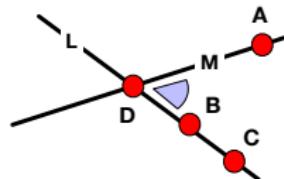
Supplementary Adjacent Angles

$\forall A, B, C, D \in \text{points} :$

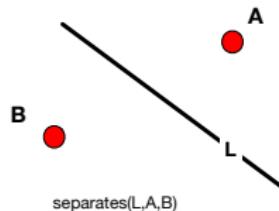
$\text{adjacent}(\text{angle}(B, D, A), \text{angle}(C, D, A)) \wedge \text{colinear}(B, D, C) \rightarrow$
 $\text{measure}(\text{angle}(B, D, C)) = 180^\circ$

*A perfect definition of adjacency is a bit more complicated.

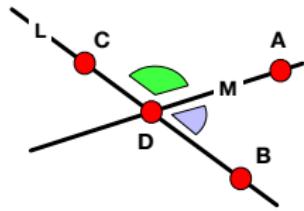
General Knowledge in Diagrams



$\text{angle}(A,D,B) = \text{angle}(A,D,C)$



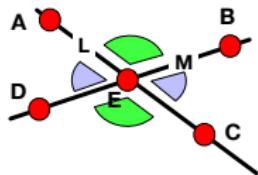
$\text{separates}(L,A,B)$



$\text{supplementary}(\text{angle}(B,D,A), \text{angle}(C,D,A))$

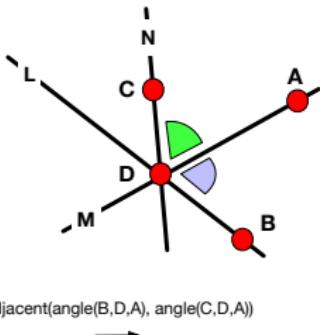
\longleftrightarrow

$\text{measure}(\text{angle}(B,D,A)) + \text{measure}(\text{angle}(C,D,A)) = 180^\circ$



$\text{opposite}(\text{angle}(A,E,B), \text{angle}(D,E,C))$

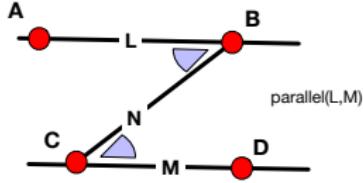
$\text{opposite}(\text{angle}(A,E,D), \text{angle}(B,E,C))$



$\text{adjacent}(\text{angle}(B,D,A), \text{angle}(C,D,A))$



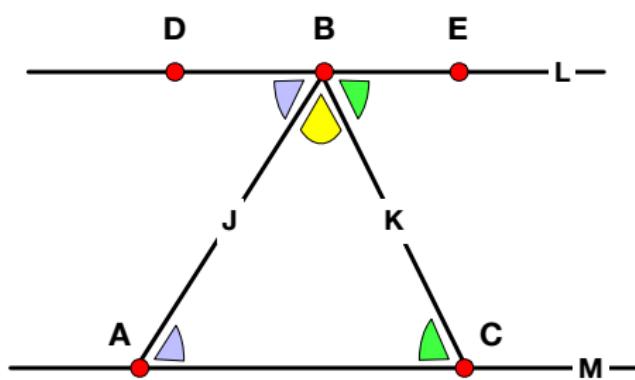
$\text{measure}(\text{angle}(B,D,A)) + \text{measure}(\text{angle}(C,D,A)) =$
 $\text{measure}(\text{angle}(B,D,C))$



$\text{parallel}(L,M)$

$\text{corresponding}(\text{angle}(A,B,C), \text{angle}(D,C,B))$

Step 5: Encode the Current Problem



Premises

line(J) line(K)
line(L) line(M)

pt(A) pt(B) pt(C)
pt(D) pt(E)

B = intersect(J,L)
B = intersect(K,L)
A = intersect(J,M)
C = intersect(K,M)

parallel(L,M)

side(D,J) = +1
side(D,K) = -1
side(D,L) = 0
side(D,M) = +1
side(A,J) = 0
side(A,M) = 0
side(A,K) = -1
side(A,L) = -1
:
:

Prove: $\text{angle}(B,A,C) + \text{angle}(A,C,B) + \text{angle}(A,B,C) = 180^\circ$

Step 6: Pose Queries and Get Answers

Query

- $m(\text{angle}(B,A,C)) + m(\text{angle}(A,C,B)) + m(\text{angle}(A,B,C)) = ??$
where $m() = \text{measure}()$

Answer (Hopefully)

- 180°
- The chain of deductions from the premises and the domain knowledge to 180°

Chain of Reasoning

$*m()$ = measure()

- $m^*(\angle ABD) + m(\angle ABE) = 180^\circ \therefore$ Adjacent, supplementary angles
- $m(\angle ABE) = m(\angle ABC) + m(\angle CBE) \therefore$ Adjacent angles
- $m(\angle ABD) + m(\angle ABC) + m(\angle CBE) = 180^\circ \therefore$ Simple substitution
- $\text{corresponding}(\angle ABD, \angle BAC) \therefore$ Def. of corresponding angles
- $\text{corresponding}(\angle CBE, \angle ACB) \therefore$ Def. of corresponding angles
- $m(\angle ABD) = m(\angle BAC) \therefore$ Equality of corresponding angles
- $m(\angle CBE) = m(\angle ACB) \therefore$ Equality of corresponding angles
- $m(\angle BAC) + m(\angle ACB) + m(\angle ABC) = 180^\circ \therefore$ Simple substitution

This is a general proof that the 3 angles of a triangle always sum to 180° .

Step 7: Debug

- The ontology and general knowledge base require frequent updates (e.g. Tore Amble's BusTuc)
- Early on, many queries force these updates.
- With each change, try to anticipate future queries and thus make the KB as general and flexible as possible.
- *Representation without reasoning is an idle exercise...Ken Forbus (well-known AI researcher)*
⇒ Building a truly general-purpose representation is a HARD job !!

Summary

Let us summarize together!

TDT4136 Introduction to Artificial Intelligence

Lecture 7 - Inference in First Order Logic

Chapter 9 in the textbook

Keith L. Downing

The Norwegian University of Science and Technology
Trondheim, Norway
keithd@idi.ntnu.no

October 2, 2024

Outline

- Two main types of inference in FOL
 - Reduction of FOL to propositional logic (and use propositional inference methods)
 - Substitution
 - Inference using FOL Inference rules
 - Unification
 - Standardizing apart
 - Conversion of FOL sentences to CNF
 - Inference using Resolution, on CNF representation
 - Forward/Backward Chaining, on Definite Clauses

Inference by Reduction to Propositionalized Sentences

Strategy: convert KB to propositional logic and then use PL inference

- Every FOL KB can be propositionalized so as to preserve entailment -
A sentence is entailed by new KB iff it is entailed by the original KB

Inference by Reduction to Propositionalized Sentences

Strategy: convert KB to propositional logic and then use PL inference

- Every FOL KB can be propositionalized so as to preserve entailment -
A sentence is entailed by new KB iff it is entailed by the original KB
- Ground atomic sentences become propositional symbols

Inference by Reduction to Propositionalized Sentences

Strategy: convert KB to propositional logic and then use PL inference

- Every FOL KB can be propositionalized so as to preserve entailment -
A sentence is entailed by new KB iff it is entailed by the original KB
- Ground atomic sentences become propositional symbols
- What about the quantifiers?

Propositionalization

- Suppose the KB contains just the following:
 - $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \implies \text{Evil}(x)$
 - $\text{King}(\text{John})$
 - $\text{Greedy}(\text{John})$
 - $\text{Brother}(\text{Richard}, \text{John})$
- The last 3 sentences can be symbols in PL
- Apply **Universal Instantiation** to the first sentence, to get rid of the *quantifier* and the *variable*.

Universal Instantiation

- UI says that from a universally quantified sentence, we can infer any sentence obtained by substituting a ground term¹ for the variable

¹A term is used to denote an object in the world. It can be a constant, variable or a function. A ground term is a term with no variables.

Universal Instantiation

- UI says that from a universally quantified sentence, we can infer any sentence obtained by substituting a ground term¹ for the variable
- The **Substitution** rule for instantiation of variables

$$\frac{\forall x \ \alpha}{\text{SUBST}(\{x/g\}, \alpha)}$$

variable x in the sentence α is substituted with a ground term g

¹A term is used to denote an object in the world. It can be a constant, variable or a function. A ground term is a term with no variables.

Universal Instantiation

- UI says that from a universally quantified sentence, we can infer any sentence obtained by substituting a ground term¹ for the variable
- The **Substitution** rule for instantiation of variables

$$\frac{\forall x \ \alpha}{\text{SUBST}(\{x/g\}, \alpha)}$$

variable x in the sentence α is substituted with a ground term g

- Variable x is *substituted* with the *ground terms* referring to the objects *John* and *Richard* in the model one by one:
 - $\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \implies \text{Evil}(\text{Richard})$
 - $\text{John} \ \text{King}(x) \wedge \text{Greedy}(\text{John}) \implies \text{Evil}(\text{John})$

¹A term is used to denote an object in the world. It can be a constant, variable or a function. A ground term is a term with no variables.

Existential Instantiation

- Through **Skolemization**: Each existentially quantified variable is replaced by a *Skolem constant* or a *Skolem function*.
- **Skolem Constant**: if the existential variable is not within the scope of any universally quantified variable. Every instance of the existentially quantified variable is replaced with the same unique constant, a brand new one that does not appear elsewhere in the knowledge base.

$$\frac{\exists x \ \alpha}{\text{SUBST}(\{x/k\}, \alpha)}$$

Example: $\exists y \ (P(y) \wedge Q(y))$ is converted to: $P(CC) \wedge Q(CC)$

Existential Instantiation

Skolem Function: If the existential quantifier is in the *scope* (i.e., "inside") of a (or more - n) universally quantified variable(s), then replace it with a unique n-ary function over these universally quantified variables. Remove then the existential quantifier.

E.g., $\forall x \exists y (P(x) \vee Q(y))$ converted to $\forall x P(x) \vee Q(F(x))$

Problems with UI

- ① instantiation of functions leads to infinite nr of instantiations
- ② creates many irrelevant instantiations

Problems with UI

- ① instantiation of functions leads to infinite nr of instantiations
- ② creates many irrelevant instantiations

Problem when a variable is substituted with a function:

- For example, suppose Father is a function in our KB:
- Then Father(John) is also an object/ground term

Problems with UI

- ① instantiation of functions leads to infinite nr of instantiations
- ② creates many irrelevant instantiations

Problem when a variable is substituted with a function:

- For example, suppose Father is a function in our KB:
- Then Father(John) is also an object/ground term
- Also Father(Father(John)) and Father(Father(Father(John)))..... are ground terms
- Infinitely many ground terms/instantiations

Problems with UI

- ① instantiation of functions leads to infinite nr of instantiations
- ② creates many irrelevant instantiations

Problem when a variable is substituted with a function:

- For example, suppose Father is a function in our KB:
- Then Father(John) is also an object/ground term
- Also Father(Father(John)) and Father(Father(Father(John)))..... are ground terms
- Infinitely many ground terms/instantiations
- Solution: Herbrand Theorem (1930). If a sentence α is entailed by an FOL KB, it is entailed by a finite subset of the propositionalized KB
 - Idea: For $n = 0$ to ∞ do
create a propositional KB by instantiating with depth- n terms
see if α is entailed by this KB

Propositionalization creates irrelevant instantiations

- Propositionalization seems to generate lots of "irrelevant sentences".
E.g., from

$$\forall x \ King(x) \wedge Greedy(x) \implies Evil(x)$$

King(John)

Greedy(John)

Brother(Richard, John)

Propositionalization creates irrelevant instantiations

- Propositionalization seems to generate lots of "irrelevant sentences".
E.g., from

$$\forall x \ King(x) \wedge Greedy(x) \implies Evil(x)$$

King(John)

Greedy(John)

Brother(Richard, John)

- It seems obvious that *Evil(John)* will be inferred at the end, but propositionalization produces lots of facts such as *Greedy(Richard)* that are irrelevant
- With p k -ary predicates and n constants, there are $p \cdot n^k$ instantiations

Problem with propositionalization

- Problem : Universal elimination gives us (too) many opportunities for substituting variables with ground terms
- Solution: avoid making blind substitution of ground terms
 - Make substitutions that help to advance inferences
 - i.e., use substitutions matching "similar" sentences in KB
- How? - Inference without propositionalization

Propositionalization is semi-decidable

- Propositionalization is complete if the query sentence is entailed.
- Works if α is entailed, loops if α is not entailed. Hence, it is semi-decidable.

Propositionalization is semi-decidable

- Propositionalization is complete if the query sentence is entailed.
- Works if α is entailed, loops if α is not entailed. Hence, it is semi-decidable.
- Semidecidable : algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence.

Inference in First-Order Logic - without propositionalization

- Inference rules for propositional logic: – Modus ponens, and-introduction, resolution, etc.
- These are valid for FOL also

Inference in First-Order Logic - without propositionalization

- Inference rules for propositional logic: – Modus ponens, and-introduction, resolution, etc.
- These are valid for FOL also
- But since these don't deal with quantifiers and variables, we need new rules, especially those that allow for substitution of variables with objects
- These are called lifted inference rules

Lifted Modus Ponens

Modus Ponens in propositional logic:

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

Generalized Modus Ponens - FOL

$$\frac{P_1', P_2', \dots, P_n', (P_1 \wedge P_2 \wedge \dots \wedge P_n \Rightarrow Q)}{Q\theta}$$

where $P_i'\theta = P_i\theta$ for all i , θ is Substitution

Lifted Inference Rules and Unification

- Generalized Modus ponens is the *lifted* version of Modus Ponens in PL(i.e., variable free version)
- Lifted Inferences need finding substitutions that make 2 sentences look identical.
- This is called **Unification**

Lifted Inference Rules and Unification

- Generalized Modus ponens is the *lifted* version of Modus Ponens in PL(i.e., variable free version)
- Lifted Inferences need finding substitutions that make 2 sentences look identical.
- This is called **Unification**
- UNIFICATION: takes two similar sentences and computes the **unifier** for them (a **substitution**) that makes them look the same, if it exists
 $\text{UNIFY}(P, Q) = \theta$; where $\text{SUBST } (\theta, P) = \text{SUBST } (\theta, Q)$

Motivating example for Unification

- Ground clauses are clauses with no variables in them. For ground clauses we can use syntactic identity to detect when we have a P and $\neg P$ pair.
- What about variables? For example, can the following two clauses be resolved?
 - $P(\text{john}) \vee Q(\text{fred}) \vee R(x)$
 - $\neg P(y) \vee R(\text{susan}) \vee R(y)$

Generalized Modus Ponens in FOL

- Suppose this KB:

$S1 : \forall x \ King(x) \wedge Greedy(x) \implies Evil(x)$

$S2 : King(John)$

$S3 : \forall y \ Greedy(y)$

- We want to find if John is Evil.
- Make a **substitution** for values of variables that make the premise of the implication identical to the sentences in the knowledge base.
 - P_1' is King(John) and P_1 is King(x)
 - P_2' is Greedy(y) and P_2 is Greedy(x)
 - θ is $\{x/John, y/John\}$,
 - Q is Evil(x)
 - SUBS(θ, Q) is Evil(John)

Unification process

- Unify procedure: $\text{Unify}(P, Q)$ takes two atomic (i.e. single predicates) sentences P and Q and returns a substitution that makes P and Q identical.
- The aim is be able to match literals even when they have variables.
- Rules for substitutions: Can replace a variable
 - by a constant.
 - by a variable.
 - by a function expression, as long as the function expression does not contain the variable.

Unifier: a substitution that makes two clauses resolvable. e.g.,
 $\theta: \{x_1/M; x_2/x_3; x_4/F(..)\}$

More on Unification

- Not all formulas can be unified - substitutions only affect variables.
 - Example: Consider the pair $P(F(x), A)$ and $P(y, F(w))$
 - This pair cannot be unified as there is no way of making $A = F(w)$ with a substitution.

More on Unification

- Not all formulas can be unified - substitutions only affect variables.
 - Example: Consider the pair $P(F(x), A)$ and $P(y, F(w))$
 - This pair cannot be unified as there is no way of making $A = F(w)$ with a substitution.
- In θ , each variable is paired at most once

More on Unification

- Not all formulas can be unified - substitutions only affect variables.
 - Example: Consider the pair $P(F(x), A)$ and $P(y, F(w))$
 - This pair cannot be unified as there is no way of making $A = F(w)$ with a substitution.
- In θ , each variable is paired at most once
- A variable's pairing term may not contain the variable directly or indirectly.
 - e.g. can't have substitution $\{ x/F(y), y/F(x) \}$

More on Unification

- Not all formulas can be unified - substitutions only affect variables.
 - Example: Consider the pair $P(F(x), A)$ and $P(y, F(w))$
 - This pair cannot be unified as there is no way of making $A = F(w)$ with a substitution.
- In θ , each variable is paired at most once
- A variable's pairing term may not contain the variable directly or indirectly.
 - e.g. can't have substitution $\{ x/F(y), y/F(x) \}$
- When unifying expressions P and R , the variable names in P and the variable names in R should be disjoint.

More on Unification

- Not all formulas can be unified - substitutions only affect variables.
 - Example: Consider the pair $P(F(x), A)$ and $P(y, F(w))$
 - This pair cannot be unified as there is no way of making $A = F(w)$ with a substitution.
- In θ , each variable is paired at most once
- A variable's pairing term may not contain the variable directly or indirectly.
 - e.g. can't have substitution $\{ x/F(y), y/F(x) \}$
- When unifying expressions P and R , the variable names in P and the variable names in R should be disjoint.
 - Yes: $\text{UNIFY}(\text{Loves (John, }x\text{)}, \text{Loves (y, Jane)}) \theta = \{ x/\text{Jane}, y/\text{John} \}$
 - No: $\text{UNIFY}(\text{Loves (John, }x\text{)}, \text{Loves (x, Jane)})$ – No unifier
 - Solution: Standardizing apart the variables - come back to this after a few slides

Most General Unifier

- Our aim is to be able to match conflicting literals (for the use of resolution), even when they have variables. Unification process determines whether there is a "specialization" that matches.
- However, we don't want to over specialize.

Most General Unifier-example

- Consider the two sentences:

$$\begin{aligned} \neg P(x) \vee S(x) \vee Q(\text{Noah}) \\ P(y) \vee R(y) \end{aligned}$$

Most General Unifier-example

- Consider the two sentences:

$$\begin{aligned}\neg P(x) \vee S(x) \vee Q(Noah) \\ P(y) \vee R(y)\end{aligned}$$

- Possible unifications:

$$(S(\text{Arvid}) \vee Q(\text{Noah}) \vee R(\text{Arvid})) \quad \{y=x, x=\text{Arvid}\}$$

Most General Unifier-example

- Consider the two sentences:

$$\begin{aligned}\neg P(x) \vee S(x) \vee Q(Noah) \\ P(y) \vee R(y)\end{aligned}$$

- Possible unifications:

$$\begin{aligned}(S(Arvid) \vee Q(Noah) \vee R(Arvid)) \quad \{y=x, x=Arvid\} \\ (S(Sophie) \vee Q(Noah) \vee R(Sophie)) \quad \{y=x, x=Sophie\}\end{aligned}$$

Most General Unifier-example

- Consider the two sentences:

$$\begin{aligned}\neg P(x) \vee S(x) \vee Q(Noah) \\ P(y) \vee R(y)\end{aligned}$$

- Possible unifications:

$$\begin{aligned}(S(Arvid) \vee Q(Noah) \vee R(Arvid)) \quad \{y=x, x=Arvid\} \\ (S(Sophie) \vee Q(Noah) \vee R(Sophie)) \quad \{y=x, x=Sophie\} \\ (S(x) \vee Q(Noah) \vee R(x)) \quad \{y=x\}\end{aligned}$$

Most General Unifier-example

- Consider the two sentences:

$$\begin{aligned}\neg P(x) \vee S(x) \vee Q(\text{Noah}) \\ P(y) \vee R(y)\end{aligned}$$

- Possible unifications:

$$\begin{aligned}(S(\text{Arvid}) \vee Q(\text{Noah}) \vee R(\text{Arvid})) & \{y=x, x=\text{Arvid}\} \\ (S(\text{Sophie}) \vee Q(\text{Noah}) \vee R(\text{Sophie})) & \{y=x, x=\text{Sophie}\} \\ (S(x) \vee Q(\text{Noah}) \vee R(x)) & \{y=x\}\end{aligned}$$

- The last unifier is the "most-general" one, the other two are specializations of it.
- We want to keep the most general clause so that we can use it in future resolution steps.

Most General Unifier

- Informally, the most general unifier (MGU) imposes the fewest constraints on the terms (contains the most variables).
- Formally, a substitution θ is more general than β iff there is a substitution σ such that $\theta \sigma = \beta$.
e.g. $\theta = z/F(w)$ is more general than $\beta = z/F(C)$ since $\sigma = w/C$

One more example on Most General Unifier

Consider clauses $P(f(x), z)$ and $P(y, A)$

- $\theta_1 = \{y = f(A), x = A, z = A\}$ is a unifier.

$$P(f(x), z)\theta_1 = P(f(A), A)$$

$$P(y, A)\theta_1 = P(f(A), A) \text{ but it is not MGU.}$$

One more example on Most General Unifier

Consider clauses $P(f(x), z)$ and $P(y, A)$

- $\theta_1 = \{y = f(A), x = A, z = A\}$ is a unifier.

$$P(f(x), z)\theta_1 = P(f(A), A)$$

$P(y, A)\theta_1 = P(f(A), A)$ but it is not MGU.

- $\theta_2 = \{y = f(x), z = A\}$ is an MGU.

$$P(f(x), z)\theta_2 = P(f(x), A)$$

$$P(y, A)\theta_2 = P(f(x), A)$$

Motivation for Standardizing apart

P	Q	θ
Knows(John,x)	Knows(John,Jane)	
Knows(John,x)	Knows(y,OJ)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,OJ)	

Standardizing apart

P	Q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,OJ)	{x/OJ, y/John}
Knows(John,x)	Knows(y,Mother(y))	{x/Mother(John),y/John}
Knows(John,x)	Knows(x,OJ)	No substitution possible yet. (i.e., "fails")

Standardizing apart

Knows (John, x) and Knows (x, OJ) cannot be unified, i.e, unifications fails.

- Intuitively we know that if we know John hates everyone he knows, and that everyone knows OJ. So we should be able to infer that John hates OJ.

Standardizing apart

Knows (John, x) and Knows (x, OJ) cannot be unified, i.e, unifications fails.

- Intuitively we know that if we know John hates everyone he knows, and that everyone knows OJ. So we should be able to infer that John hates OJ.
- This is why we require that every variable has a separate name.
- Need **Standardizing apart** that eliminates overlap of variables.

Standardizing apart

Knows (John, x) and Knows (x, OJ) cannot be unified, i.e, unifications fails.

- Intuitively we know that if we know John hates everyone he knows, and that everyone knows OJ. So we should be able to infer that John hates OJ.
- This is why we require that every variable has a separate name.
- Need **Standardizing apart** that eliminates overlap of variables.
- UNIFY *Knows(John, z₂₇)* and *Knows(z₁₇, OJ)* This works!!!
 $\{z_{17}/\text{John}, z_{27}/\text{OJ}\}$

Resolution Refutation in FOL

- The idea is the same as in Propositional Logic:
Our goal is to determine if $\text{KB} \models \alpha$:
 - ① Add $\neg\alpha$ to the KB
 - ② Convert KB and α to Conjunctive Normal Form
 - ③ Use the lifted/generalized resolution rule and search to determine whether the system is satisfiable (SAT)

Lifted Resolution Rule

Full first-order version:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \quad m_1 \vee \cdots \vee m_n}{(\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

where $\text{UNIFY}(\ell_i, \neg m_j) = \theta$.

Two standardized clauses can be resolved if they contain complementary literals (one is the negation of the other).

FOL literals are complementary if one *unifies* with the negation of the other.

Lifted Resolution is refutation-complete.

Resolution Refutation - example

Suppose we have the following knowledge in the KB:

The law says that it is a crime for an American to sell weapons to hostile nations.

The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

Resolution Refutation - example

Suppose we have the following knowledge in the KB:

The law says that it is a crime for an American to sell weapons to hostile nations.

The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

We want to prove that "**Colonel West is a criminal**".

Conversion to CNF - example

Let us first look at the procedure for conversion of FOL sentences to CNF through an example:

"Everyone who loves all animals is loved by someone"

Conversion to CNF - example

Everyone who loves all animals is loved by someone:

$$\forall x \ [\forall y \ Animal(y) \implies Loves(x, y)] \implies [\exists y \ Loves(y, x)]$$

1. Eliminate biconditionals and implications

Conversion to CNF - example

Everyone who loves all animals is loved by someone:

$$\forall x [\forall y \text{ } \textit{Animal}(y) \Rightarrow \textit{Loves}(x, y)] \Rightarrow [\exists y \text{ } \textit{Loves}(y, x)]$$

1. Eliminate biconditionals and implications

Conversion to CNF - example

Everyone who loves all animals is loved by someone:

$$\forall x [\forall y \text{ } \textit{Animal}(y) \Rightarrow \textit{Loves}(x, y)] \Rightarrow [\exists y \text{ } \textit{Loves}(y, x)]$$

1. Eliminate biconditionals and implications

$$\forall x \neg[\forall y \neg\textit{Animal}(y) \vee \textit{Loves}(x, y)] \vee [\exists y \textit{Loves}(y, x)]$$

Conversion to CNF - example

Everyone who loves all animals is loved by someone:

$$\forall x [\forall y \text{ } \textit{Animal}(y) \Rightarrow \textit{Loves}(x, y)] \Rightarrow [\exists y \text{ } \textit{Loves}(y, x)]$$

1. Eliminate biconditionals and implications

$$\forall x \neg[\forall y \neg\textit{Animal}(y) \vee \textit{Loves}(x, y)] \vee [\exists y \textit{Loves}(y, x)]$$

2. Move \neg inwards: $\neg\forall x, p \equiv \exists x \neg p$, $\neg\exists x, p \equiv \forall x \neg p$:

Conversion to CNF - example

Everyone who loves all animals is loved by someone:

$$\forall x [\forall y \text{ } \textit{Animal}(y) \Rightarrow \textit{Loves}(x, y)] \Rightarrow [\exists y \text{ } \textit{Loves}(y, x)]$$

1. Eliminate biconditionals and implications

$$\forall x \neg[\forall y \neg\textit{Animal}(y) \vee \textit{Loves}(x, y)] \vee [\exists y \textit{Loves}(y, x)]$$

2. Move \neg inwards: $\neg\forall x, p \equiv \exists x \neg p$, $\neg\exists x, p \equiv \forall x \neg p$:

$$\forall x [\exists y \neg(\neg\textit{Animal}(y) \vee \textit{Loves}(x, y))] \vee [\exists y \textit{Loves}(y, x)]$$

Conversion to CNF - example

Everyone who loves all animals is loved by someone:

$$\forall x [\forall y \text{ } \textit{Animal}(y) \Rightarrow \textit{Loves}(x, y)] \Rightarrow [\exists y \text{ } \textit{Loves}(y, x)]$$

1. Eliminate biconditionals and implications

$$\forall x \neg[\forall y \neg\textit{Animal}(y) \vee \textit{Loves}(x, y)] \vee [\exists y \textit{Loves}(y, x)]$$

2. Move \neg inwards: $\neg\forall x, p \equiv \exists x \neg p$, $\neg\exists x, p \equiv \forall x \neg p$:

$$\forall x [\exists y \neg(\neg\textit{Animal}(y) \vee \textit{Loves}(x, y))] \vee [\exists y \textit{Loves}(y, x)]$$

$$\forall x [\exists y \neg\neg\textit{Animal}(y) \wedge \neg\textit{Loves}(x, y)] \vee [\exists y \textit{Loves}(y, x)]$$

Conversion to CNF - example

Everyone who loves all animals is loved by someone:

$$\forall x [\forall y \text{ Animal}(y) \Rightarrow \text{Loves}(x, y)] \Rightarrow [\exists y \text{ Loves}(y, x)]$$

1. Eliminate biconditionals and implications

$$\forall x \neg[\forall y \neg\text{Animal}(y) \vee \text{Loves}(x, y)] \vee [\exists y \text{ Loves}(y, x)]$$

2. Move \neg inwards: $\neg\forall x, p \equiv \exists x \neg p$, $\neg\exists x, p \equiv \forall x \neg p$:

$$\forall x [\exists y \neg(\neg\text{Animal}(y) \vee \text{Loves}(x, y))] \vee [\exists y \text{ Loves}(y, x)]$$

$$\forall x [\exists y \neg\neg\text{Animal}(y) \wedge \neg\text{Loves}(x, y)] \vee [\exists y \text{ Loves}(y, x)]$$

$$\forall x [\exists y \text{ Animal}(y) \wedge \neg\text{Loves}(x, y)] \vee [\exists y \text{ Loves}(y, x)]$$

Conversion to CNF contd.

3. Standardize variables: each quantifier should use a different variable name

Conversion to CNF contd.

3. Standardize variables: each quantifier should use a different variable name

$$\forall x \ [\exists y \ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists z \ Loves(z, x)]$$

Conversion to CNF contd.

3. Standardize variables: each quantifier should use a different variable name

$$\forall x \ [\exists y \ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists z \ Loves(z, x)]$$

4. Skolemize: Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

Conversion to CNF contd.

3. Standardize variables: each quantifier should use a different variable name

$$\forall x \ [\exists y \ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists z \ Loves(z, x)]$$

4. Skolemize: Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

$$\forall x \ [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$$

Conversion to CNF contd.

3. Standardize variables: each quantifier should use a different variable name

$$\forall x \ [\exists y \ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists z \ Loves(z, x)]$$

4. Skolemize: Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

$$\forall x \ [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$$

5. Drop universal quantifiers:

$$[Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$$

Conversion to CNF contd.

3. Standardize variables: each quantifier should use a different variable name

$$\forall x \ [\exists y \ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists z \ Loves(z, x)]$$

4. Skolemize: Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

$$\forall x \ [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$$

5. Drop universal quantifiers:

$$[Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$$

6. Distribute \vee over \wedge :

$$[Animal(f(x)) \vee Loves(G(x), x)] \wedge [\neg Loves(x, F(x)) \vee Loves(G(x), x)]$$

Colonel West example using Resolution Refutation

S1: It is a crime for an American to sell weapons to hostile nations:

Colonel West example using Resolution Refutation

S1: It is a crime for an American to sell weapons to hostile nations:

$$\forall x, y, z \text{ } American(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$$

Colonel West example using Resolution Refutation

S1: It is a crime for an American to sell weapons to hostile nations:

$$\forall x, y, z \text{ } American(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$$

S2: Nono . . . has some missiles

Colonel West example using Resolution Refutation

S1: It is a crime for an American to sell weapons to hostile nations:

$$\forall x, y, z \text{ } American(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$$

S2: Nono . . . has some missiles

$$\exists x \text{ } \text{Owns}(\text{Nono}, x) \wedge \text{Missile}(x):$$

Colonel West example using Resolution Refutation

S1: It is a crime for an American to sell weapons to hostile nations:

$$\forall x, y, z \text{ } American(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$$

S2: Nono . . . has some missiles

$$\exists x \text{ } \text{Owns}(\text{Nono}, x) \wedge \text{Missile}(x):$$

S3: . . . all of its missiles were sold to it by Colonel West

Colonel West example using Resolution Refutation

S1: It is a crime for an American to sell weapons to hostile nations:

$$\forall x, y, z \text{ } American(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$$

S2: Nono . . . has some missiles

$$\exists x \text{ } \text{Owns}(\text{Nono}, x) \wedge \text{Missile}(x):$$

S3: . . . all of its missiles were sold to it by Colonel West

$$\forall x \text{ } \text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono})$$

Colonel West example using Resolution Refutation

S1: It is a crime for an American to sell weapons to hostile nations:

$$\forall x, y, z \text{ } American(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$$

S2: Nono . . . has some missiles

$$\exists x \text{ } \text{Owns}(\text{Nono}, x) \wedge \text{Missile}(x):$$

S3: . . . all of its missiles were sold to it by Colonel West

$$\forall x \text{ } \text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono})$$

S4: Missiles are weapons.

Colonel West example using Resolution Refutation

S1: It is a crime for an American to sell weapons to hostile nations:

$$\forall x, y, z \text{ } American(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$$

S2: Nono . . . has some missiles

$$\exists x \text{ } \text{Owns}(\text{Nono}, x) \wedge \text{Missile}(x):$$

S3: . . . all of its missiles were sold to it by Colonel West

$$\forall x \text{ } \text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono})$$

S4: Missiles are weapons.

$$\forall x \text{ } \text{Missile}(x) \Rightarrow \text{Weapon}(x)$$

Colonel West example using Resolution Refutation

S1: It is a crime for an American to sell weapons to hostile nations:

$$\forall x, y, z \text{ } American(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$$

S2: Nono . . . has some missiles

$$\exists x \text{ } \text{Owns}(\text{Nono}, x) \wedge \text{Missile}(x):$$

S3: . . . all of its missiles were sold to it by Colonel West

$$\forall x \text{ } \text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono})$$

S4: Missiles are weapons.

$$\forall x \text{ } \text{Missile}(x) \Rightarrow \text{Weapon}(x)$$

Colonel West example using Resolution Refutation

S1: It is a crime for an American to sell weapons to hostile nations:

$$\forall x, y, z \text{ } American(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$$

S2: Nono . . . has some missiles

$$\exists x \text{ } \text{Owns}(\text{Nono}, x) \wedge \text{Missile}(x):$$

S3: . . . all of its missiles were sold to it by Colonel West

$$\forall x \text{ } \text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono})$$

S4: Missiles are weapons.

$$\forall x \text{ } \text{Missile}(x) \Rightarrow \text{Weapon}(x)$$

S5: An enemy of America counts as “hostile”:

Colonel West example using Resolution Refutation

S1: It is a crime for an American to sell weapons to hostile nations:

$$\forall x, y, z \text{ } American(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$$

S2: Nono . . . has some missiles

$$\exists x \text{ } \text{Owns}(\text{Nono}, x) \wedge \text{Missile}(x):$$

S3: . . . all of its missiles were sold to it by Colonel West

$$\forall x \text{ } \text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono})$$

S4: Missiles are weapons.

$$\forall x \text{ } \text{Missile}(x) \Rightarrow \text{Weapon}(x)$$

S5: An enemy of America counts as “hostile”:

Colonel West example using Resolution Refutation

S1: It is a crime for an American to sell weapons to hostile nations:

$$\forall x, y, z \text{ } American(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$$

S2: Nono . . . has some missiles

$$\exists x \text{ } \text{Owns}(\text{Nono}, x) \wedge \text{Missile}(x):$$

S3: . . . all of its missiles were sold to it by Colonel West

$$\forall x \text{ } \text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono})$$

S4: Missiles are weapons.

$$\forall x \text{ } \text{Missile}(x) \Rightarrow \text{Weapon}(x)$$

S5: An enemy of America counts as “hostile”:

S6: West, who is American . . .

$$S6: \text{American}(\text{West})$$

S7: The country Nono, an enemy of America . . .

$$S7: \text{Enemy}(\text{Nono}, \text{America})$$

Colonel West example using Resolution Refutation

- KB:

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \implies Criminal(x)$

$Owns(Nono, M_1)$ and $Missile(M_1)$

$Missile(x) \Rightarrow Weapon(x)$

$Missile(x) \wedge Owns(Nono, x) \implies Sells(West, x, Nono)$

$Enemy(x, America) \implies Hostile(x)$

$American(West)$

$Enemy(Nono, America)$

- Question: Is colonel West criminal?

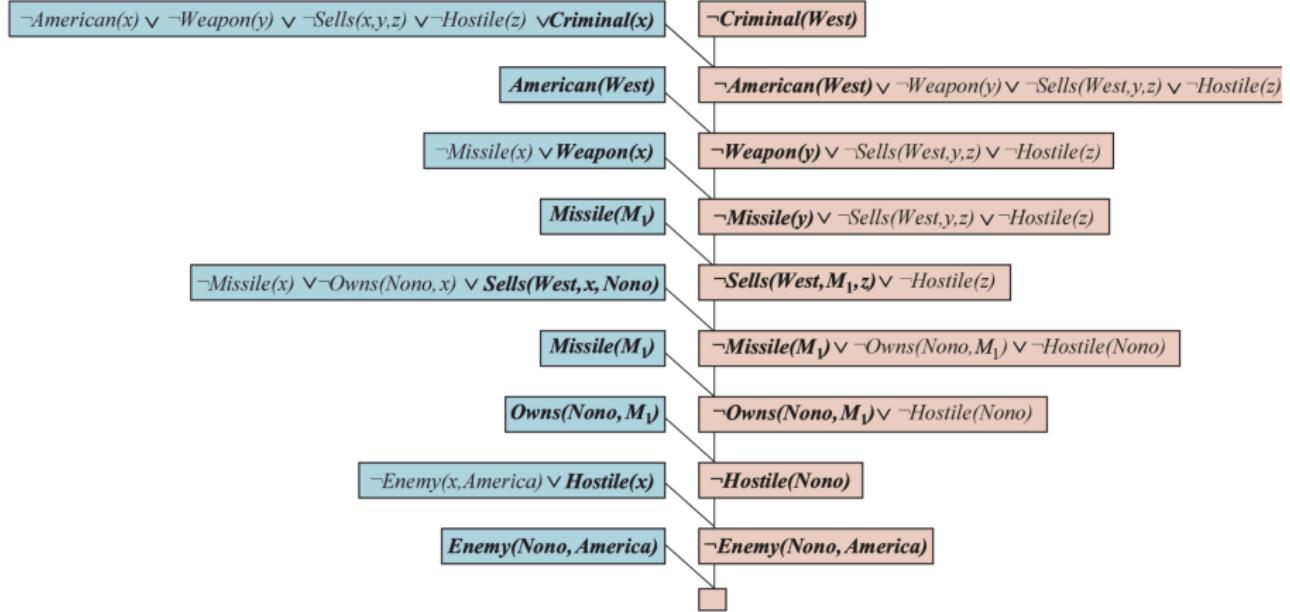
Colonel West example in CNF form

- ① $\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x, y, z) \vee \neg Hostile(z) \vee Criminal(x)$
- ② $\neg Missile(x) \vee \neg Owns(Nono, x) \vee Sells(West, x, Nono)$
- ③ $\neg Enemy(x, America) \vee Hostile(x)$
- ④ $\neg Missile(x) \vee Weapon(x)$

Colonel West example in CNF form

- ① $\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x, y, z) \vee \neg Hostile(z) \vee Criminal(x)$
 - ② $\neg Missile(x) \vee \neg Owns(Nono, x) \vee Sells(West, x, Nono)$
 - ③ $\neg Enemy(x, America) \vee Hostile(x)$
 - ④ $\neg Missile(x) \vee Weapon(x)$
-
- Facts:
 $American(West), Owns(Nono, M1), Missile(M1), Enemy(Nono, America)$
 - Add $\neg Criminal(West)$

Colonel West example using Resolution Refutation - cont



Efficient Resolution Strategies

- Unit preference: prefer inferences where one of the sentences is a single literal - towards shorter clauses, finally empty one
- Set of support: start with the negated query, and add resolvents into the set. In each inference use one of the sentences from this set - search space is reduced.
- Input resolution: In each inference use one of the sentences in the original KB or query, and one other sentence (i.e., a resolvent).

Forward chaining algorithm

- Employs (repeatedly) generalized Modus ponens - which is sound
- Continue matching and deriving consequences of implication sentences until deriving the goal.
- FC is complete for definite clause KBs without functions - i.e. finds an answer to each query of which answer is entailed by the KB

Inference through Forward Chaining in FOL

- Forward Chaining is an important inference in FOL - without propositionalisation
- FC uses definite clauses.
 - A definite clause: either atomic, or implication
 - Existential quantifiers are not allowed
 - Universal quantifications are implicit.

Inference through Forward Chaining in FOL

- Forward Chaining is an important inference in FOL - without propositionalisation
- FC uses definite clauses.
 - A definite clause: either atomic, or implication
 - Existential quantifiers are not allowed
 - Universal quantifications are implicit.
 - Example sentences in DC:

King (John) - literal

$\text{King}(x) \rightarrow \text{Evil}(x)$

$\text{Evil}(x)$. (i.e., everyone is evil). - literal with variable

Forward Chaining - Colonel West example

- KB: The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

We want to prove that "**Colonel West is a criminal**".

- We need
 - to translate these natural language sentences to FOL

Forward Chaining - Colonel West example

- KB: The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

We want to prove that "**Colonel West is a criminal**".

- We need
 - to translate these natural language sentences to FOL
 - convert these sentences in the KB to Definite Clauses if they are not in DC form by eliminating quantifiers:

Forward Chaining - Colonel West example

- KB: The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

We want to prove that "**Colonel West is a criminal**".

- We need
 - to translate these natural language sentences to FOL
 - convert these sentences in the KB to Definite Clauses if they are not in DC form by eliminating quantifiers:
 - remove \forall
 - eliminate \exists by skolemization

Forward Chaining - Colonel West example

S1: It is a crime for an American to sell weapons to hostile nations:

Forward Chaining - Colonel West example

S1: It is a crime for an American to sell weapons to hostile nations:

$$\forall x, y, z \text{ } American(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$$

Forward Chaining - Colonel West example

S1: It is a crime for an American to sell weapons to hostile nations:

$\forall x, y, z \text{ } American(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$

S1: $American(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$

Forward chaining - Colonel West

... It is a crime for an American to sell weapons to hostile nations:

S1: $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \implies Criminal(x)$

S2: Nono ... has some missiles

Forward chaining - Colonel West

... It is a crime for an American to sell weapons to hostile nations:

S1: $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \implies Criminal(x)$

S2: Nono ... has some missiles

$\exists x \ Owns(Nono, x) \wedge Missile(x)$:

Forward chaining - Colonel West

... It is a crime for an American to sell weapons to hostile nations:

S1: $\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$

S2: Nono ... has some missiles

$\exists x \text{ } \text{Owns}(\text{Nono}, x) \wedge \text{Missile}(x)$:

S2: $\text{Owns}(\text{Nono}, M_1)$ and $\text{Missile}(M_1)$

Forward chaining - Colonel West

S3: ... all of its missiles were sold to it by Colonel West

Forward chaining - Colonel West

S3: ... all of its missiles were sold to it by Colonel West

$$\forall x \text{ } Missile(x) \wedge Owns(Nono, x) \implies Sells(West, x, Nono)$$

Forward chaining - Colonel West

S3: ... all of its missiles were sold to it by Colonel West

$\forall x \text{ } Missile(x) \wedge \text{Owns}(Nono, x) \implies \text{Sells}(West, x, Nono)$

S3: $Missile(x) \wedge \text{Owns}(Nono, x) \implies \text{Sells}(West, x, Nono)$

Forward chaining - Colonel West

S4: Missiles are weapons.

Forward chaining - Colonel West

S4: Missiles are weapons.

$$\forall x \text{ } \textit{Missile}(x) \Rightarrow \textit{Weapon}(x)$$

Forward chaining - Colonel West

S4: Missiles are weapons.

$$\forall x \text{ } \textit{Missile}(x) \Rightarrow \textit{Weapon}(x)$$

S4: *Missile(x) ⇒ Weapon(x)*

S5: An enemy of America counts as “hostile”:

Forward chaining - Colonel West

S4: Missiles are weapons.

$$\forall x \text{ } \textit{Missile}(x) \Rightarrow \textit{Weapon}(x)$$

S4: *Missile(x) ⇒ Weapon(x)*

S5: An enemy of America counts as “hostile”:

$$\forall x \text{ } \textit{Enemy}(x, \textit{America}) \implies \textit{Hostile}(x)$$

Forward chaining - Colonel West

S4: Missiles are weapons.

$$\forall x \text{ } \textit{Missile}(x) \Rightarrow \textit{Weapon}(x)$$

S4: *Missile(x) ⇒ Weapon(x)*

S5: An enemy of America counts as “hostile”:

$$\forall x \text{ } \textit{Enemy}(x, \textit{America}) \implies \textit{Hostile}(x)$$

S5: *Enemy(x, America) ⇒ Hostile(x)*

Forward chaining - Colonel West

S6: West, who is American ...

S6: *American(West)*

S7: The country Nono, an enemy of America ...

S7: *Enemy(Nono, America)*

Forward chaining - Colonel West

Our KB:

- ① $\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$
- ② $\text{Owns}(\text{Nono}, M_1)$
- ③ $\text{Missile}(M_1)$
- ④ $\text{Missile}(x) \Rightarrow \text{Weapon}(x)$
- ⑤ $\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono})$
- ⑥ $\text{Enemy}(x, \text{America}) \implies \text{Hostile}(x)$
- ⑦ $\text{American}(\text{West})$
- ⑧ $\text{Enemy}(\text{Nono}, \text{America})$

Forward chaining proof of Colonel West

American(West)

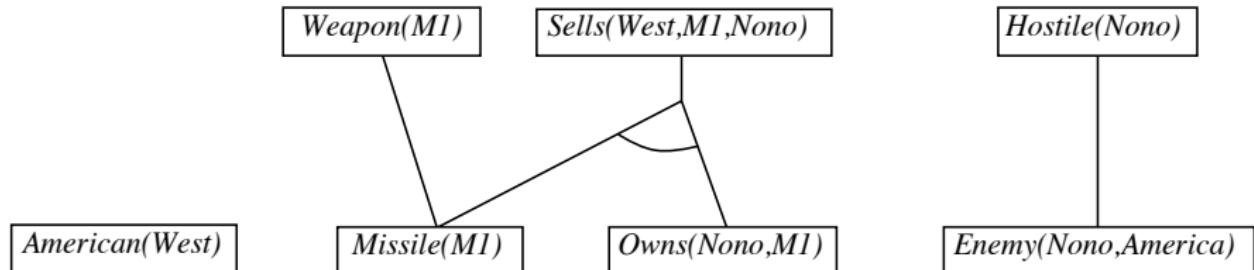
Missile(M1)

Owns(Nono,M1)

Enemy(Nono,America)

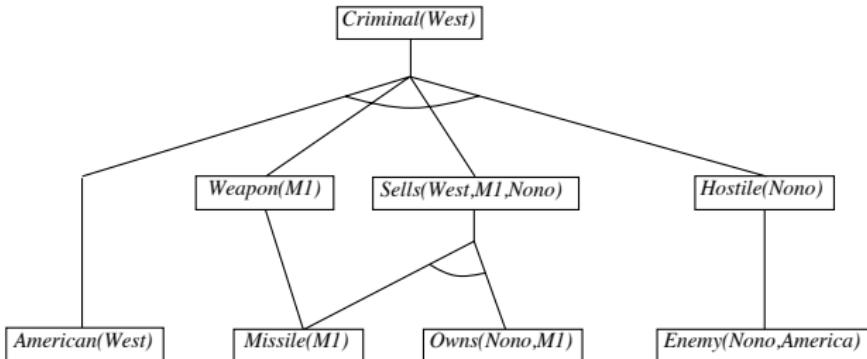
- ① $\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$
- ② $\text{Owns}(\text{Nono}, M_1)$
- ③ $\text{Missile}(M_1)$
- ④ $\text{Missile}(x) \Rightarrow \text{Weapon}(x)$
- ⑤ $\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono})$
- ⑥ $\text{Enemy}(x, \text{America}) \implies \text{Hostile}(x)$
- ⑦ $\text{American}(\text{West})$
- ⑧ $\text{Enemy}(\text{Nono}, \text{America})$

Forward chaining proof



- ① $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \implies Criminal(x)$
- ② $Owns(Nono, M_1)$
- ③ $Missile(M_1)$
- ④ $Missile(x) \Rightarrow Weapon(x)$
- ⑤ $Missile(x) \wedge Owns(Nono, x) \implies Sells(West, x, Nono)$
- ⑥ $Enemy(x, America) \implies Hostile(x)$
- ⑦ $American(West)$
- ⑧ $Enemy(Nono, America)$

Forward chaining proof



- ① $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \implies Criminal(x)$
- ② $Owes(Nono, M_1)$
- ③ $Missile(M_1)$
- ④ $Missile(x) \Rightarrow Weapon(x)$
- ⑤ $Missile(x) \wedge Owes(Nono, x) \implies Sells(West, x, Nono)$
- ⑥ $Enemy(x, America) \implies Hostile(x)$
- ⑦ $American(West)$
- ⑧ $Enemy(Nono, America)$

Forward Chaining and efficiency

- checks every rule against every fact

$$\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono})$$

Heuristic: Order the conjuncts to be checked in the premise. Check the one with fewest fact sentence in the KB first - reminds the *Minimum remaining values* heuristic in CSP.

- rechecks every rule in every iteration

e.g., eliminate redundant rule-matching through incremental FC: Check a rule at iteration t if only its premise includes a conjunct P1 that unifies with a fact $P1'$ inferred at iteration t-1

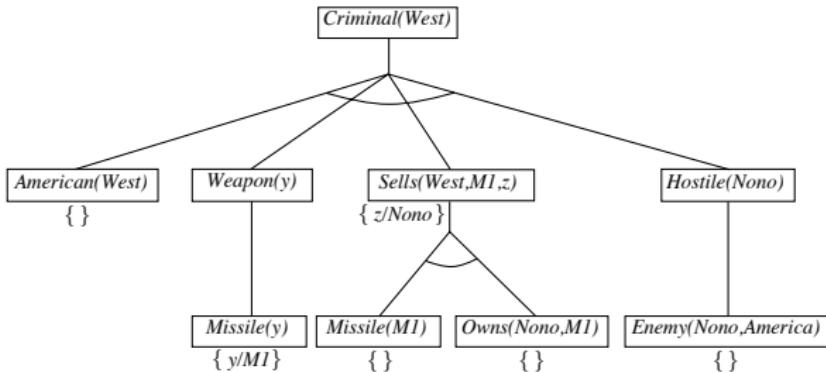
- can generate many facts irrelevant to the goal

e.g., do Backward Chaining instead

Backward chaining example

- Employs depth-first search
- BC is incomplete as it may have the problem of repeated states.

Backward chaining example

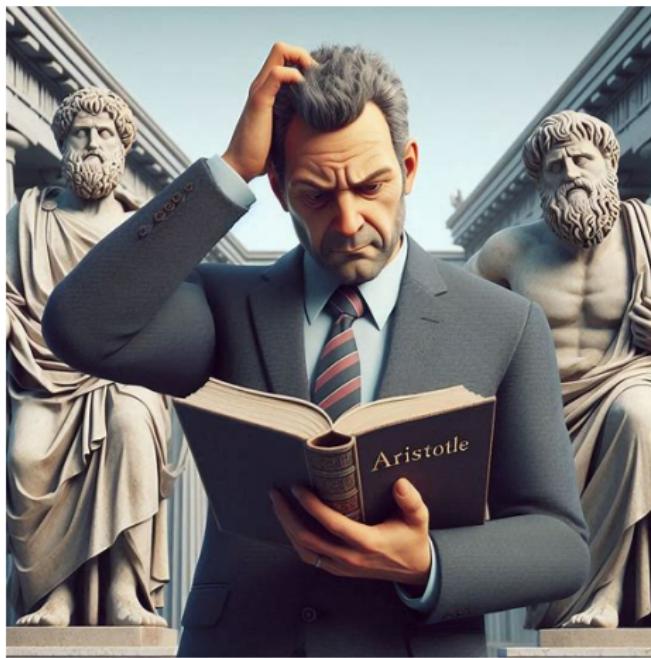


- ① $\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \implies \text{Criminal}(x)$
- ② $\text{Owns}(\text{Nono}, M_1)$
- ③ $\text{Missile}(M_1)$
- ④ $\text{Missile}(x) \Rightarrow \text{Weapon}(x)$
- ⑤ $\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \implies \text{Sells}(\text{West}, x, \text{Nono})$
- ⑥ $\text{Enemy}(x, \text{America}) \implies \text{Hostile}(x)$
- ⑦ $\text{American}(\text{West})$

More on Skolemization -Examples

- $\exists x \text{ Sibling}(\text{sofie}, x)$
Skolemized: $\text{Sibling}(\text{Sofie}, \text{SkSister})$
- $\forall x \exists y \text{ Parent}(x, y)$
Skolemized: $\forall x \text{ Parent}(x, F(x))$
- $\forall x, y \text{ Grandpa}(x, y) \implies \exists z \text{ Parent}(x, z) \wedge \text{Parent}(z, y)$
Skolemized: $\forall x, y \text{ Grandpa}(x, y) \implies \text{Parent}(x, F(x, y)) \wedge \text{Parent}(F(x, y), y)$
- $\forall x [\exists y \text{ Animal}(y) \wedge \neg \text{Loves}(x, y)] \vee [\exists z \text{ Loves}(z, x)]$
Skolemized: $\forall x [\text{Animal}(F(x)) \wedge \neg \text{Loves}(x, F(x))] \vee \text{Loves}(G(x), x)$

Logic for Politicians



Propositional Logic

The Knowledge Base

$$(D \wedge G) \implies (I \vee W)$$

$$(W \vee C) \implies R$$

$$I \implies C$$

$$D \vee R$$

The Knowledge Base in Conjunctive Normal Form

$$\neg D \vee \neg G \vee I \vee W$$

$$\neg W \vee R$$

$$\neg C \vee R$$

$$\neg I \vee C$$

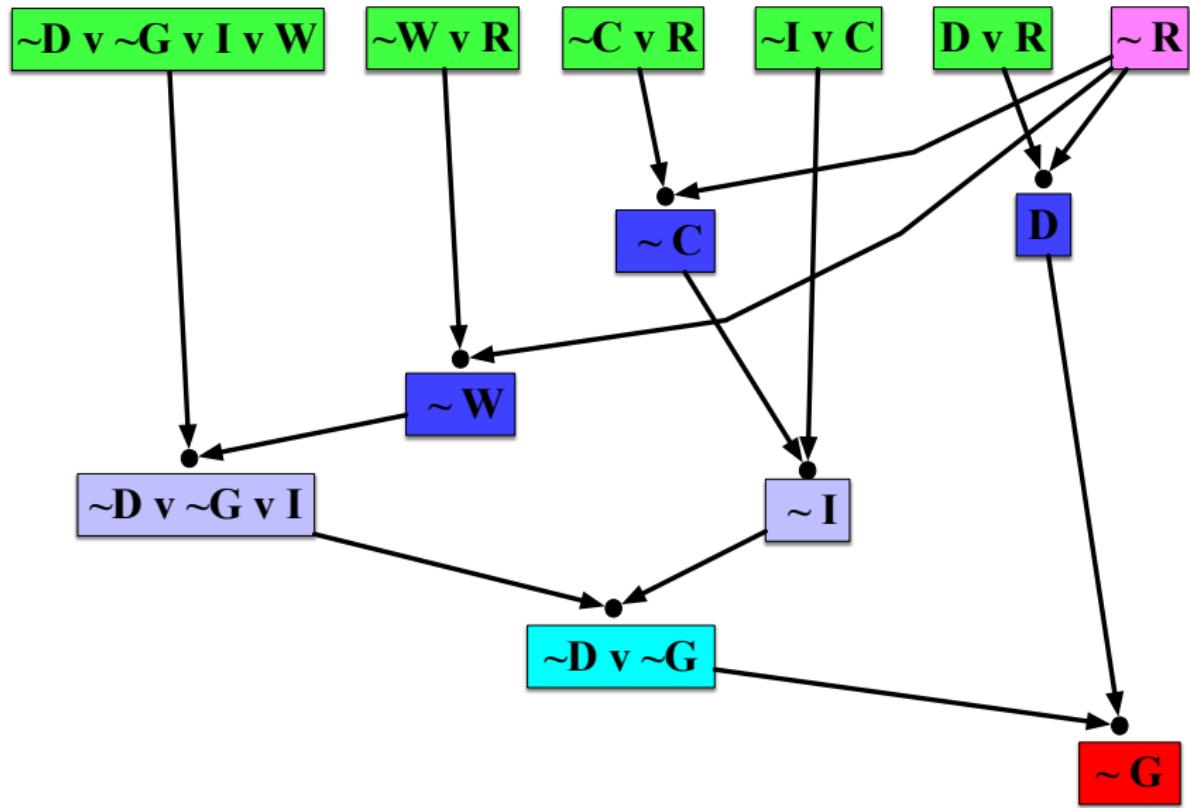
$$D \vee R$$

Problem

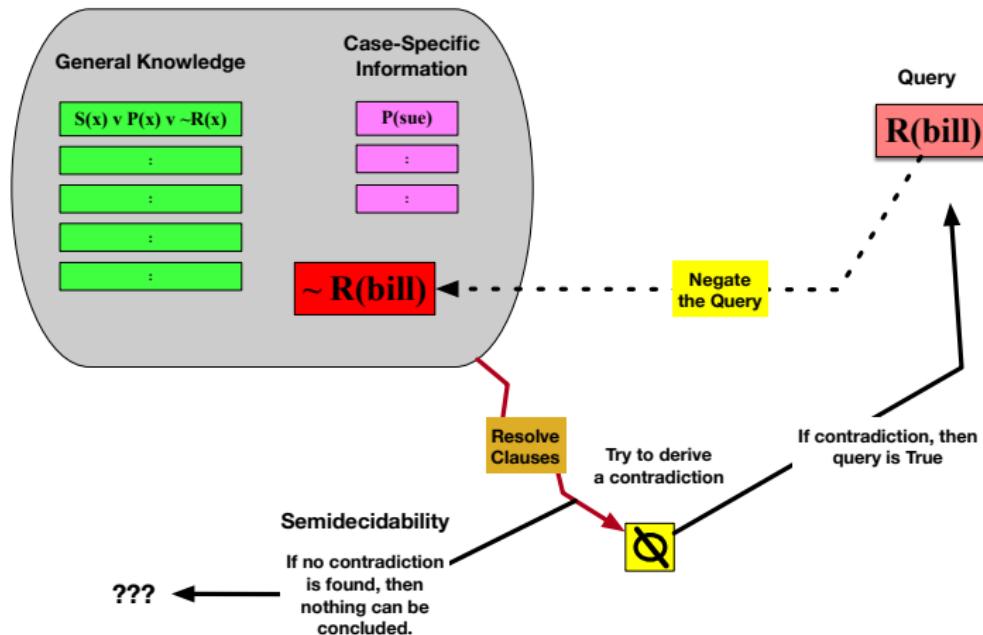
Given: $\neg R$

Prove: $\neg G$

Resolving Clauses in Propositional Logic



The Basic Resolution Algorithm

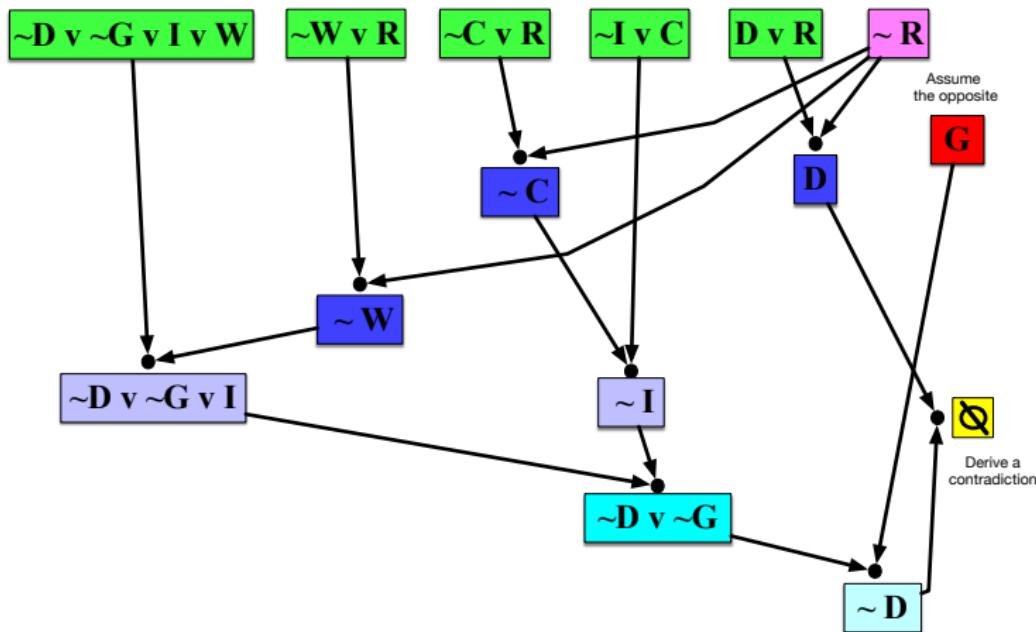


* Propositional logic is decidable (Just use model checking), but first-order logic is only semidecidable \Rightarrow In FOL, failure to find a contradiction does NOT prove that the query is False, but any sound and complete inference procedure (such as resolution) will eventually prove a true query.

The Resolution Algorithm for Propositional Logic

To Prove: $\neg G$

Assume: $\neg\neg G \equiv G$



Conjunctive Normal Form for FOL

A candidate wins a debate only if they appear wise on one issue and their opponent appears confused on another issue.

$$\forall x, y : [\text{Debate}(x, y) \wedge \text{Wins}(x, y)] \implies [\exists w : \text{Wise}(x, w) \wedge \exists w : \text{Confused}(y, w)]$$

- ➊ Eliminate implications

$$\forall x, y : \neg[\text{Debate}(x, y) \wedge \text{Wins}(x, y)] \vee [\exists w : \text{Wise}(x, w) \wedge \exists w : \text{Confused}(y, w)]$$

- ➋ Move \neg inwards.

$$\forall x, y : \neg\text{Debate}(x, y) \vee \neg\text{Wins}(x, y) \vee [\exists w : \text{Wise}(x, w) \wedge \exists w : \text{Confused}(y, w)]$$

- ➌ Standardize Variables

$$\forall x, y : \neg\text{Debate}(x, y) \vee \neg\text{Wins}(x, y) \vee [\exists w : \text{Wise}(x, w) \wedge \exists z : \text{Confused}(y, z)]$$

- ➍ Skolemize (Drop existential quantifiers and replace their variables with constants or functions of the scoping universally-quantified variables.)

$$\forall x, y : \neg\text{Debate}(x, y) \vee \neg\text{Wins}(x, y) \vee [\text{Wise}(x, F(x, y)) \wedge \text{Confused}(y, G(x, y))]$$

- ➎ Drop Universal Quantifiers

$$\neg\text{Debate}(x, y) \vee \neg\text{Wins}(x, y) \vee [\text{Wise}(x, F(x, y)) \wedge \text{Confused}(y, G(x, y))]$$

- ➏ Distribute \vee over \wedge

$$[\neg\text{Debate}(x, y) \vee \neg\text{Wins}(x, y) \vee \text{Wise}(x, F(x, y))] \wedge [\neg\text{Debate}(x, y) \vee \neg\text{Wins}(x, y) \vee \text{Confused}(y, G(x, y))]$$

FOL Conjunctive Normal Form: Exercise

There is at least one problem that any pair of candidates agree is important, but they never agree on any of its solutions.

$$\exists w : \text{Problem}(w) \wedge \forall x, y : [\text{Candidate}(x) \wedge \text{Candidate}(y)] \implies [\text{Agree}(x, y, w) \wedge [\forall z : \text{Solution}(z, w) \implies \neg \text{Agree}(x, y, z)]]$$

$$\exists w : P(w) \wedge \forall x, y : [C(x) \wedge C(y)] \implies [A(x, y, w) \wedge [\forall z : S(z, w) \implies \neg A(x, y, z)]]$$

- ➊ Eliminate implications
- ➋ Move \neg inwards.
- ➌ Standardize Variables
- ➍ Skolemize (Drop existential quantifiers and replace their variables with constants or functions of the scoping universally-quantified variables.)
- ➎ Drop Universal Quantifiers
- ➏ Distribute \vee over \wedge

First-Order Logic (FOL) for Politics

The Knowledge Base

- $\forall x : (\text{Democrat}(x) \wedge \text{Generous}(x)) \implies (\text{Idealist}(x) \vee \text{Wealthy}(x))$
- $\forall x : (\text{Wealthy}(x) \vee \text{Credulous}(x)) \implies \text{Republican}(x)$
- $\forall x : \text{Idealist}(x) \implies \text{Credulous}(x)$
- $\forall x : \text{Democrat}(x) \vee \text{Republican}(x)$

The Knowledge Base in Conjunctive Normal Form

Note: variables renamed to avoid conflicts

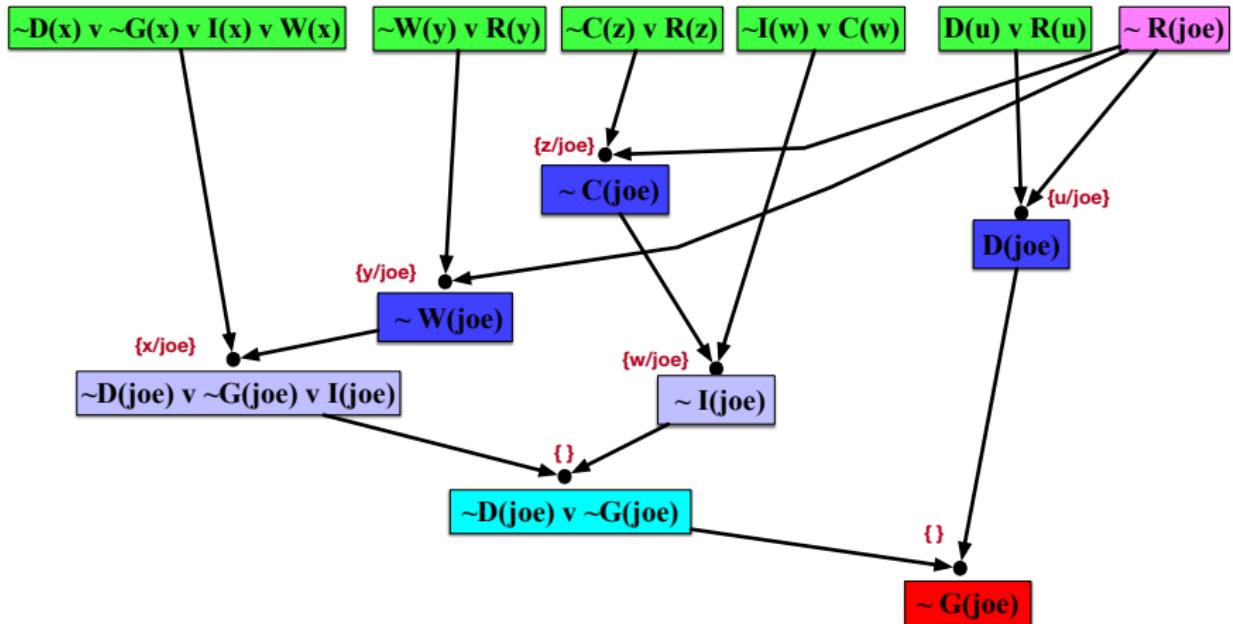
- $\neg \text{Democrat}(x) \vee \neg \text{Generous}(x) \vee \text{Idealist}(x) \vee \text{Wealthy}(x)$
- $\neg \text{Wealthy}(y) \vee \text{Republican}(y)$
- $\neg \text{Credulous}(z) \vee \text{Republican}(z)$
- $\neg \text{Idealist}(w) \vee \text{Credulous}(w)$
- $\text{Democrat}(u) \vee \text{Republican}(u)$

Problem

Given: $\neg \text{Republican}(\text{joe})$

Prove: $\neg \text{Generous}(\text{joe})$

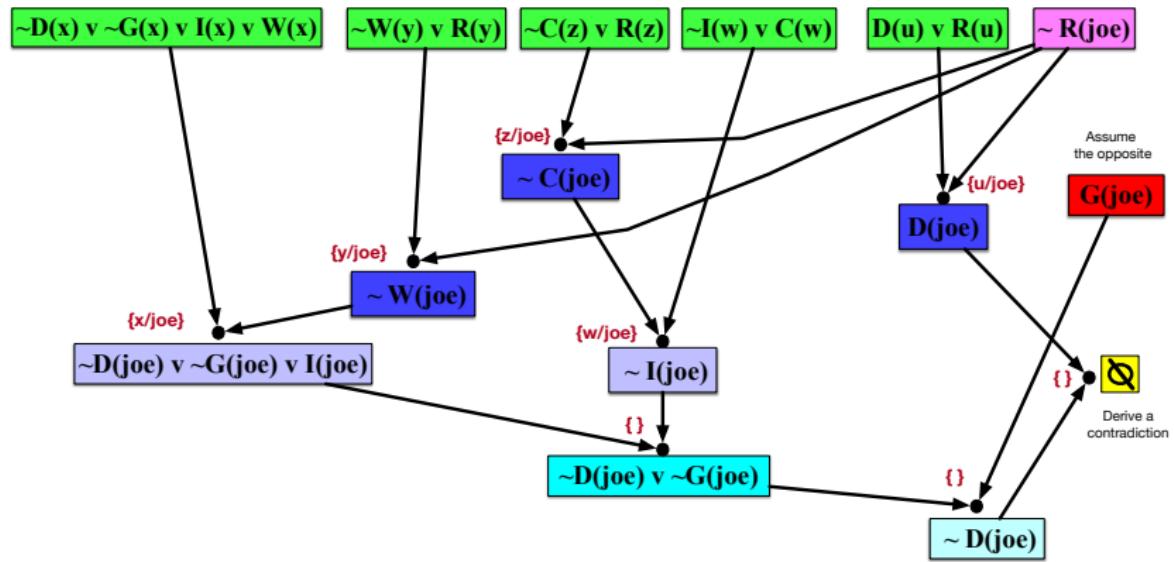
Resolving Clauses for Political FOL



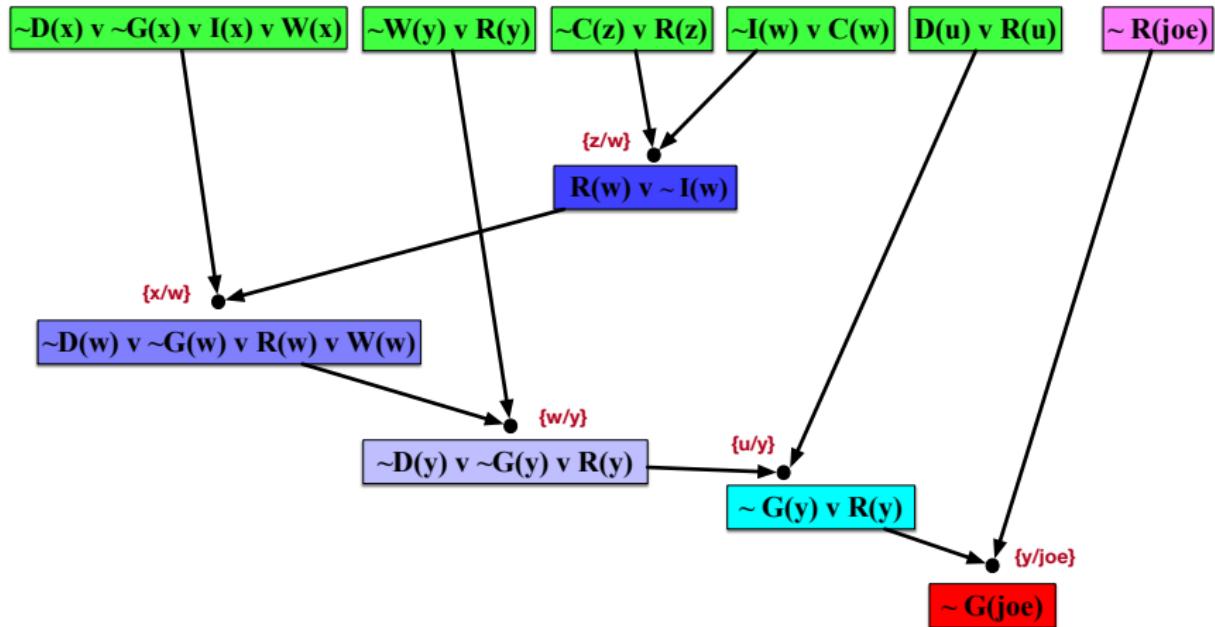
The Resolution Algorithm for Political FOL

To Prove: $\neg G(joe)$

Assume: $\neg\neg G(joe) \equiv G(joe)$



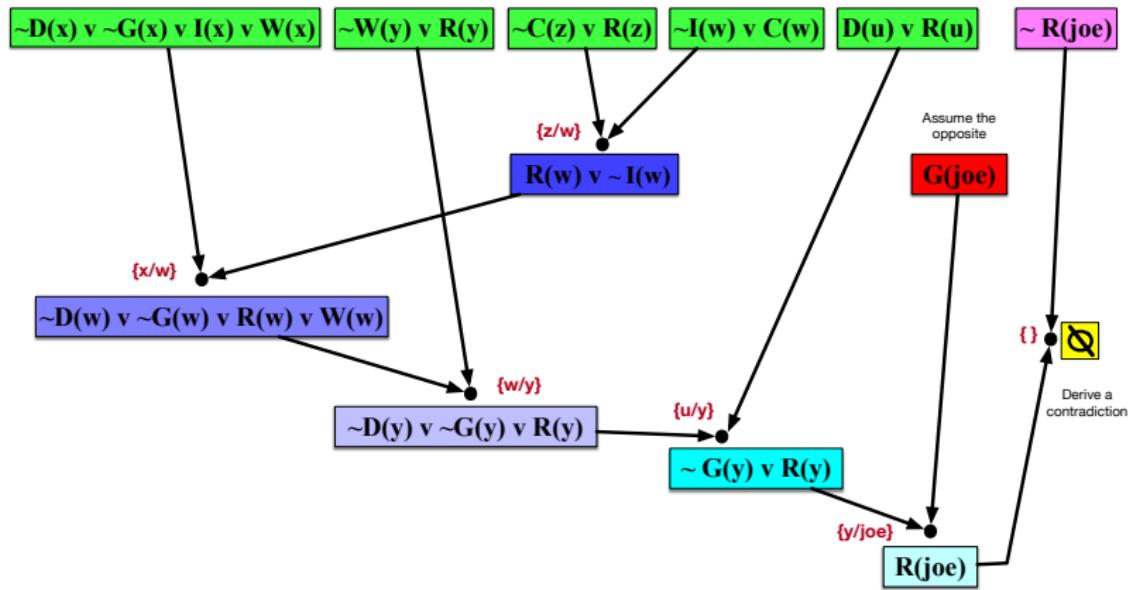
An Alternate Resolution Sequence



An Alternate Run of the Resolution Algorithm

To Prove: $\neg G(joe)$

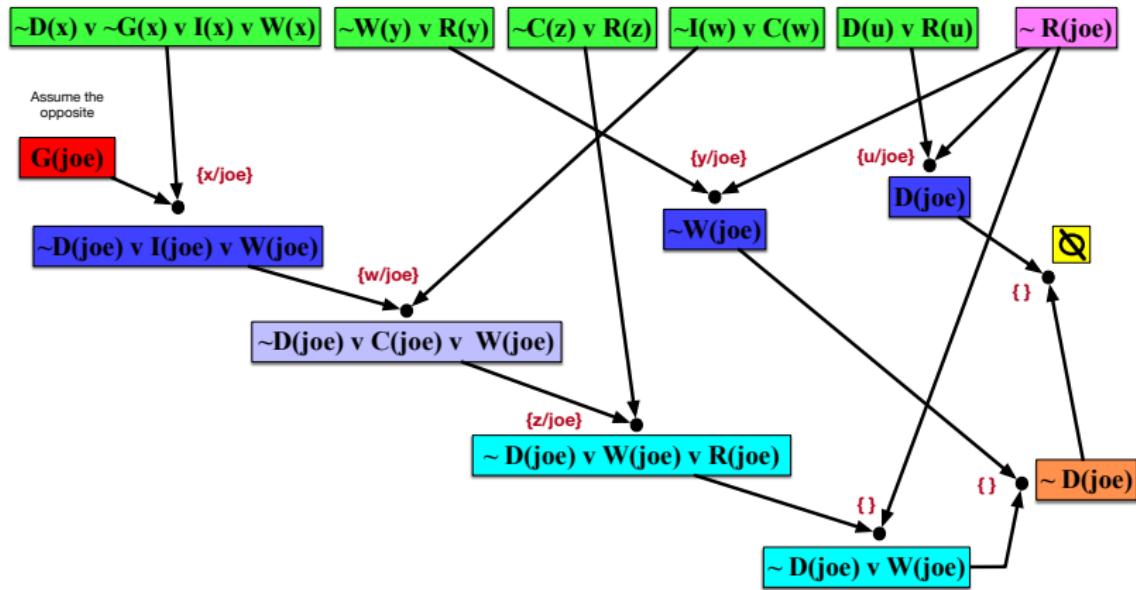
Assume: $\neg\neg G(joe) \equiv G(joe)$



...and Another Alternate

To Prove: $\neg G(joe)$

Assume: $\neg\neg G(joe) \equiv G(joe)$



FOL with Existential Quantification

The Knowledge Base

- $\forall x : (\text{Democrat}(x) \wedge \text{Generous}(x)) \implies (\text{Idealist}(x) \vee \text{Wealthy}(x))$
- $\forall x : (\text{Wealthy}(x) \vee \text{Credulous}(x)) \implies \text{Republican}(x)$
- $\forall x : \text{Idealist}(x) \implies \text{Credulous}(x)$
- $\forall x : \text{Democrat}(x) \vee \text{Republican}(x)$
- $\forall x : \text{Generous}(x) \iff \exists y : \text{Supports}(x, y)$

The Knowledge Base in Conjunctive Normal Form

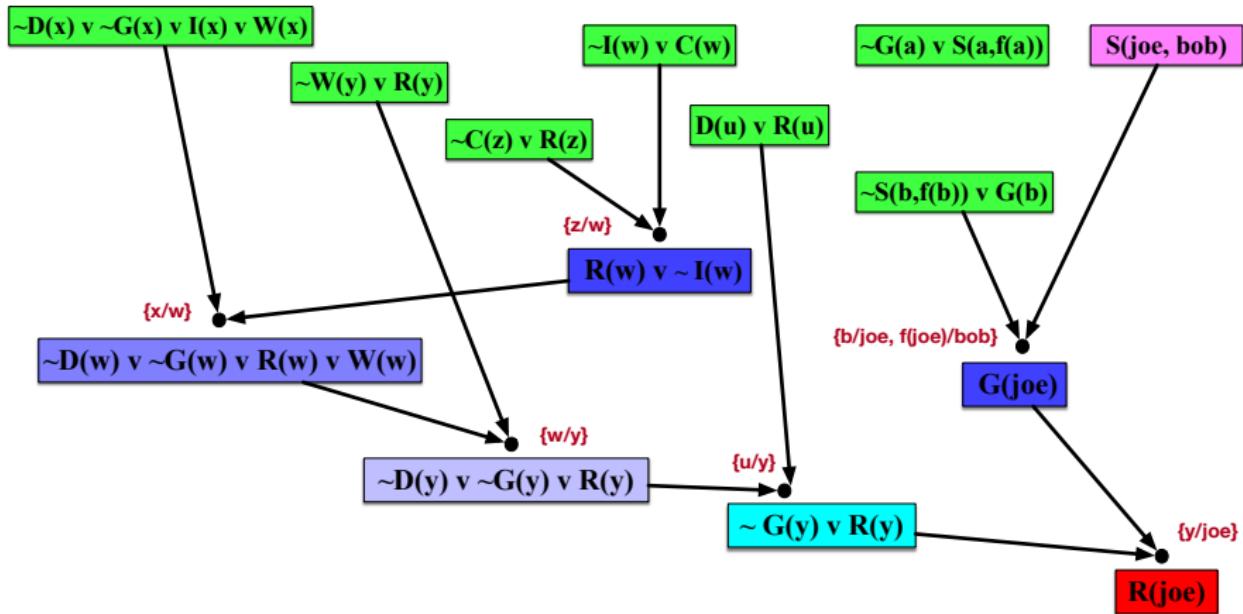
Note: variables renamed to avoid conflicts

- $\neg \text{Democrat}(x) \vee \neg \text{Generous}(x) \vee \text{Idealist}(x) \vee \text{Wealthy}(x)$
- $\neg \text{Wealthy}(y) \vee \text{Republican}(y)$
- $\neg \text{Credulous}(z) \vee \text{Republican}(z)$
- $\neg \text{Idealist}(w) \vee \text{Credulous}(w)$
- $\text{Democrat}(u) \vee \text{Republican}(u)$
- $\neg \text{Generous}(a) \vee \text{Supports}(a, f(a))$ f() = skolem function
- $\neg \text{Supports}(b, f(b)) \vee \text{Generous}(b)$

Resolution with Existential Quantification

Given: $\text{Supports}(joe, bob)$

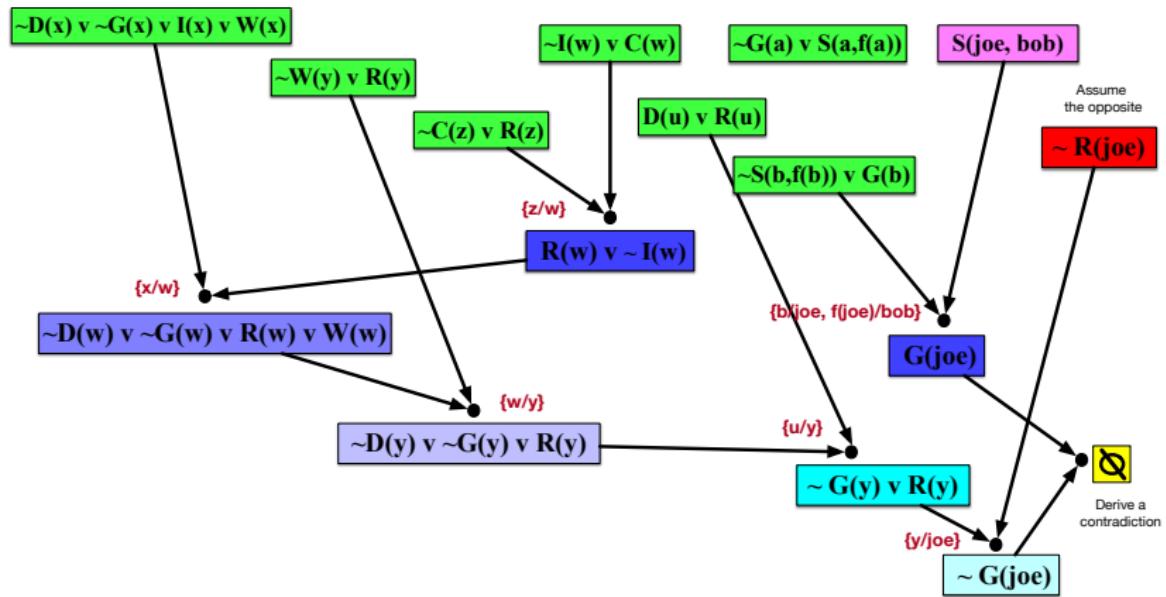
Prove: $\text{Republican}(joe)$



Resolution Algorithm with Existential Quantification

To Prove: *Republican(joe)*

Assume: $\neg \text{Republican}(joe)$



Forward Chaining Politics

The previous kwg base is not amenable to forward chaining, because:

- It involves an existential quantifier, but forward-chaining only deals with universal quantification.
- Several of the disjunctions are not *definite clauses*, since they have more than one positive literal.
- Let's try a different knowledge base.

Kwg Base

$\forall x, y : Democrat(x) \wedge Tax(y) \implies Likes(x, y)$

$\forall x, y, z : Likes(x, z) \wedge Likes(y, z) \implies Likes(x, y)$

$\forall x, y, z : Republican(x) \wedge Republican(y) \wedge Democrat(z) \wedge Likes(y, z)$

$\implies Dislikes(x, y)$

Democrat(kamala)

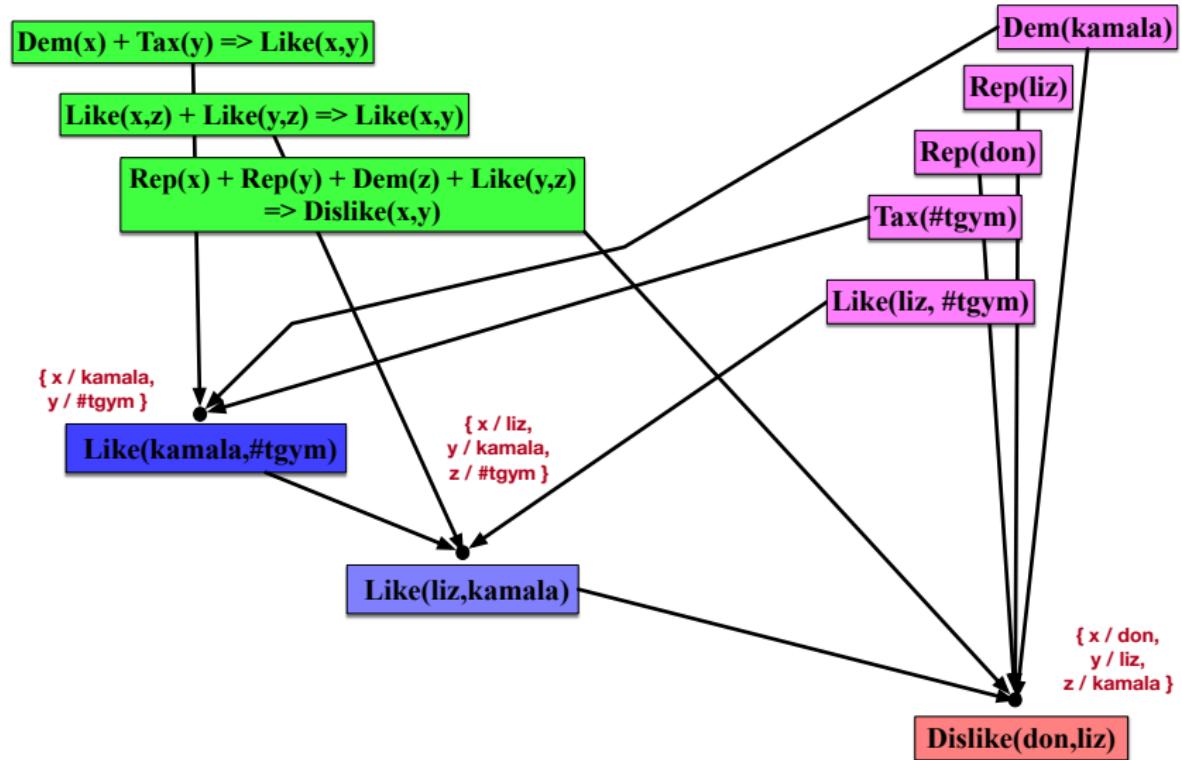
Republican(liz)

Republican(don)

Tax(#tgym) #tgym = training studio tax

Likes(Liz, #tgym)

The Logic of Polarization



Appendix: CNF Exercise Solution

There is at least one problem that any pair of candidates agree is important, but they never agree on any of its solutions.

$$\exists w : \text{Problem}(w) \wedge \forall x, y : [\text{Candidate}(x) \wedge \text{Candidate}(y)] \implies [\text{Agree}(x, y, w) \wedge [\forall z : \text{Solution}(z, w) \implies \neg \text{Agree}(x, y, z)]]$$

$$\exists w : P(w) \wedge \forall x, y : [C(x) \wedge C(y)] \implies [A(x, y, w) \wedge [\forall z : S(z, w) \implies \neg A(x, y, z)]]$$

- ① Eliminate implications

$$\exists w : P(w) \wedge \forall x, y : \neg [C(x) \wedge C(y)] \vee [A(x, y, w) \wedge [\forall z : \neg S(z, w) \vee \neg A(x, y, z)]]$$

- ② Move \neg inwards.

$$\exists w : P(w) \wedge \forall x, y : \neg C(x) \vee \neg C(y) \vee [A(x, y, w) \wedge [\forall z : \neg S(z, w) \vee \neg A(x, y, z)]]$$

- ③ Standardize Variables (not necessary here)

$$\exists w : P(w) \wedge \forall x, y : \neg C(x) \vee \neg C(y) \vee [A(x, y, w) \wedge [\forall z : \neg S(z, w) \vee \neg A(x, y, z)]]$$

- ④ Skolemize (K = skolem constant)

$$P(K) \wedge \forall x, y : \neg C(x) \vee \neg C(y) \vee [A(x, y, K) \wedge [\forall z : \neg S(z, K) \vee \neg A(x, y, z)]]$$

- ⑤ Drop Universal Quantifiers

$$P(K) \wedge [\neg C(x) \vee \neg C(y) \vee [A(x, y, K) \wedge [\neg S(z, K) \vee \neg A(x, y, z)]]]$$

- ⑥ Distribute \vee over \wedge (Yielding 3 conjuncts)

$$P(K) \wedge$$

$$[\neg C(x) \vee \neg C(y) \vee A(x, y, K)] \wedge$$

$$[\neg C(x) \vee \neg C(y) \vee \neg S(z, K) \vee \neg A(x, y, z)]$$

* Note: No skolem **function** was needed, since w was not **declared** within the scope of any universal variables, though it was used within their scope.

Summary

Let us summarize together!

Adversarial search and games

Ole C. Eidheim

October 10, 2024

Department of Computer Science

Example games

	Deterministic	Chance (external)
Perfect information		
Imperfect information		

Example games

	Deterministic	Chance (external)
Perfect information	Chess, Go	Backgammon, Monopoly
Imperfect information	Battleship, Rock paper scissors	Poker, Scrabble

Outline

A game

Optimal decisions in games

Cutting off search

Monte Carlo tree search

Stochastic games

Partially observable games

Turn-taking, perfect information games

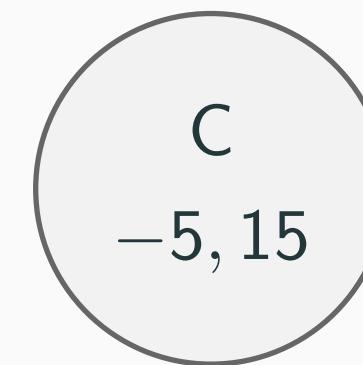
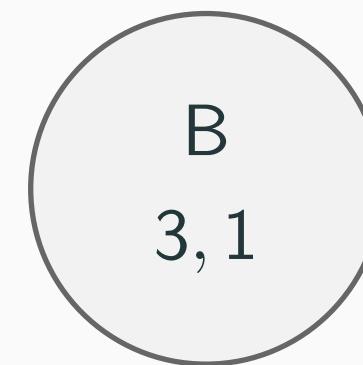
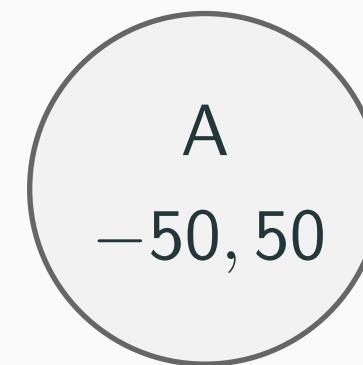
Definitions

- initialState : starting state, including which player's turn it is
- $\text{toMove}(s)$: which player's turn it is at state s
- $\text{actions}(s)$: legal actions at state s
- $\text{result}(s, a)$: next state after taking action a at state s
- $\text{isTerminal}(s)$: is state s an end state?
- $\text{utility}(s, p)$: the value of the terminal state s for player p

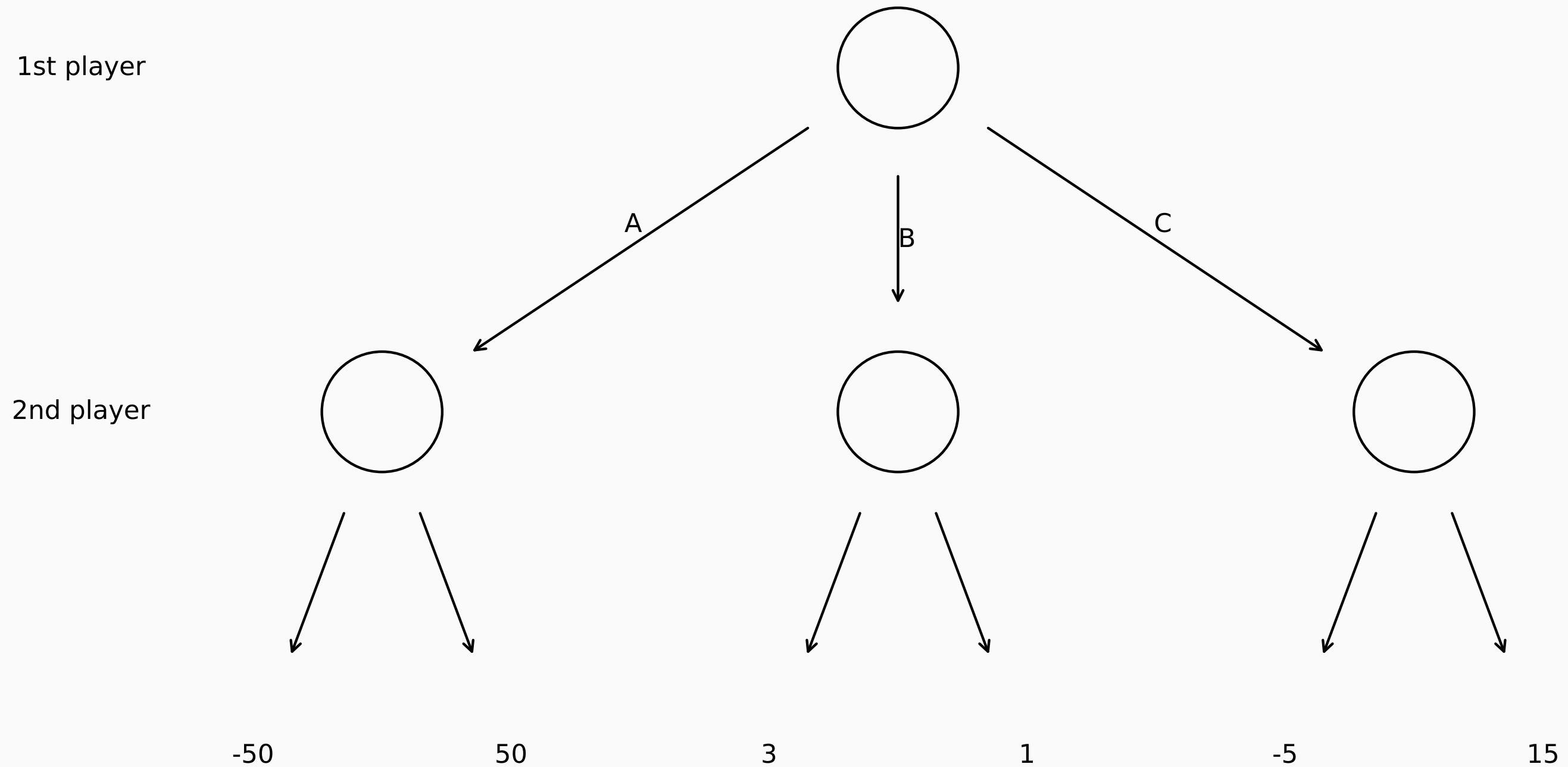
Objective: find player p 's actions from initialState to a terminal state s that maximize $\text{utility}(s, p)$

Example: the bucket game

- The first player chooses bucket A, B or C
- The second player selects a number from the chosen bucket
- The goal of the first player is to maximize the selected number



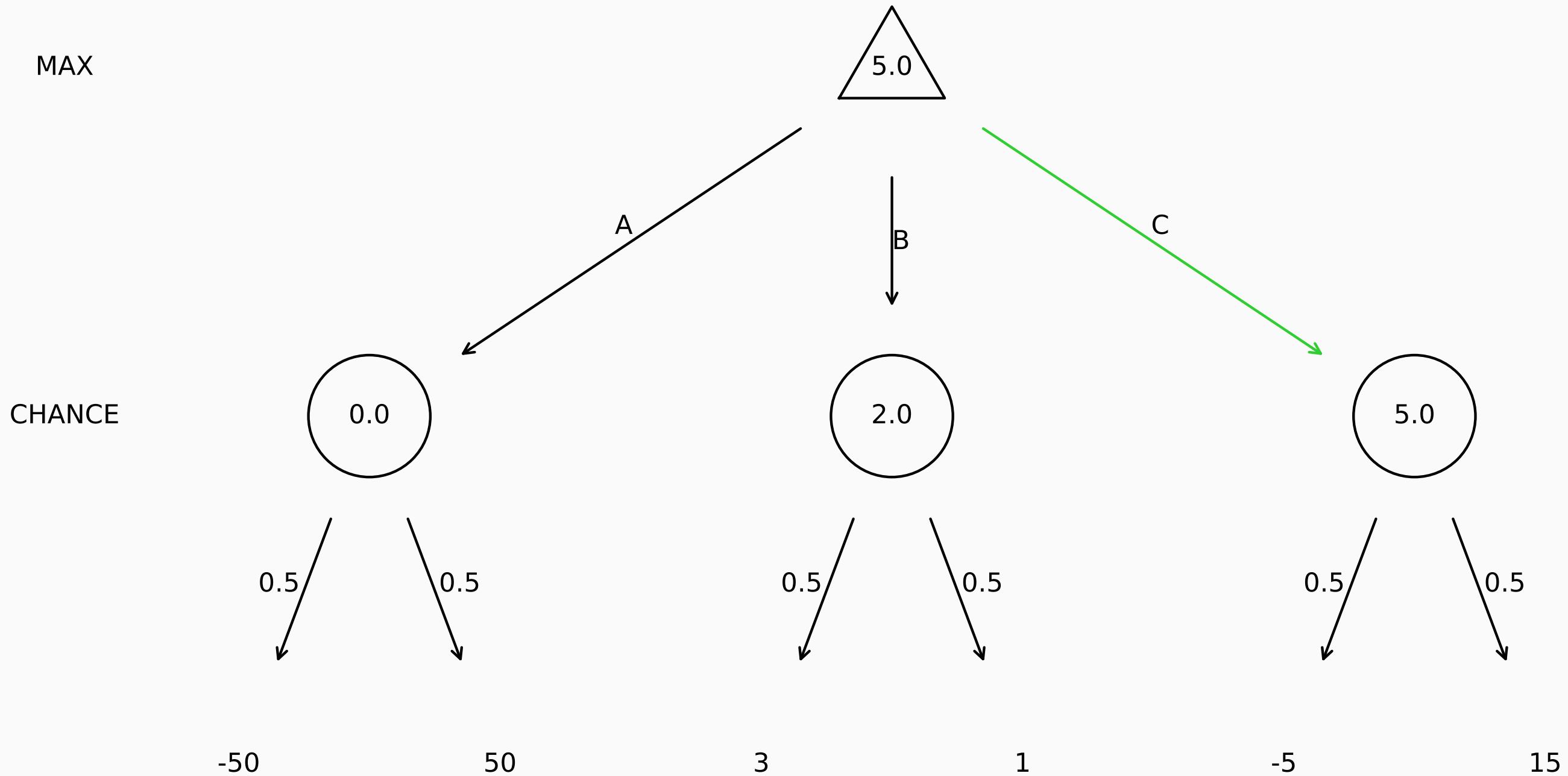
Example: the bucket game, game tree



The expectimax search algorithm

$$\text{expectimax}(s) = \begin{cases} \text{utility}(s, \text{MAX}), & \text{if } \text{isTerminal}(s) \\ \max_{a \in \text{actions}(s)} \text{expectimax}(\text{result}(s, a)), & \text{if } \text{toMove}(s) \text{ is MAX} \\ \sum_r P(r) \text{expectimax}(\text{result}(s, r)), & \text{if } \text{toMove}(s) \text{ is CHANCE} \end{cases}$$

The expectimax search algorithm, bucket game tree



Outline

A game

Optimal decisions in games

Cutting off search

Monte Carlo tree search

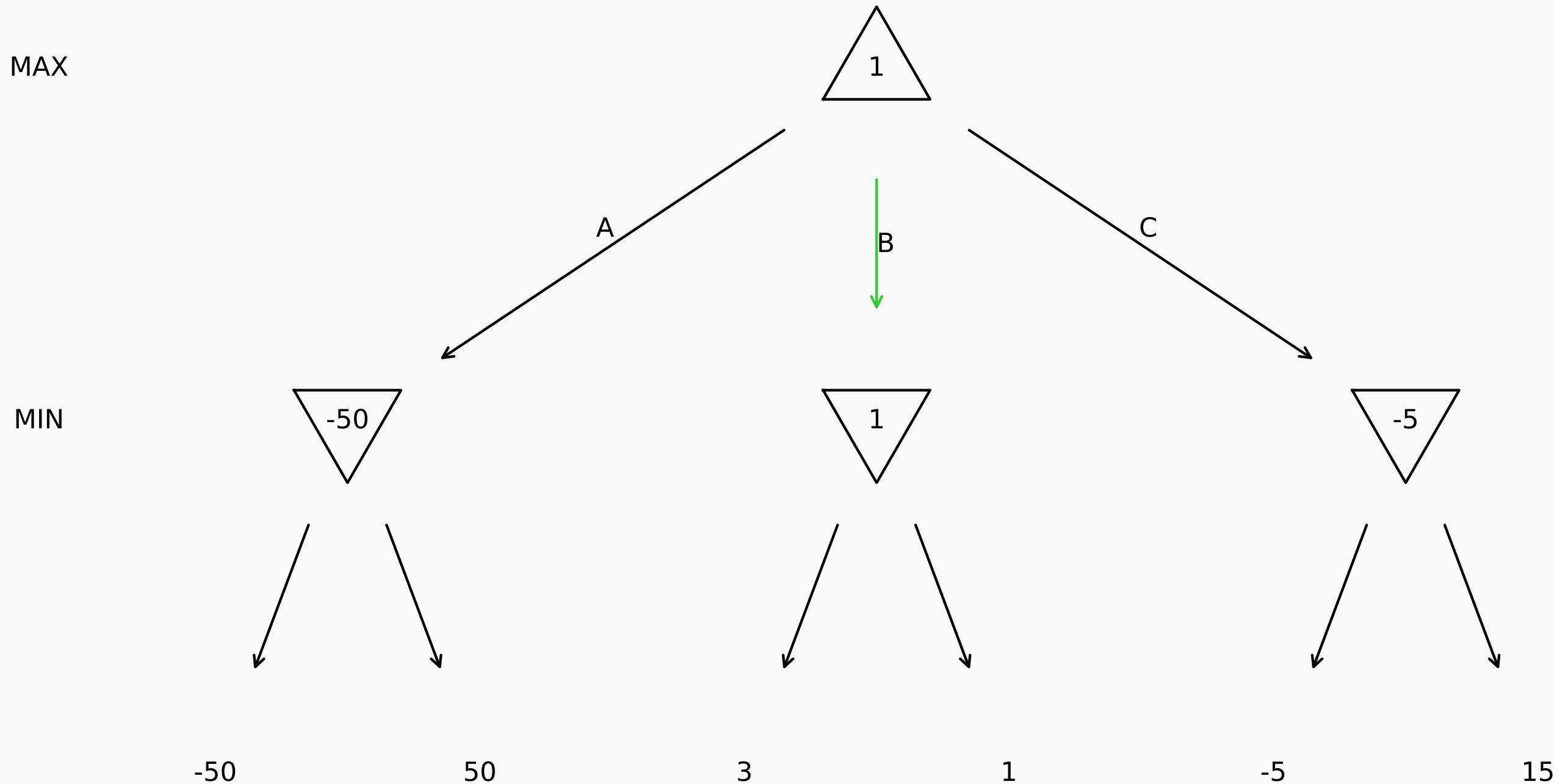
Stochastic games

Partially observable games

The minimax search algorithm

$$\text{minimax}(s) = \begin{cases} \text{utility}(s, \text{MAX}), & \text{if } \text{isTerminal}(s) \\ \max_{a \in \text{actions}(s)} \text{minimax}(\text{result}(s, a)), & \text{if } \text{toMove}(s) \text{ is MAX} \\ \min_{a \in \text{actions}(s)} \text{minimax}(\text{result}(s, a)), & \text{if } \text{toMove}(s) \text{ is MIN} \end{cases}$$

The minimax search algorithm, bucket game tree



Alpha-beta pruning

$$\text{alphaBeta}(s, \alpha, \beta) = \begin{cases} \text{utility}(s, \text{MAX}), & \text{if } \text{isTerminal}(s) \\ \text{maxValue}(s, \alpha, \beta), & \text{if } \text{toMove}(s) \text{ is MAX} \\ \text{minValue}(s, \alpha, \beta), & \text{if } \text{toMove}(s) \text{ is MIN} \end{cases}$$

α : the current best value of a max-node

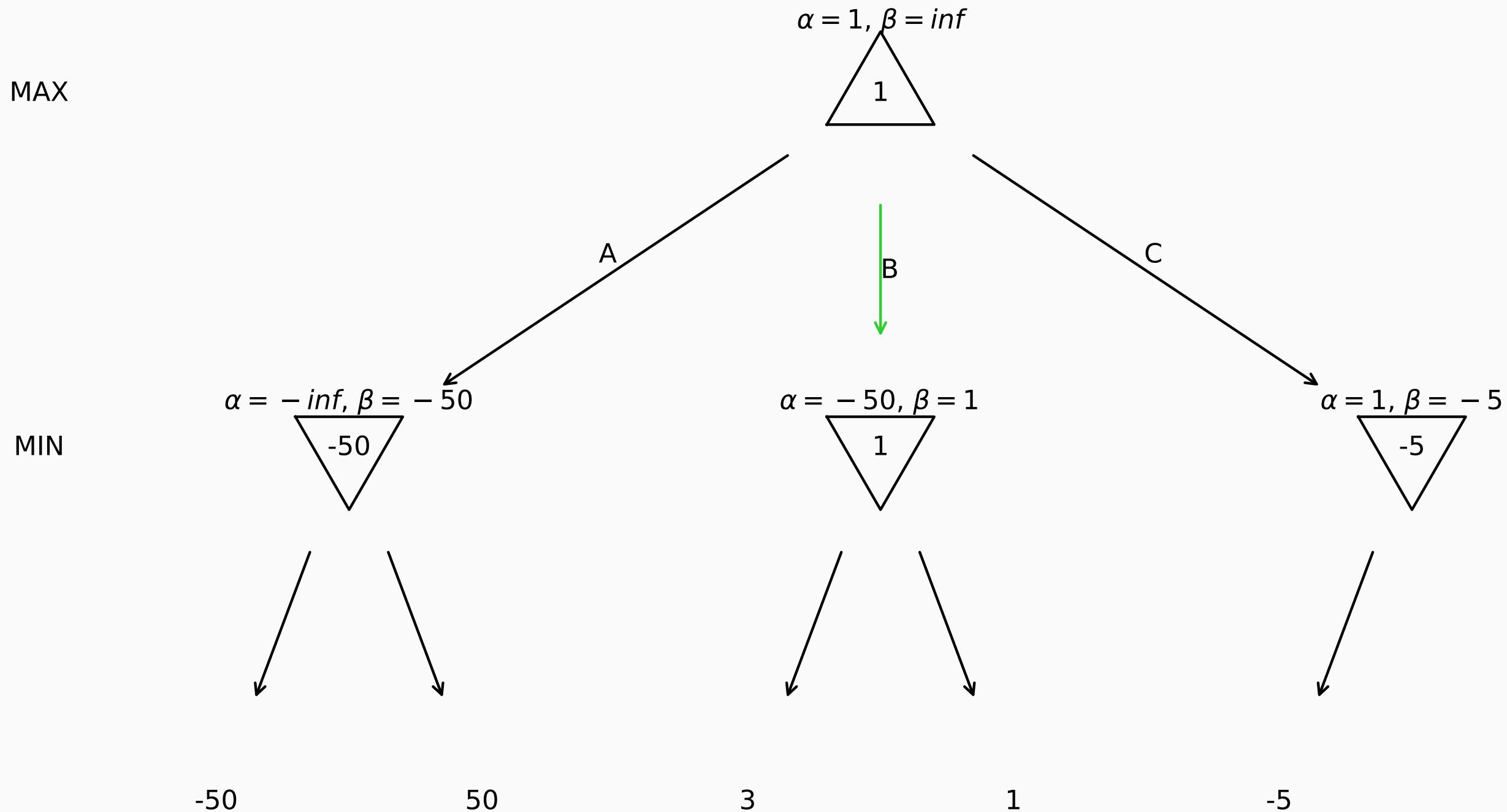
β : the current best value of a min-node

```
function MAXVALUE( $s, \alpha, \beta$ )
     $v \leftarrow -\infty$ 
    for each  $a$  in  $\text{actions}(s)$  do
         $v \leftarrow \max(v, \text{alphaBeta}(\text{result}(s, a), \alpha, \beta))$ 
        if  $v \geq \beta$  then return  $v$ 
         $\alpha \leftarrow \max(\alpha, v)$ 
    end for
    return  $v$ 
end function
```

```
function MINVALUE( $s, \alpha, \beta$ )
     $v \leftarrow \infty$ 
    for each  $a$  in  $\text{actions}(s)$  do
         $v \leftarrow \min(v, \text{alphaBeta}(\text{result}(s, a), \alpha, \beta))$ 
        if  $v \leq \alpha$  then return  $v$ 
         $\beta \leftarrow \min(\beta, v)$ 
    end for
    return  $v$ 
end function
```

Initially, $\text{alphaBeta}(s, -\infty, \infty)$ is called

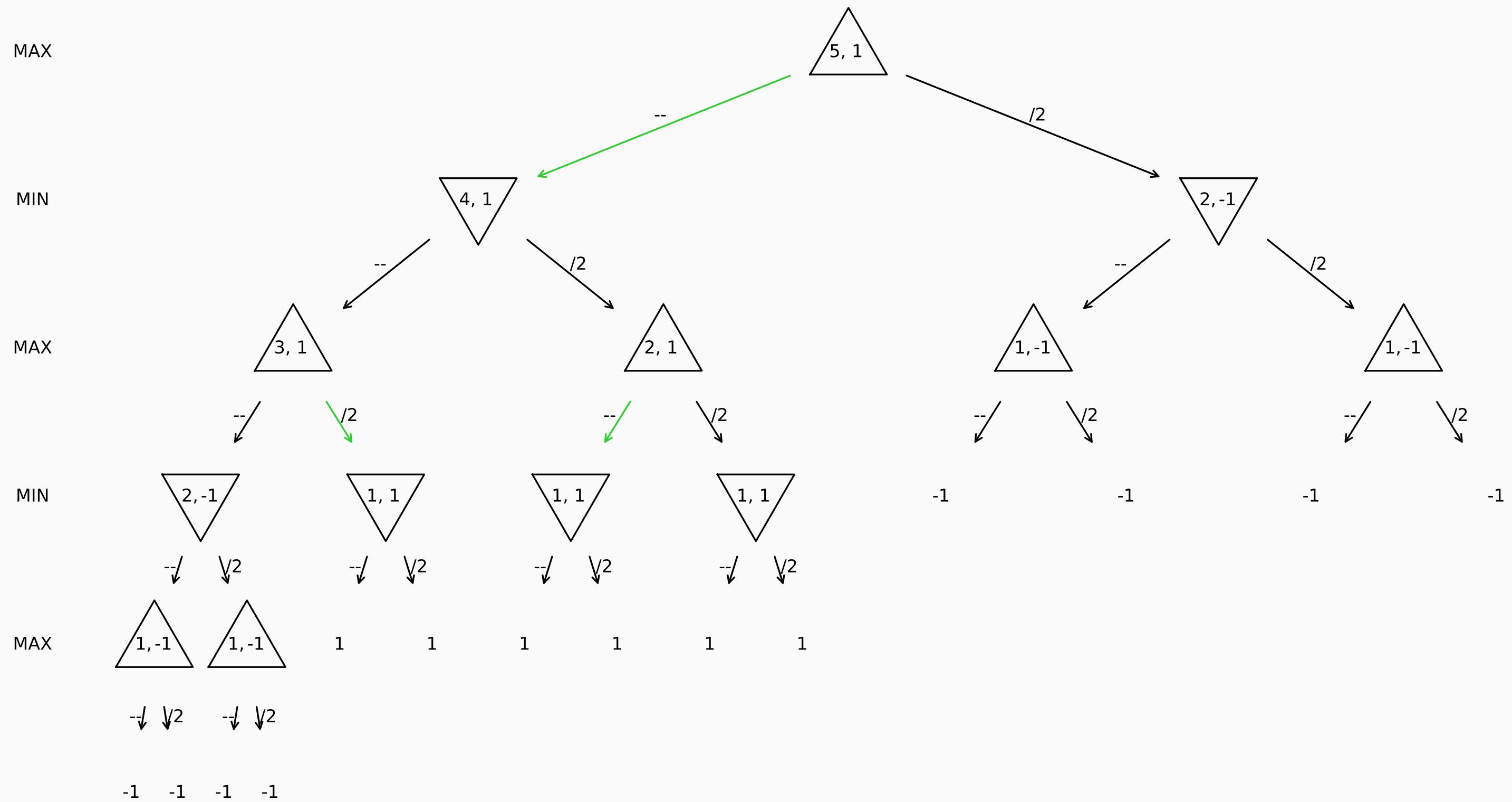
Alpha-beta pruning, bucket game tree



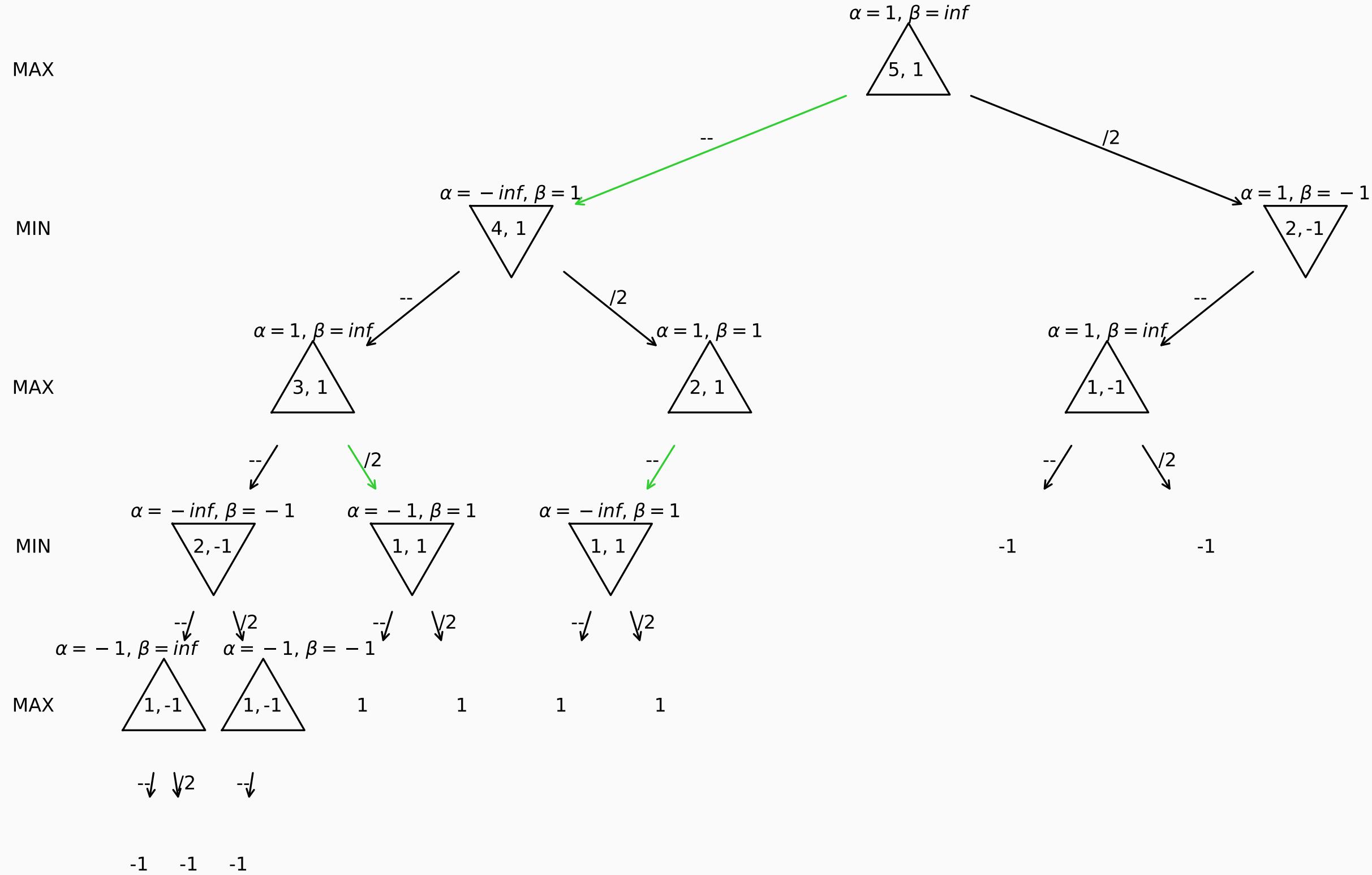
Example: the halving game (a variant of the Nim game)

- Start with a number N
- Players take turns either decrementing N or replacing it with $\frac{N}{2}$ rounded down
- The player that is left with 0 wins

The halving game tree, $N = 5$, without alpha-beta pruning



The halving game tree, $N = 5$, with alpha-beta pruning



Outline

A game

Optimal decisions in games

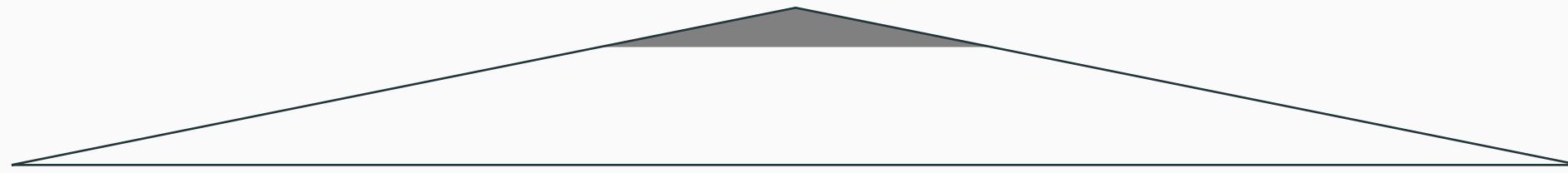
Cutting off search

Monte Carlo tree search

Stochastic games

Partially observable games

Heuristic minimax (depth limited search)



$$h\text{Minimax}(s, d) = \begin{cases} \text{utility}(s, \text{MAX}), & \text{if } \text{isTerminal}(s) \\ \text{eval}(s, \text{MAX}), & \text{if } \text{isCutoff}(s, d) \\ \max_{a \in \text{actions}(s)} h\text{Minimax}(\text{result}(s, a), d + 1), & \text{if } \text{toMove}(s) \text{ is MAX} \\ \min_{a \in \text{actions}(s)} h\text{Minimax}(\text{result}(s, a), d + 1), & \text{if } \text{toMove}(s) \text{ is MIN} \end{cases}$$

- $\text{eval}(s, \text{MAX})$ is an estimate of the true value $\text{minimax}(s)$

- Chess example:

- $\text{eval}(s) = w_1 \text{material}(s) + w_2 \text{mobility}(s) + w_3 \text{kingSafety}(s) + w_4 \text{centerControl}(s)$
 - $\text{material}(s) = 16(Q - Q') + 8(R - R') + 5(B - B' + N - N') + P - P'$
 - $\text{mobility}(s) = \text{numLegalMoves} - \text{numLegalMoves}'$
 - ...

Search versus lookup

Should we perform search on the first chess move in a new game?

Outline

A game

Optimal decisions in games

Cutting off search

Monte Carlo tree search

Stochastic games

Partially observable games

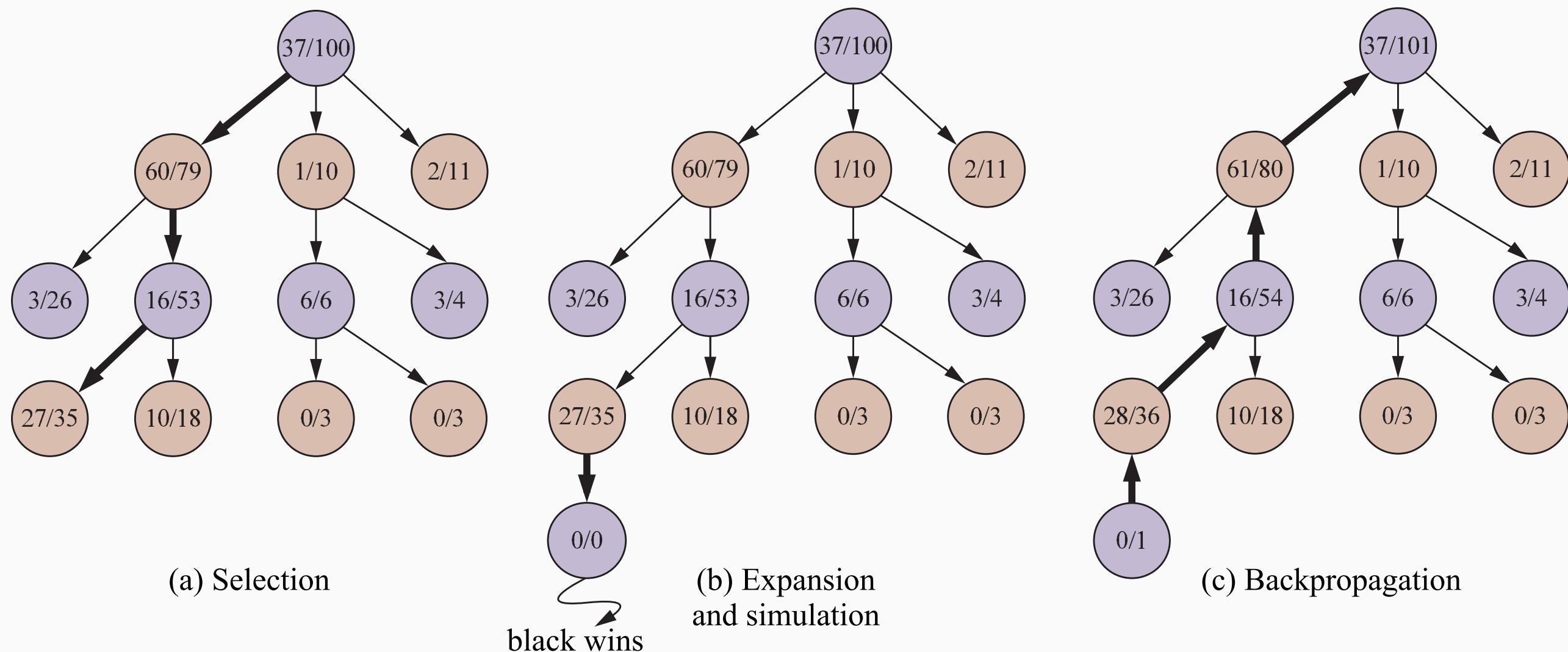
Monte Carlo methods

Monte Carlo methods, or Monte Carlo experiments, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle.

[...] The method is useful for obtaining numerical solutions to problems too complicated to solve analytically.

Wikipedia

Monte Carlo tree search



- It is black's turn (root node is black)
- Black nodes show white's wins and white nodes show black's wins
 - Root node shows that white has won 37 out of 100 simulations (playouts)
- Black first chooses the move that has lead to 60 wins out of 79 playouts

Monte Carlo tree search algorithm

```
function MONTE-CARLO-TREE-SEARCH(state) returns an action
  tree  $\leftarrow$  NODE(state)
  while IS-TIME-REMAINING() do
    leaf  $\leftarrow$  SELECT(tree)
    child  $\leftarrow$  EXPAND(leaf)
    result  $\leftarrow$  SIMULATE(child)
    BACK-PROPAGATE(result, child)
  return the move in ACTIONS(state) whose node has highest number of playouts
```

Monte Carlo tree search algorithm

function MONTE-CARLO-TREE-SEARCH(*state*) **returns** *an action*

tree \leftarrow NODE(*state*)

while IS-TIME-REMAINING() **do**

leaf \leftarrow SELECT(*tree*)

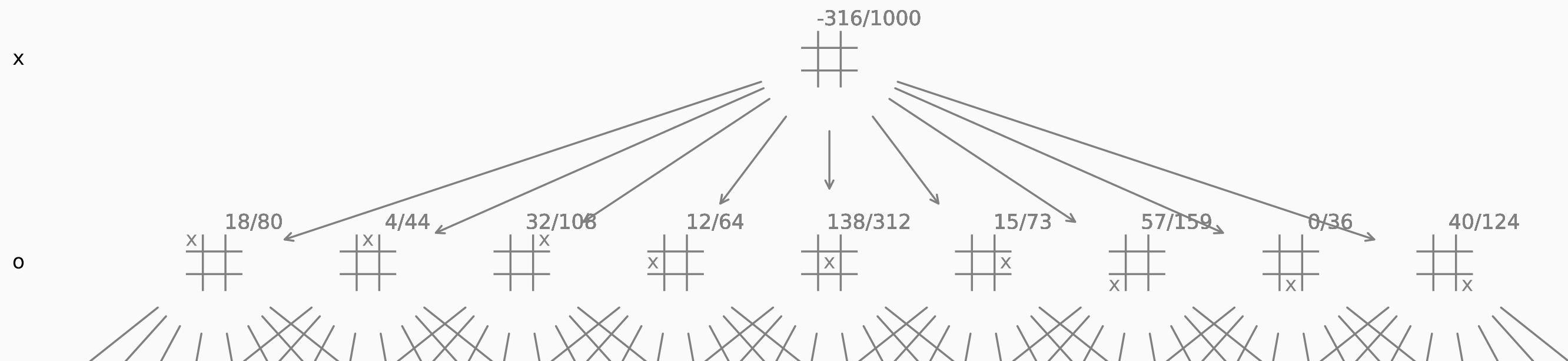
child \leftarrow EXPAND(*leaf*)

result \leftarrow SIMULATE(*child*)

 BACK-PROPAGATE(*result*, *child*)

return the move in ACTIONS(*state*) whose node has highest number of playouts

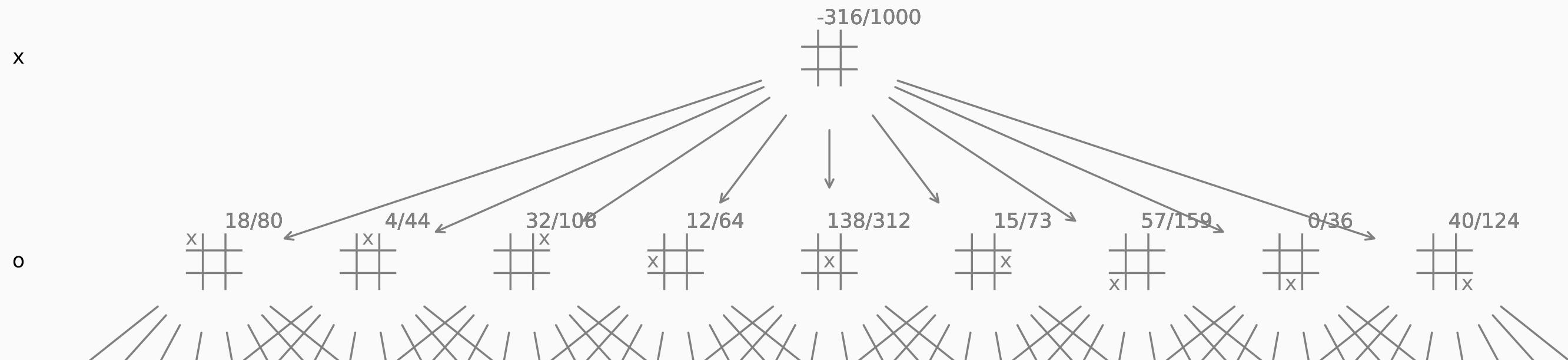
For example, which action would be returned here:



Monte Carlo tree search algorithm

```
function MONTE-CARLO-TREE-SEARCH(state) returns an action
  tree  $\leftarrow$  NODE(state)
  while IS-TIME-REMAINING() do
    leaf  $\leftarrow$  SELECT(tree)
    child  $\leftarrow$  EXPAND(leaf)
    result  $\leftarrow$  SIMULATE(child)
    BACK-PROPAGATE(result, child)
  return the move in ACTIONS(state) whose node has highest number of playouts
```

For example, which action would be returned here:



Answer: place x in the middle square

Selection policy

- Exploration
 - Explore states that have had few playouts
- Exploitation
 - Exploit states that have done well in past playouts,

Selection policy

- Exploration
 - Explore states that have had few playouts
- Exploitation
 - Exploit states that have done well in past playouts,

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\ln N(\text{parent}(n))}{N(n)}}$$

- The node n with highest $UCB1(n)$ is selected
- $U(n)$: total utility of all playouts that went through node n
- $N(n)$: the number of playouts through node n
- $\text{parent}(n)$: parent node of n
- C : constant that balances exploration and exploitation
 - High C favors exploration
 - Usually $C = \sqrt{2}$ is used

Playouts

A Tic-tac-toe agent is quite good after 1000 iterations

AlphaZero (2017): Monte Carlo tree search with neural networks

- Modified Monte Carlo tree search:
 - Value network: how likely the current state is a win
 - Used instead of playouts
 - Trained through self-play games (games against itself using modified Monte Carlo tree search)

AlphaZero (2017): Monte Carlo tree search with neural networks

- Modified Monte Carlo tree search:
 - Value network: how likely the current state is a win
 - Used instead of playouts
 - Trained through self-play games (games against itself using modified Monte Carlo tree search)
 - Policy network: action probabilities, where good actions should have high probabilities
 - Used to improve the selection policy
 - Trained through self-play games

Outline

A game

Optimal decisions in games

Cutting off search

Monte Carlo tree search

Stochastic games

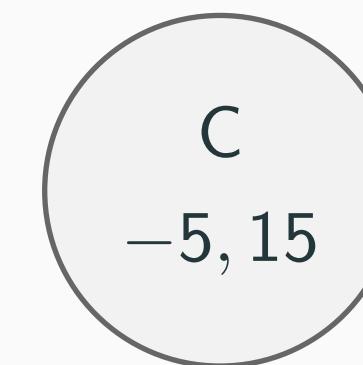
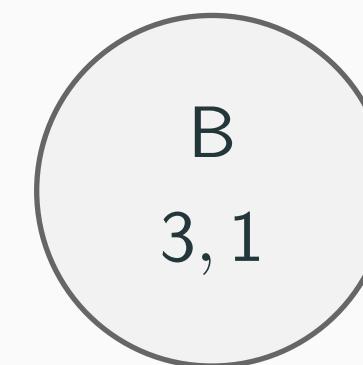
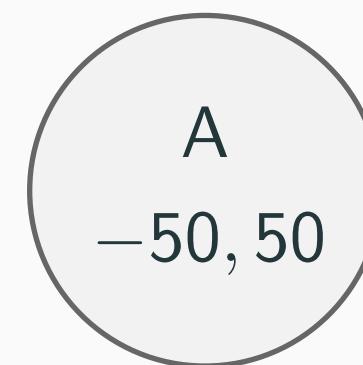
Partially observable games

Stochastic games

Games involving external random elements, such as dice throws

Example: modified bucket game

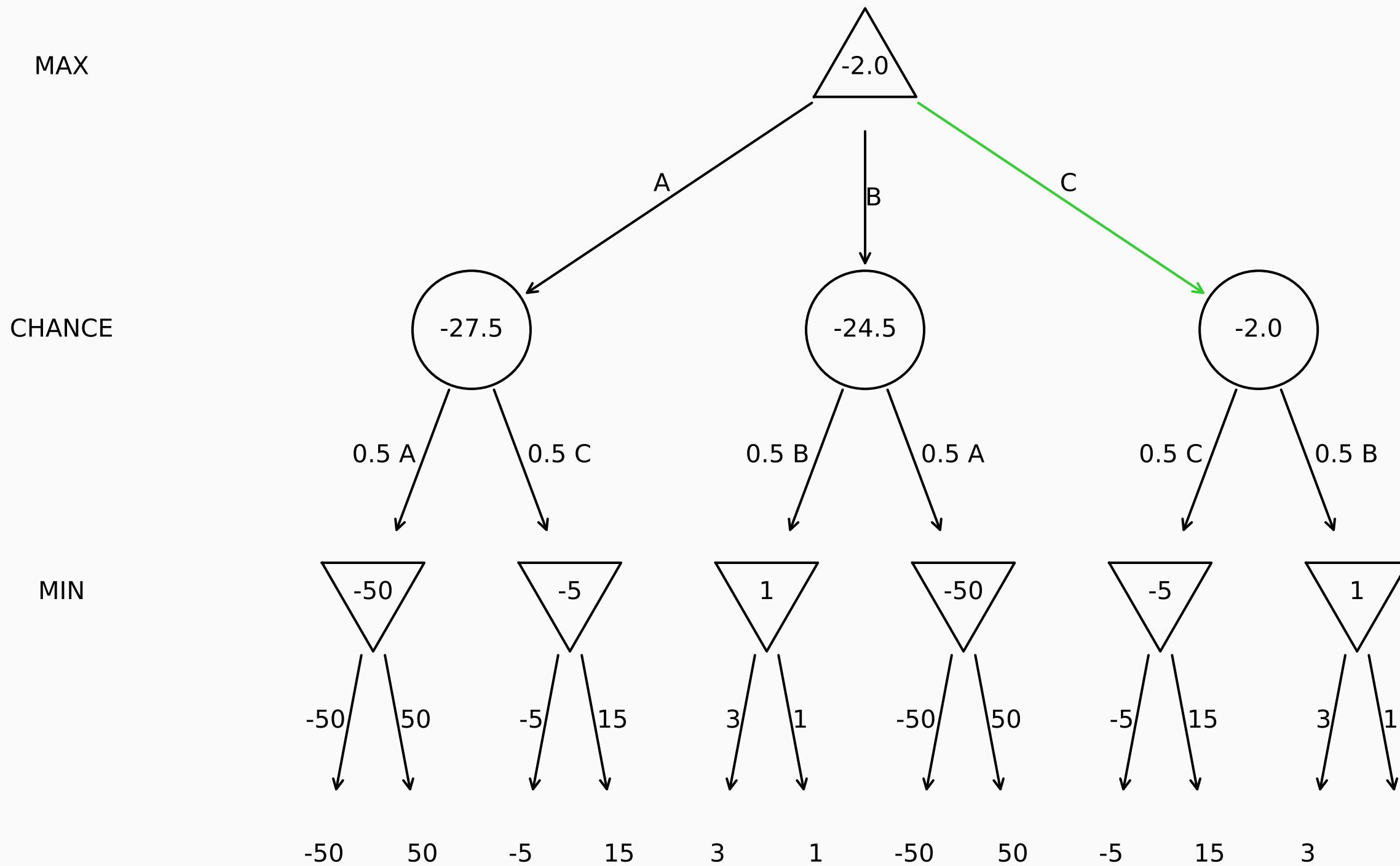
- The first player chooses bucket A, B or C
- **Flip a coin; if heads, move one bucket to the left (with wraparound)**
- The second player selects a number from the chosen bucket
- The goal of the first player is to maximize the selected number



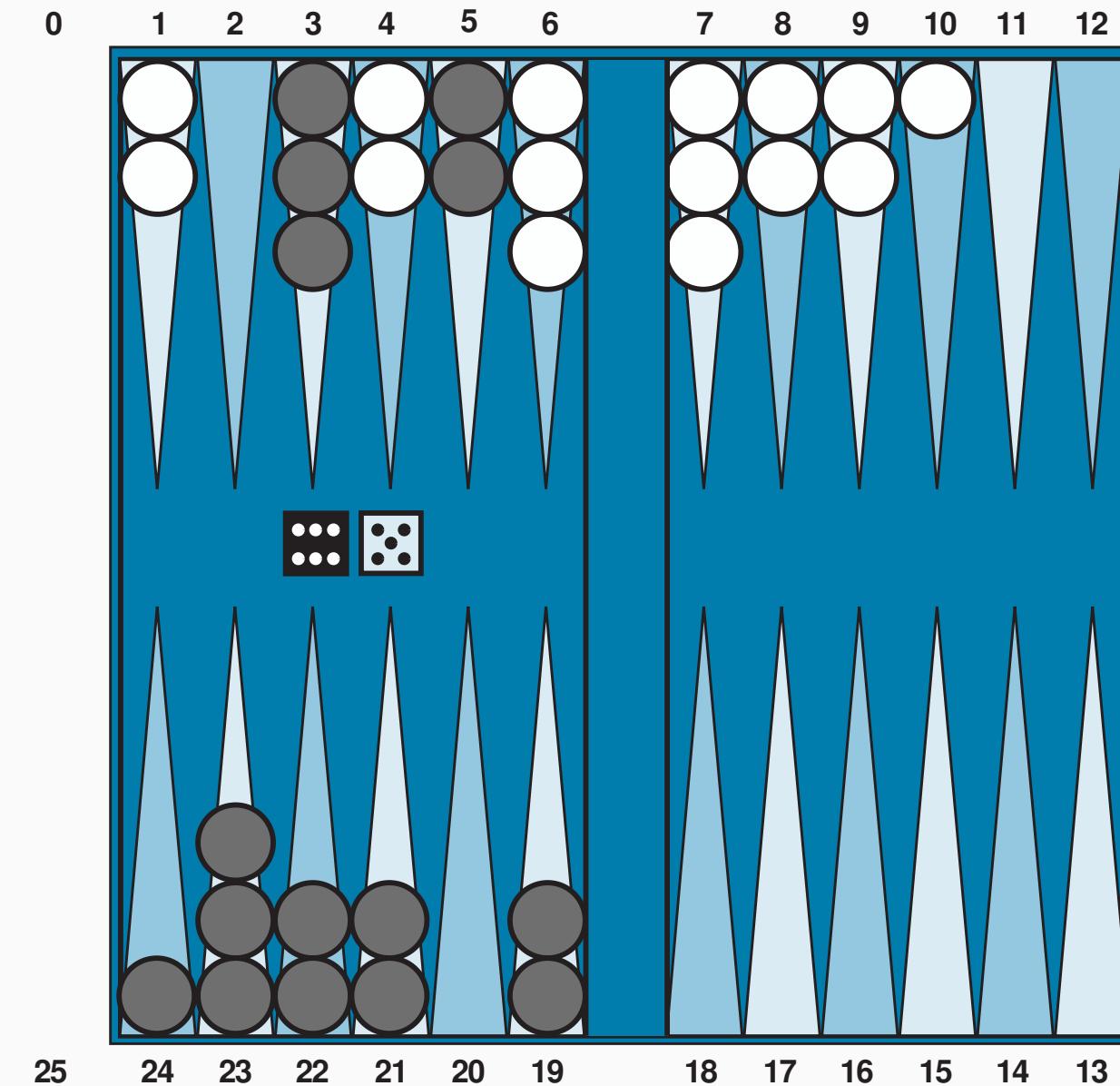
The expectiminimax search algorithm

$$\text{expectiminimax}(s) = \begin{cases} \text{utility}(s, \text{MAX}), & \text{if } \text{isTerminal}(s) \\ \max_{a \in \text{actions}(s)} \text{expectiminimax}(\text{result}(s, a)), & \text{if } \text{toMove}(s) \text{ is MAX} \\ \min_{a \in \text{actions}(s)} \text{expectiminimax}(\text{result}(s, a)), & \text{if } \text{toMove}(s) \text{ is MIN} \\ \sum_{a \in \text{actions}(s)} P(a) \text{expectiminimax}(\text{result}(s, a)), & \text{if } \text{toMove}(s) \text{ is CHANCE} \end{cases}$$

The expectiminimax search algorithm, modified bucket game tree



Example: backgammon



For more advanced stochastic games, Monte Carlo Tree Search can be used where each playout includes random dice rolls

Outline

A game

Optimal decisions in games

Cutting off search

Monte Carlo tree search

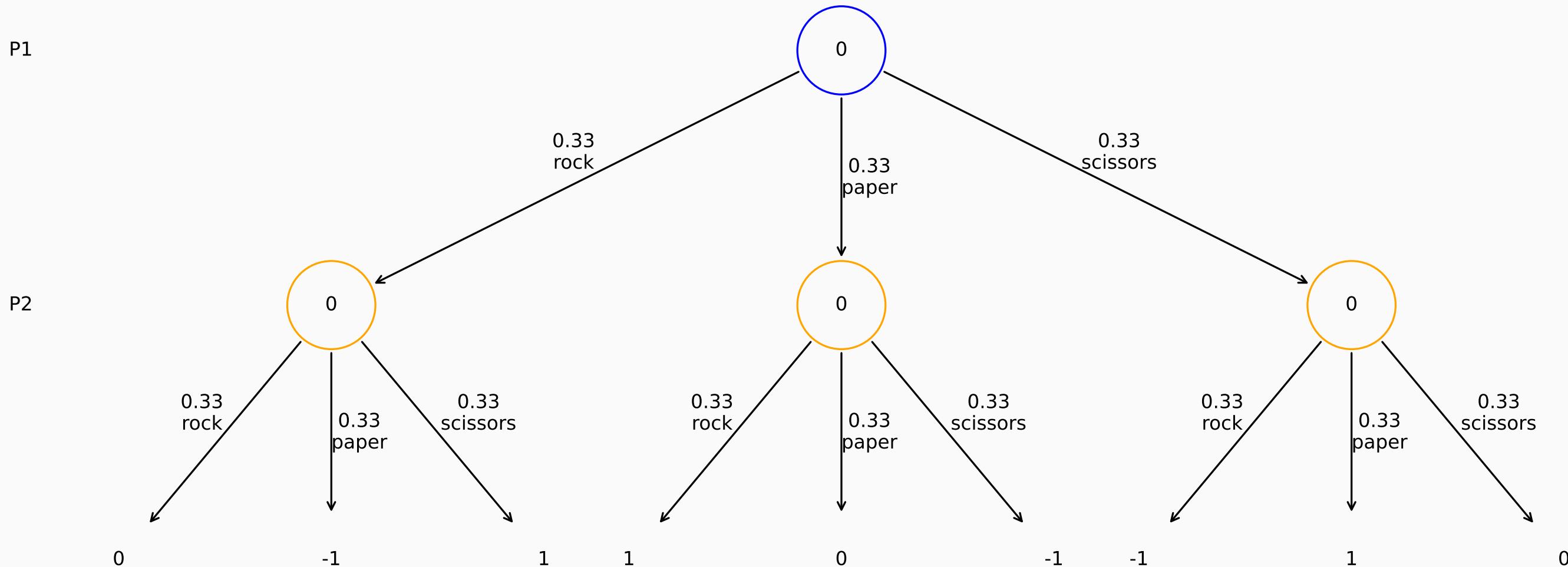
Stochastic games

Partially observable games

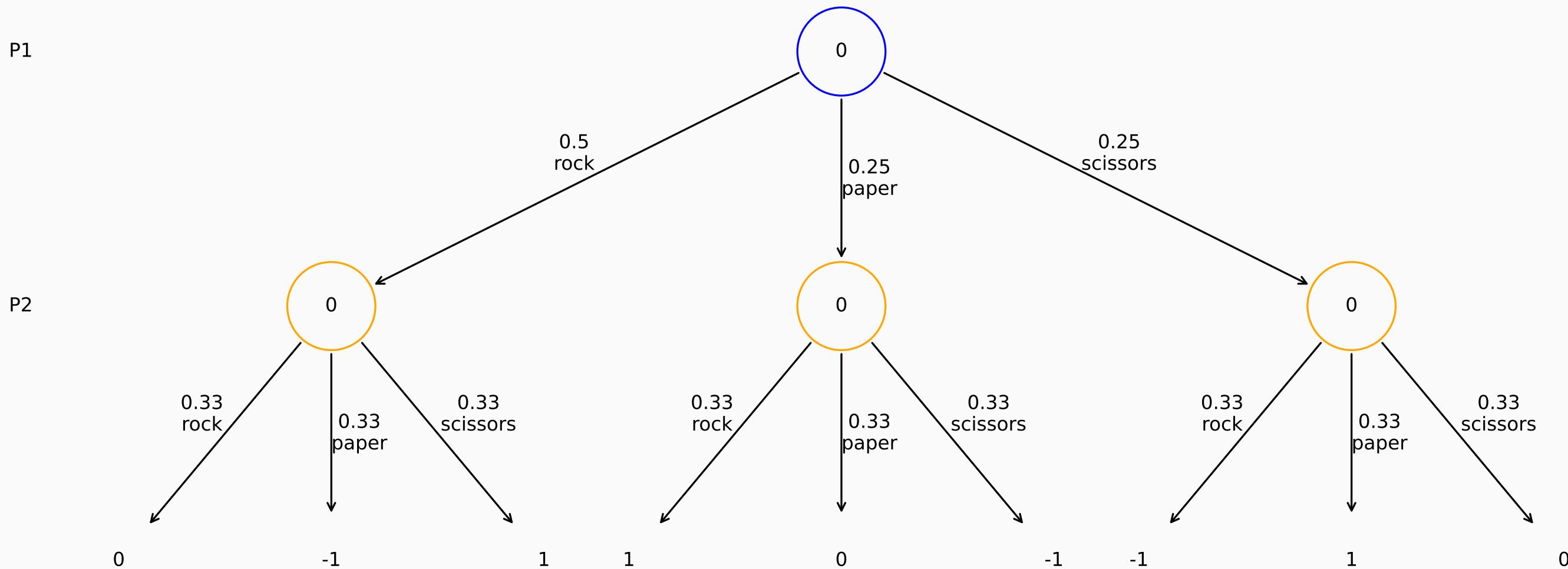
Partially observable games

- Game states not fully observable or players act simultaneously
 - Minimax and similar algorithms cannot be used

Example: rock paper scissors, Nash equilibrium (next lecture)

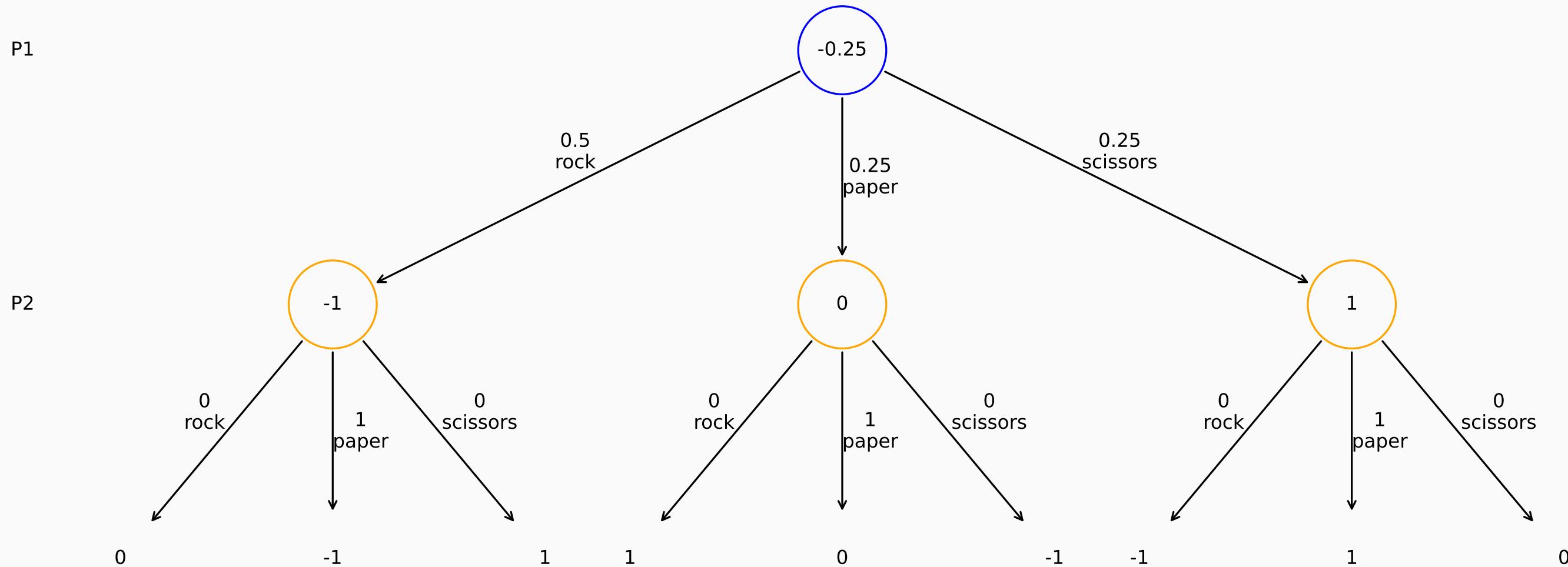


Example: rock paper scissors, P1 plays rock more often



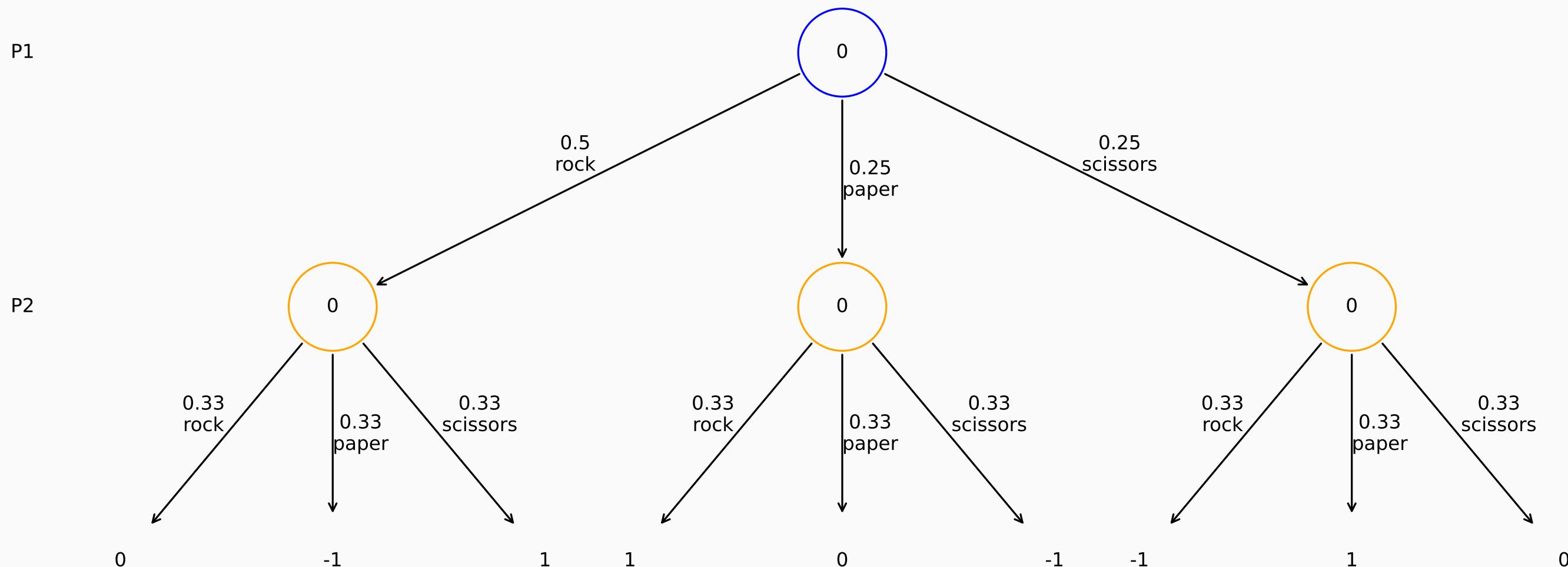
How can P2 exploit this?

Example: rock paper scissors, P1 plays rock more often, P2 exploits this



How can P2 exploit this? By playing paper more often

Regret minimization

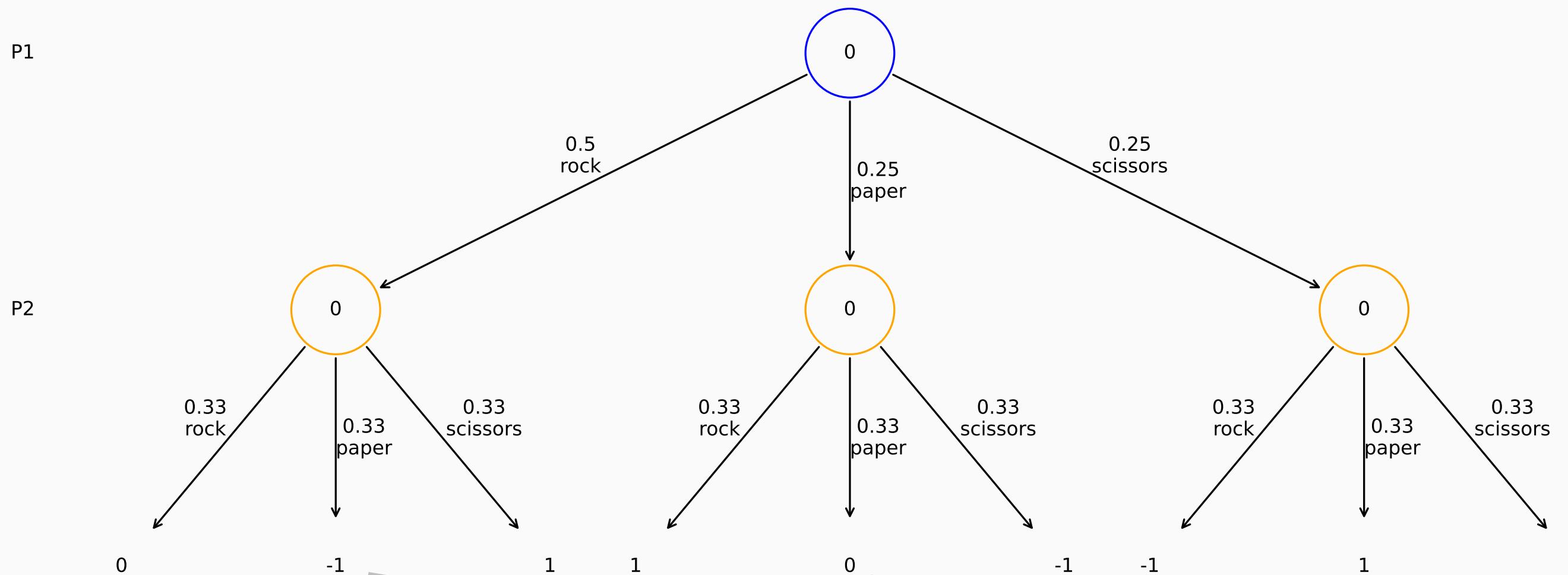


Regret

The utility an action would have received minus the utility actually received

- Regrets for each player's actions are summed over all games played
- Possible algorithm: choose the action with the highest summed regret

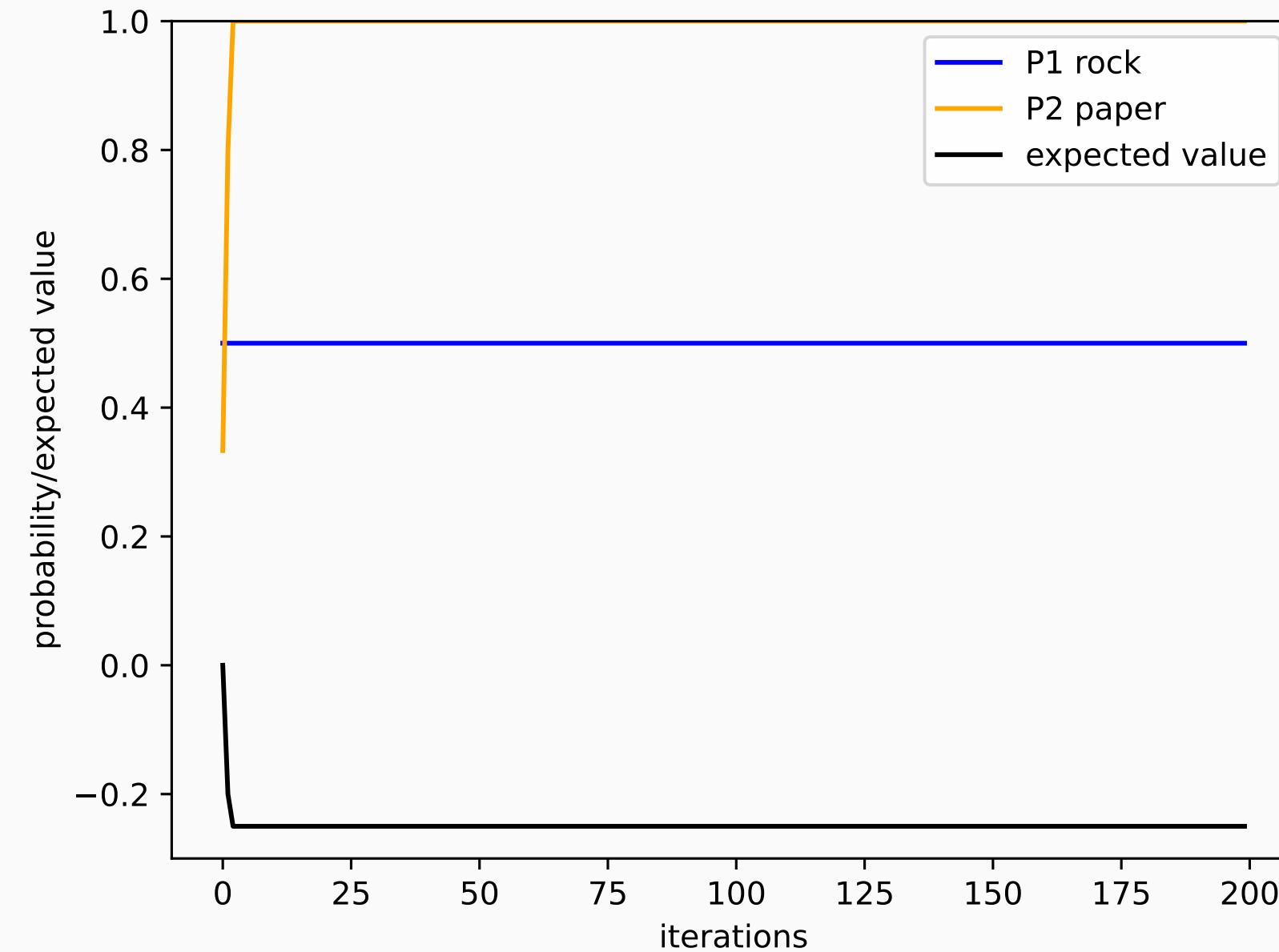
Regret minimization



For example, initially for P1, for actions (*rock, paper, scissors*): $\text{regrets} \leftarrow (0, 0, 0)$:

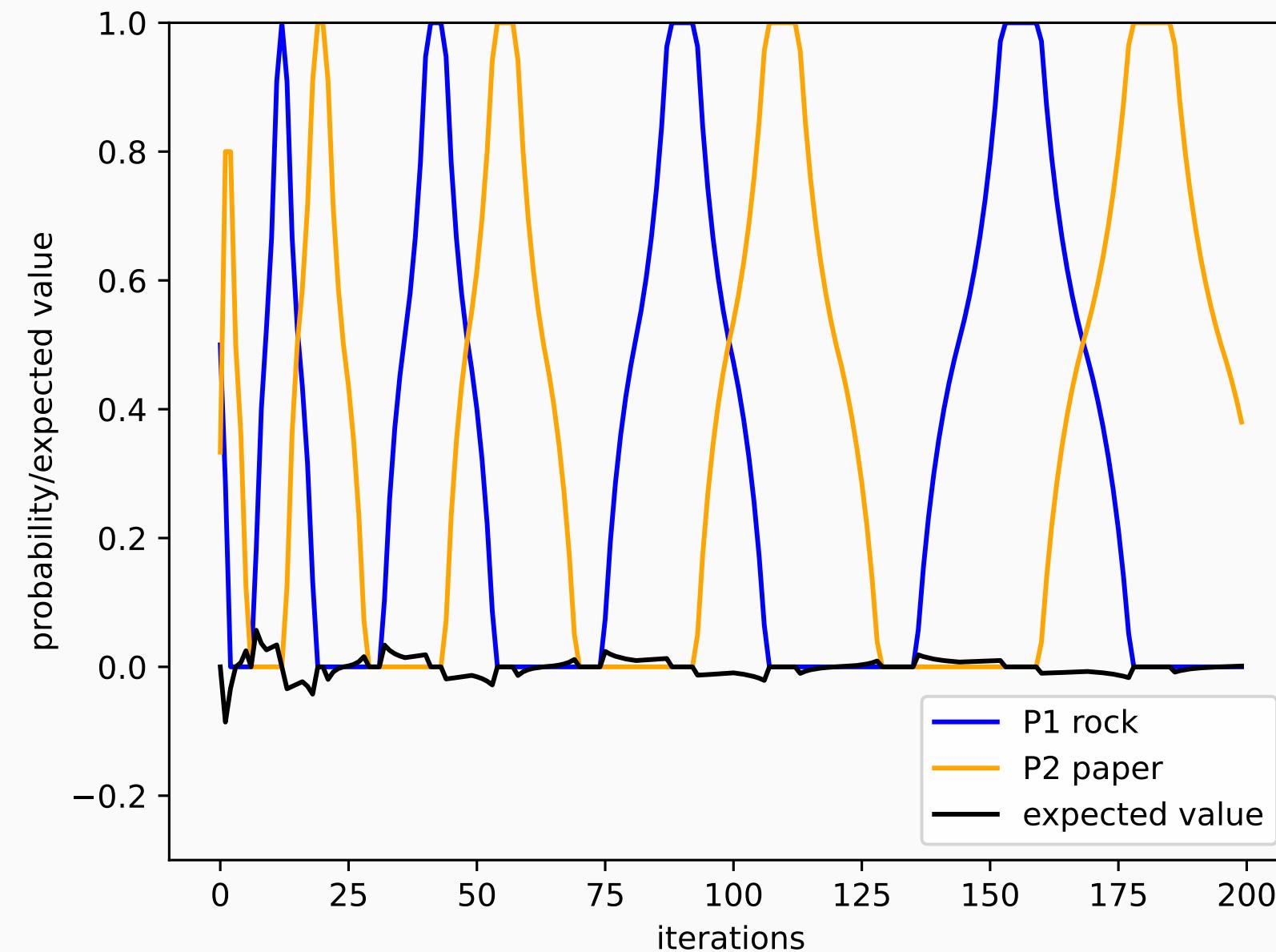
1. P1 chooses rock, P2 chooses paper: $\text{utility} \leftarrow -1$, $\text{regret} \leftarrow (-1, 0, 1) - \text{utility}$
2. Update: $\text{regrets} \leftarrow \text{regrets} + \text{regret}$
3. Result: $\text{regrets} = (0, 1, 2)$

Example: rock paper scissors, regret minimization result for P2



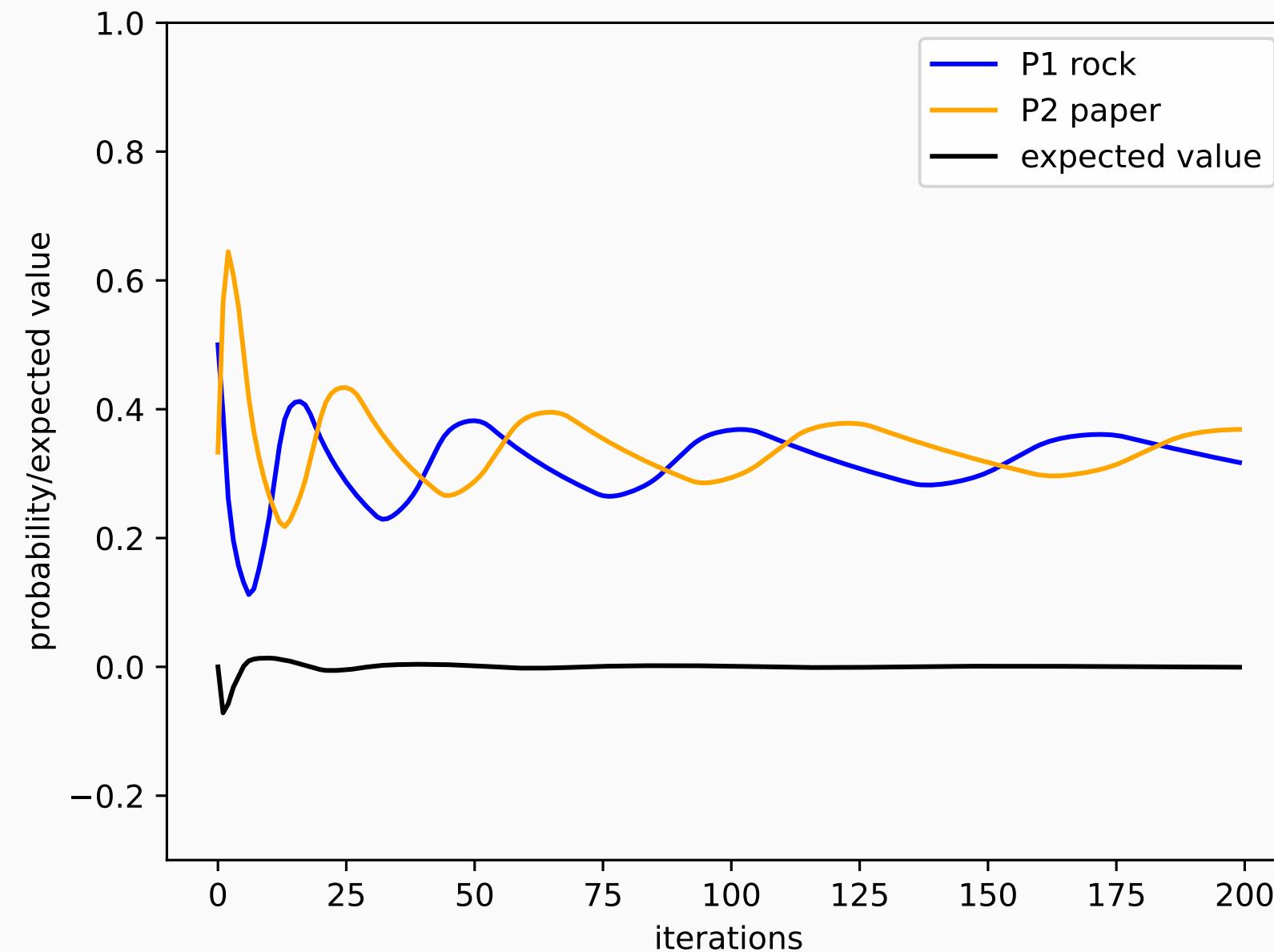
Regrets converted to probabilities, with the negative regrets first set to zero, for example,
 $(-1, 1, 2) \rightarrow (0, 1, 2) \rightarrow (0, \frac{1}{3}, \frac{2}{3})$

Example: rock paper scissors, regret minimization results for P1 and P2



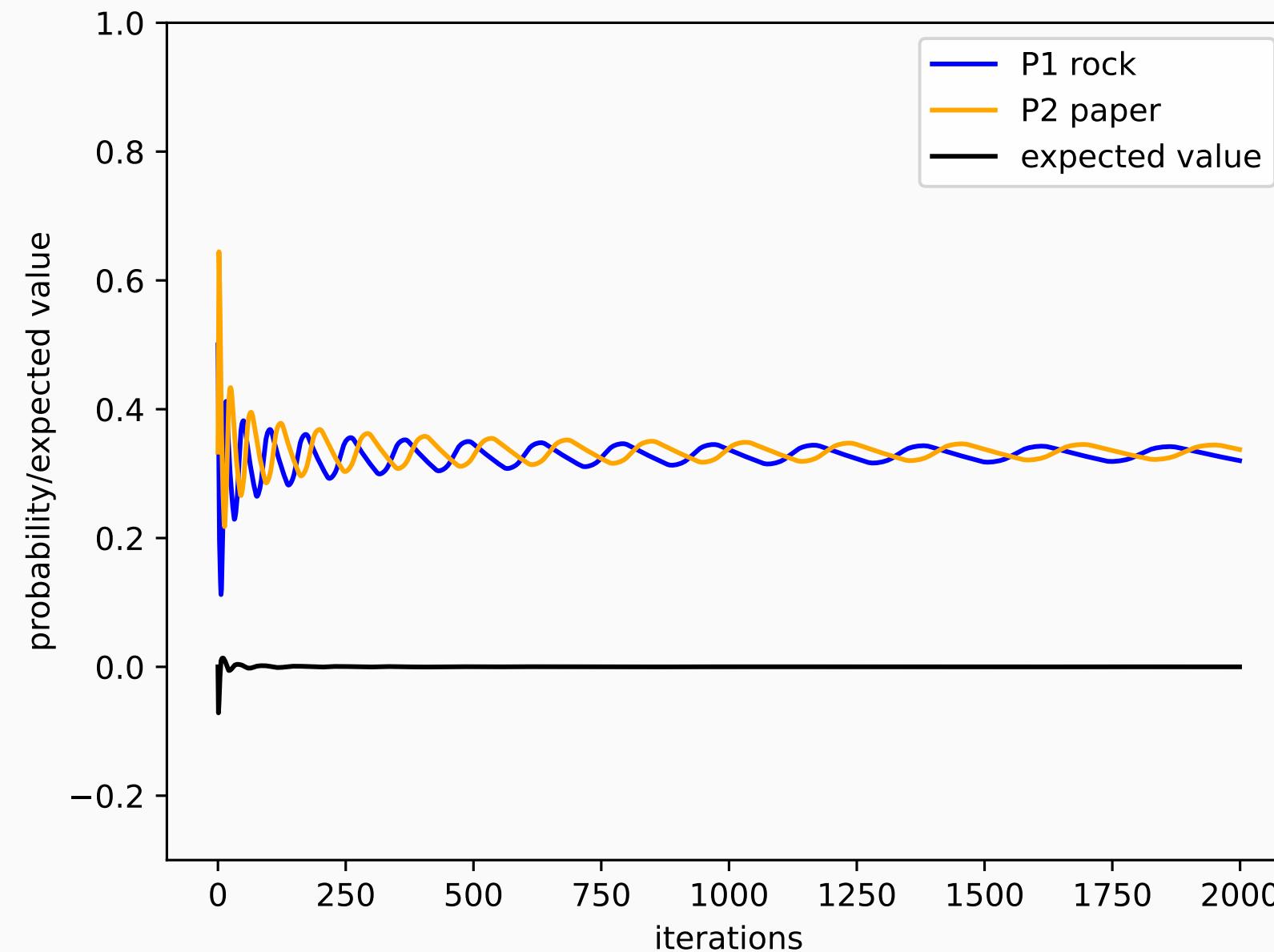
Regrets converted to probabilities, with the negative regrets first set to zero, for example,
 $(-1, 1, 2) \rightarrow (0, 1, 2) \rightarrow (0, \frac{1}{3}, \frac{2}{3})$

Example: rock paper scissors, regret minimization results for P1 and P2



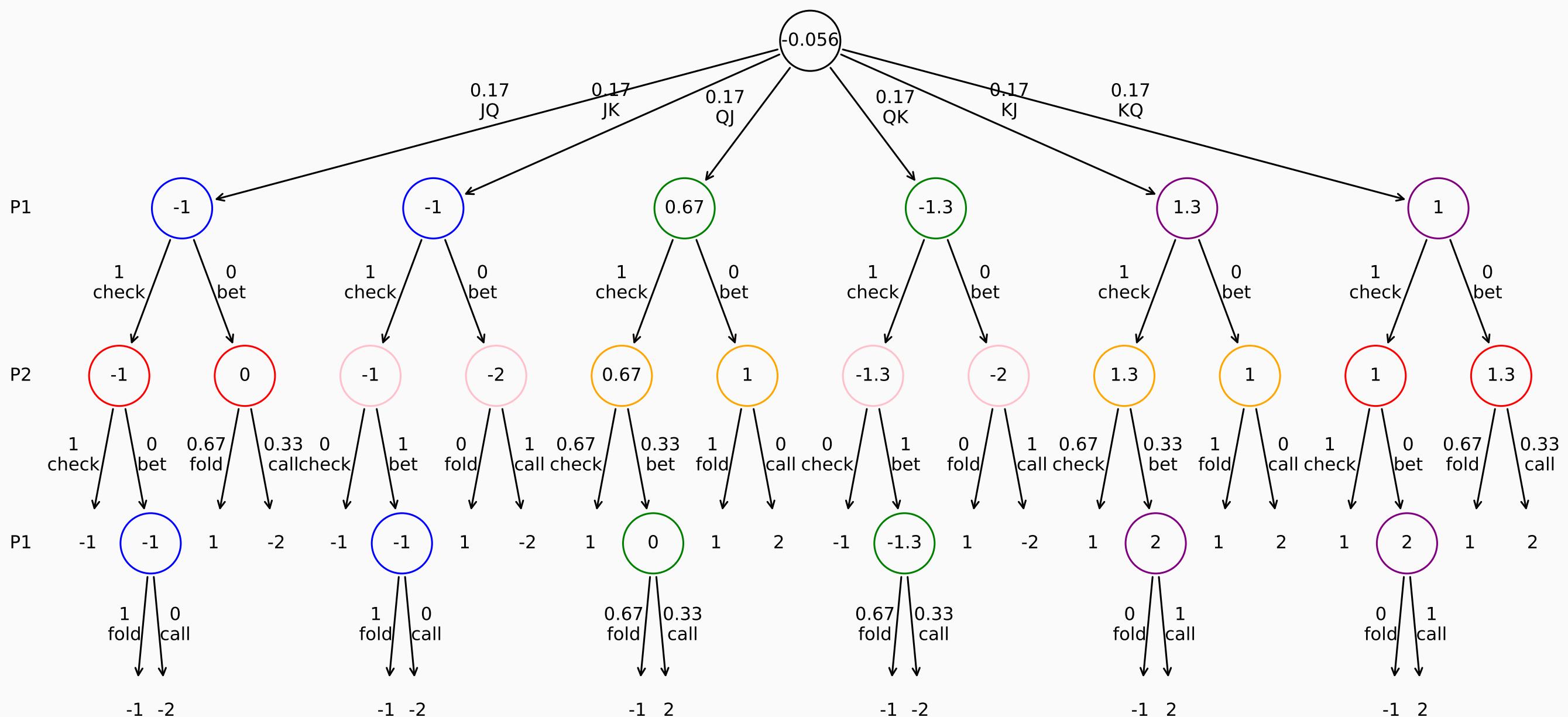
Convergence to Nash equilibrium can be achieved by using the cumulative average, $CA(x) = \frac{x_1 + \dots + x_n}{n}$, for each iteration n and the probabilities x_1, \dots, x_n up until then

Example: rock paper scissors, regret minimization results for P1 and P2



Convergence to Nash equilibrium can be achieved by using the cumulative average, $CA(x) = \frac{x_1 + \dots + x_n}{n}$, for each iteration n and the probabilities x_1, \dots, x_n up until then

Example: Kuhn poker, a Nash equilibrium



Pluribus (2019): poker bot



- Consistently beat professional poker players at 6-player no-limit Hold'em
- Key methods:
 - Regret minimization
 - Monte Carlo tree search
 - Self-play

AlphaStar (2019): StarCraft II bot



- Similar to AlphaZero, but was initially made to imitate the moves of the best human players
- Grandmaster level (top 0.2 percent of human players)

Game theory

Ole C. Eidheim

October 16, 2024

Department of Computer Science

Game theory definitions

Game theory

Systematic study of strategic interactions among rational individuals

Rational individual

Has well-defined objectives and implements the best available strategy to pursue them

Applications of game theory

- Economics
- Business
- Project management
- Political science
- Defense science and technology
- Biology
- Computer science and logic
- Epidemiology
- Philosophy
- Epidemiology
- Artificial intelligence and machine learning

Applications of game theory

- Economics
- Business
- Project management
- Political science
- Defense science and technology
- Biology
- Computer science and logic
- Epidemiology
- Philosophy
- Epidemiology
- Artificial intelligence and machine learning
- Although in practice:
 - Players are not always rational
 - Might be difficult to define the utility function (the value at the end a game)

Nash equilibrium for two-player simultaneous action games

Definitions

- $\pi_p(s, a)$: strategy for player $p \in \{1, 2\}$, i.e. probability of taking action a at state s
 - In this lecture:
 - In all but the last section, the state is empty and is therefore omitted
 - Each player has only two actions, thus the probability of taking the first action is defined to be π_p and the probability of taking the second action is then $1 - \pi_p$
- $E_p(\pi_1, \pi_2)$: expected value for player p of the given players' strategies

Nash equilibrium for two-player simultaneous action games

Definitions

- $\pi_p(s, a)$: strategy for player $p \in \{1, 2\}$, i.e. probability of taking action a at state s
 - In this lecture:
 - In all but the last section, the state is empty and is therefore omitted
 - Each player has only two actions, thus the probability of taking the first action is defined to be π_p and the probability of taking the second action is then $1 - \pi_p$
- $E_p(\pi_1, \pi_2)$: expected value for player p of the given players' strategies

The specific strategies (π_1^*, π_2^*) is a Nash equilibrium if:

$$E_1(\pi_1^*, \pi_2^*) \geq E_1(\pi_1, \pi_2^*) \text{ for all possible strategies } \pi_1$$

$$E_2(\pi_1^*, \pi_2^*) \geq E_2(\pi_1^*, \pi_2) \text{ for all possible strategies } \pi_2$$

and no player will then have an incentive to change his/her strategy

Pure vs mixed strategy Nash equilibria

- Pure strategy: selects a predetermined action, i.e. $\pi_p \in \{0, 1\}$
- Mixed strategy: selects actions according to a probability distribution, i.e. $\pi_p \in [0, 1]$
 - A pure strategy can be seen as a special case of a mixed strategy

Outline

Mixed strategies

Pure strategies

Repeated games

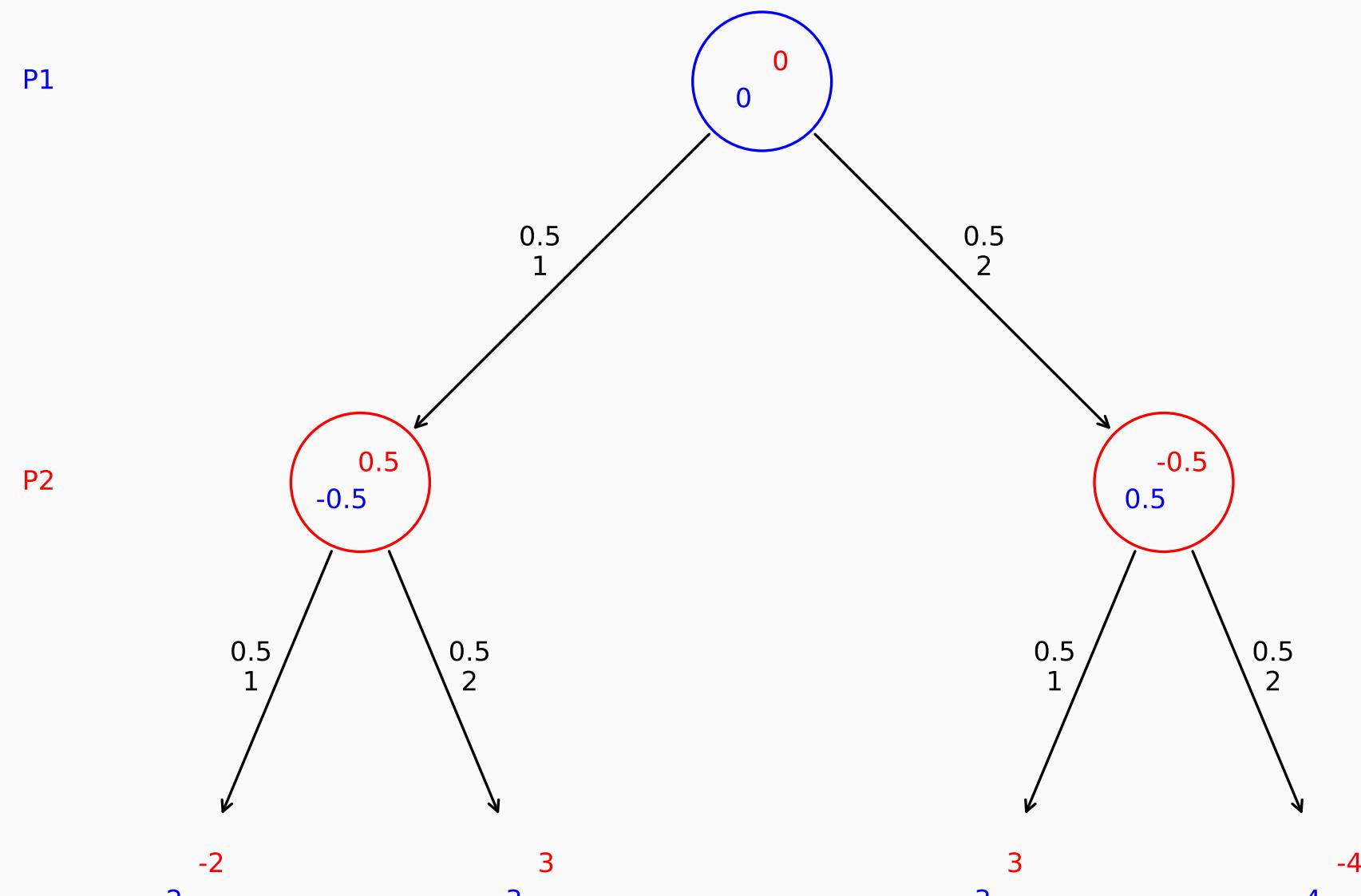
Infinite rounds

Unknown number of rounds

Two-finger Morra, payoff (utility) matrix

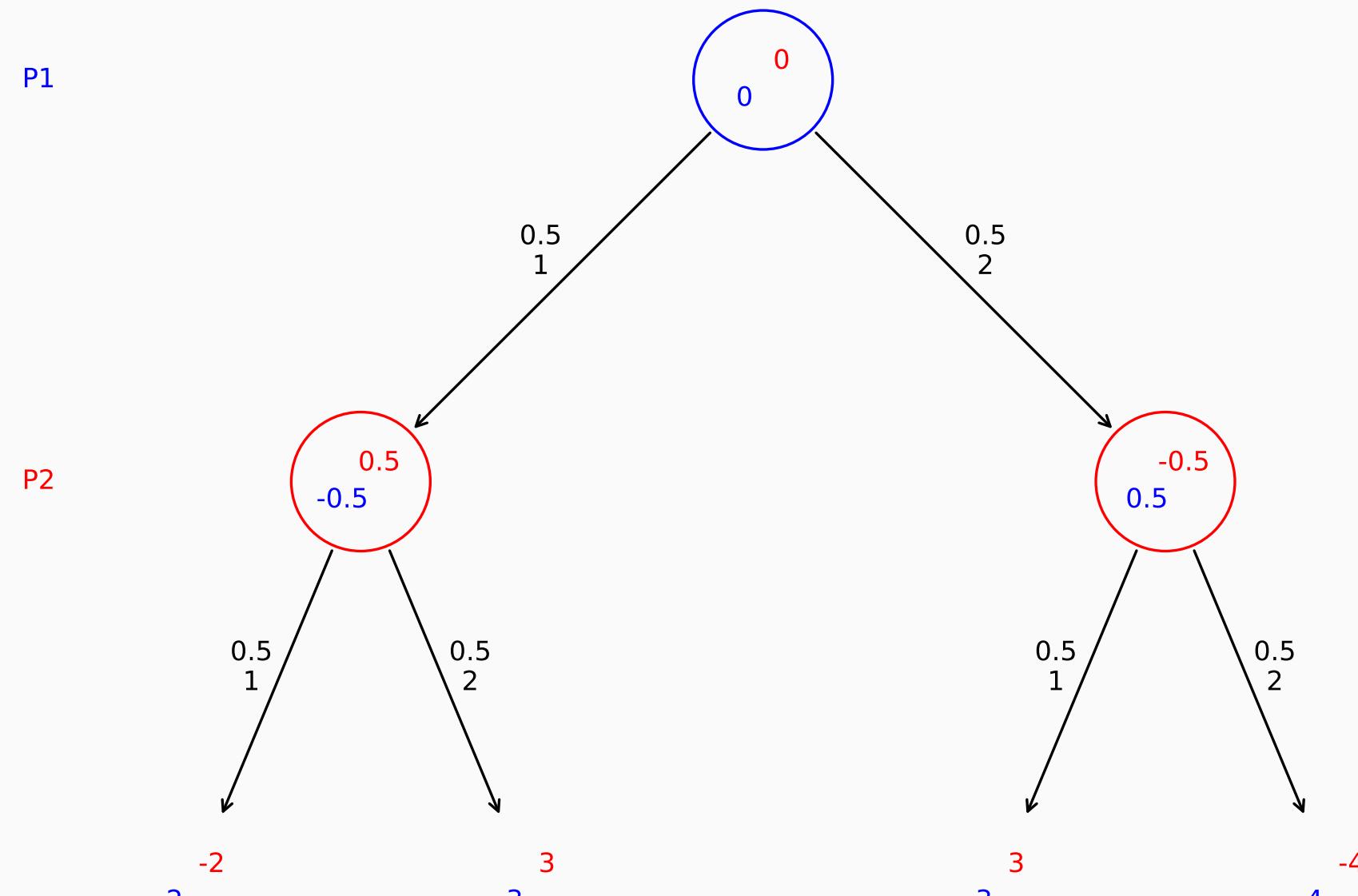
		P2	
		1	2
		-2	3
P1	1	2	-3
	2	3	-4
		-3	4

Two-finger Morra



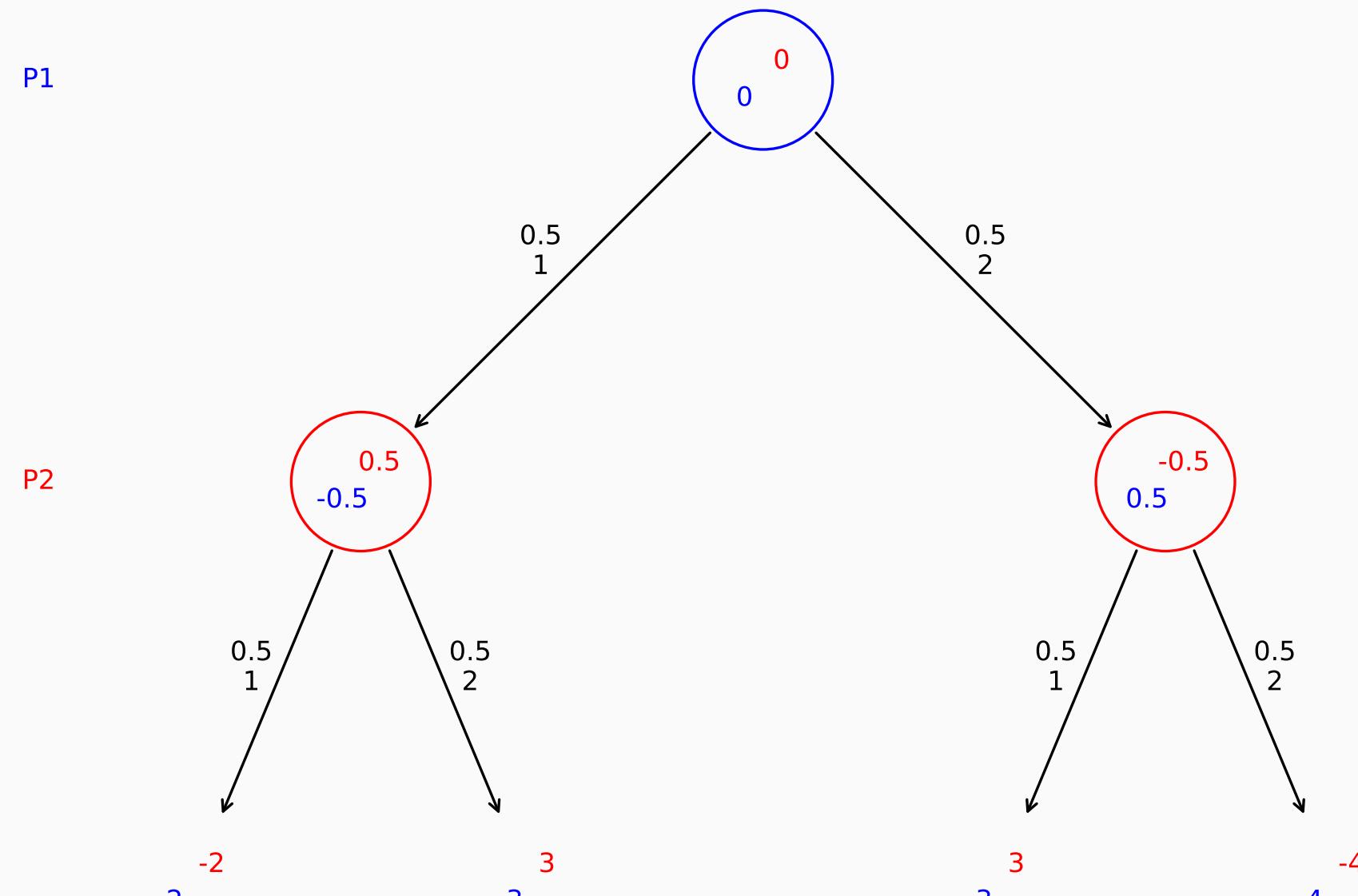
How can P1 exploit P2's strategy?

Two-finger Morra



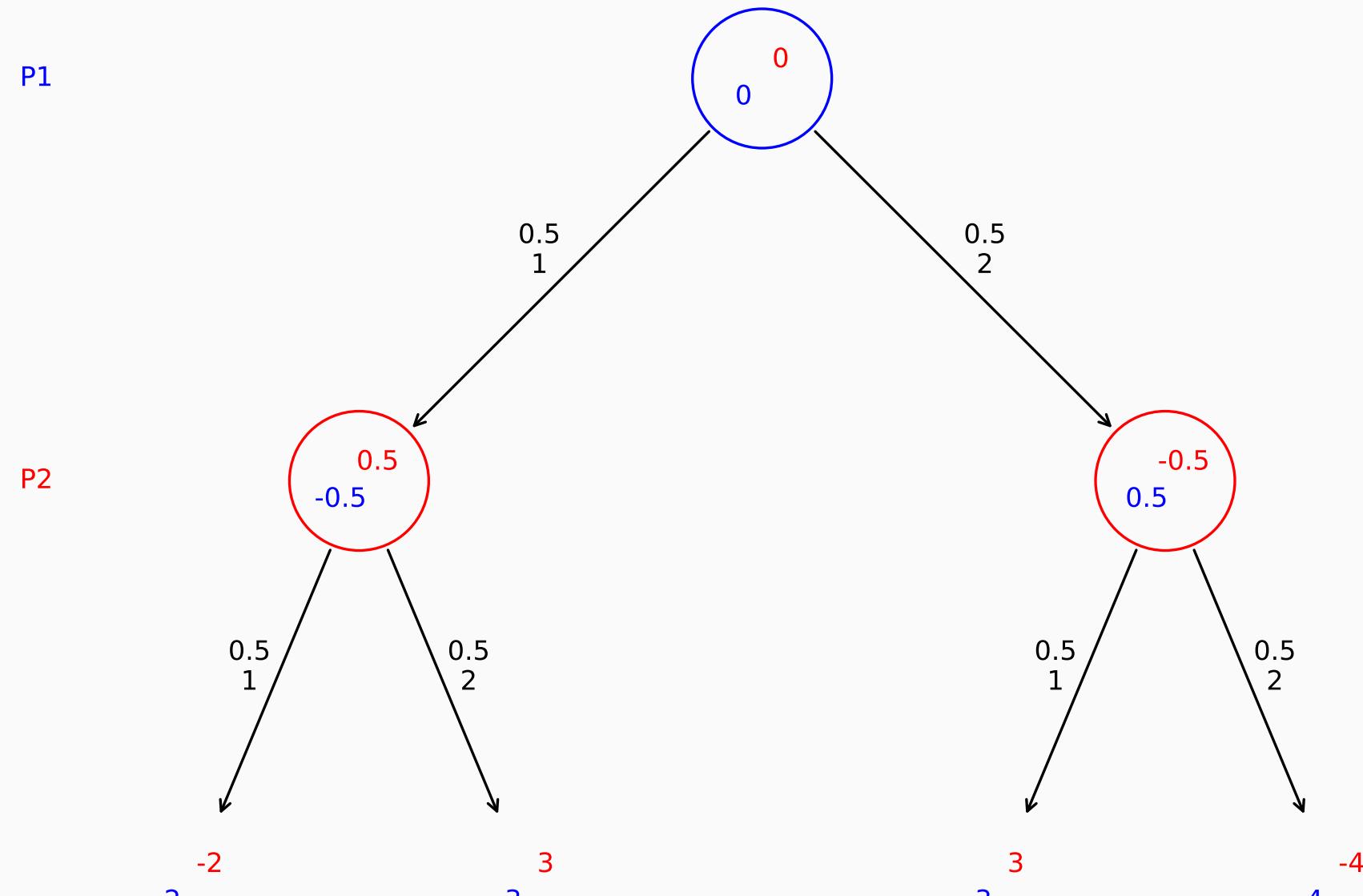
How can P1 exploit P2's strategy? Choose 2 more often than 1

Two-finger Morra



How can P2 play such that P1 cannot exploit P2?

Two-finger Morra, P2's mixed strategy

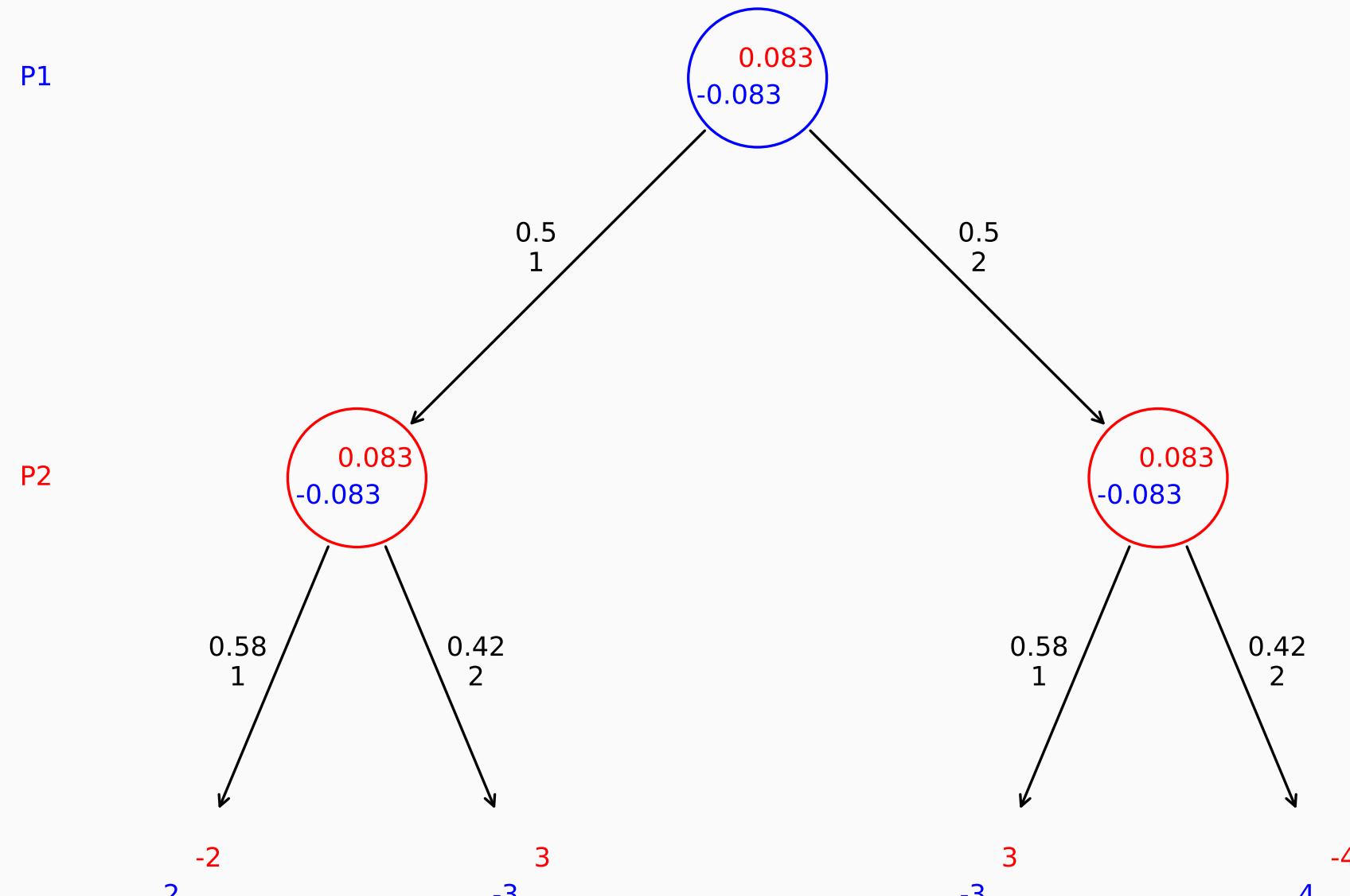


Mixed strategy for P2 with probability π_2 of choosing 1, where the expected values of P1 in the red nodes are set equal:

$$2\pi_2 - 3(1 - \pi_2) = -3\pi_2 + 4(1 - \pi_2)$$

$$\pi_2 = \frac{7}{12}$$

Two-finger Morra, P2's mixed strategy

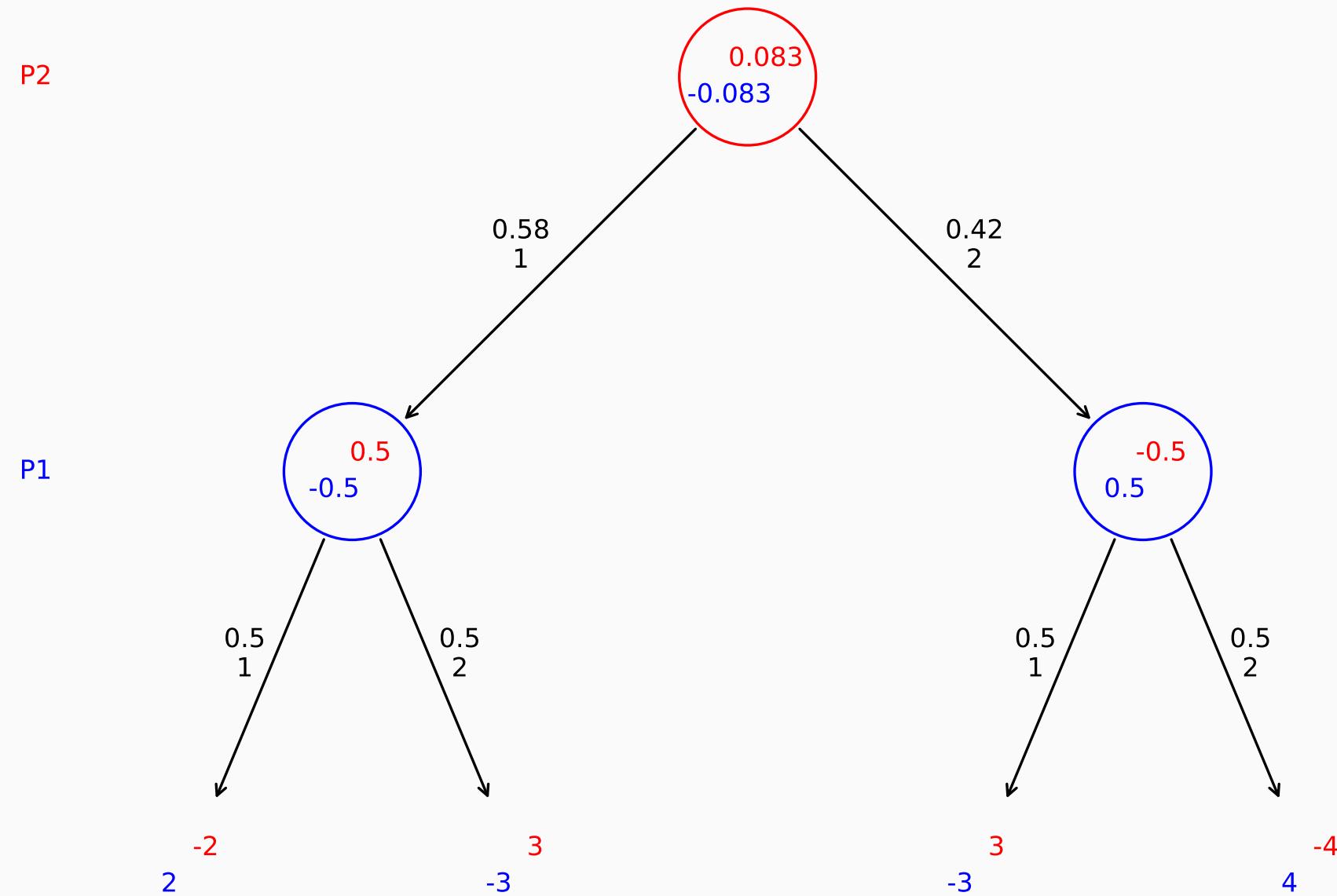


Mixed strategy for P2 with probability π_2 of choosing 1, where the expected values of P1 in the red nodes are set equal:

$$2\pi_2 - 3(1 - \pi_2) = -3\pi_2 + 4(1 - \pi_2)$$

$$\pi_2 = \frac{7}{12}$$

Two-finger Morra, P1's mixed strategy

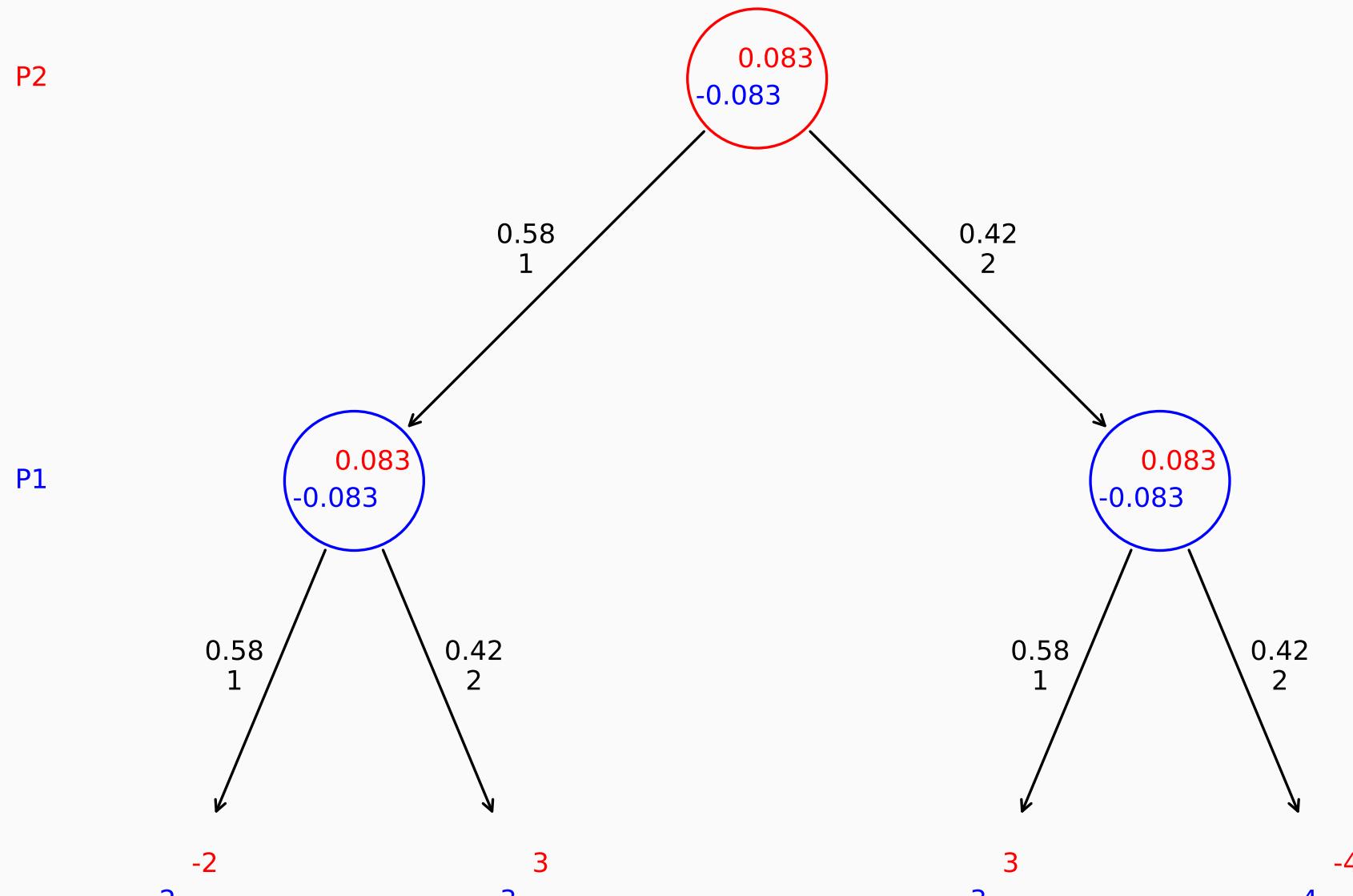


Mixed strategy for P1 with probability π_1 of choosing 1, where the expected values of P2 in the blue nodes are set equal:

$$-2\pi_1 + 3(1 - \pi_1) = 3\pi_1 - 4(1 - \pi_1)$$

$$\pi_1 = \frac{7}{12}$$

Two-finger Morra, P1's mixed strategy

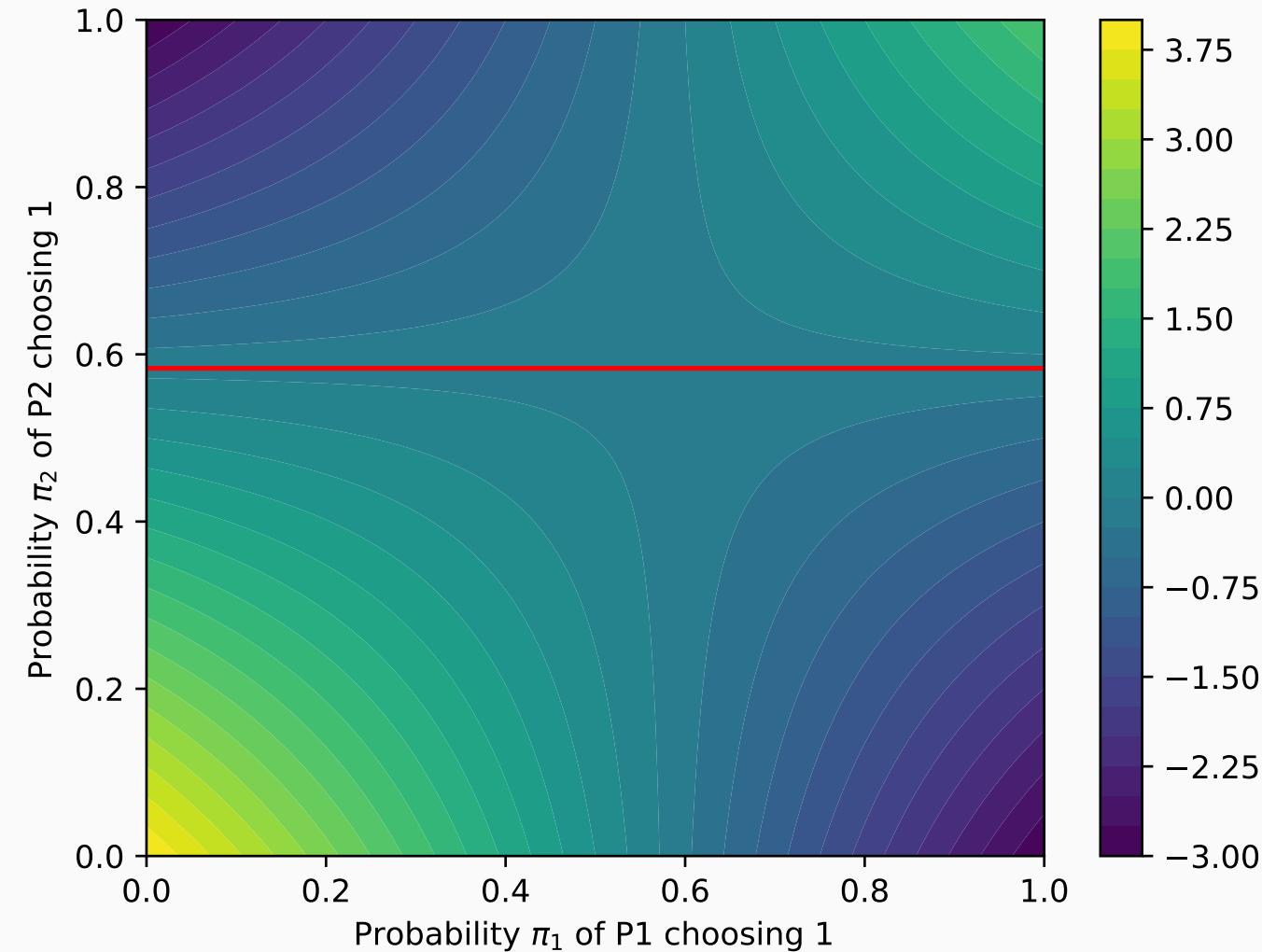


Mixed strategy for P1 with probability π_1 of choosing 1, where the expected values of P2 in the blue nodes are set equal:

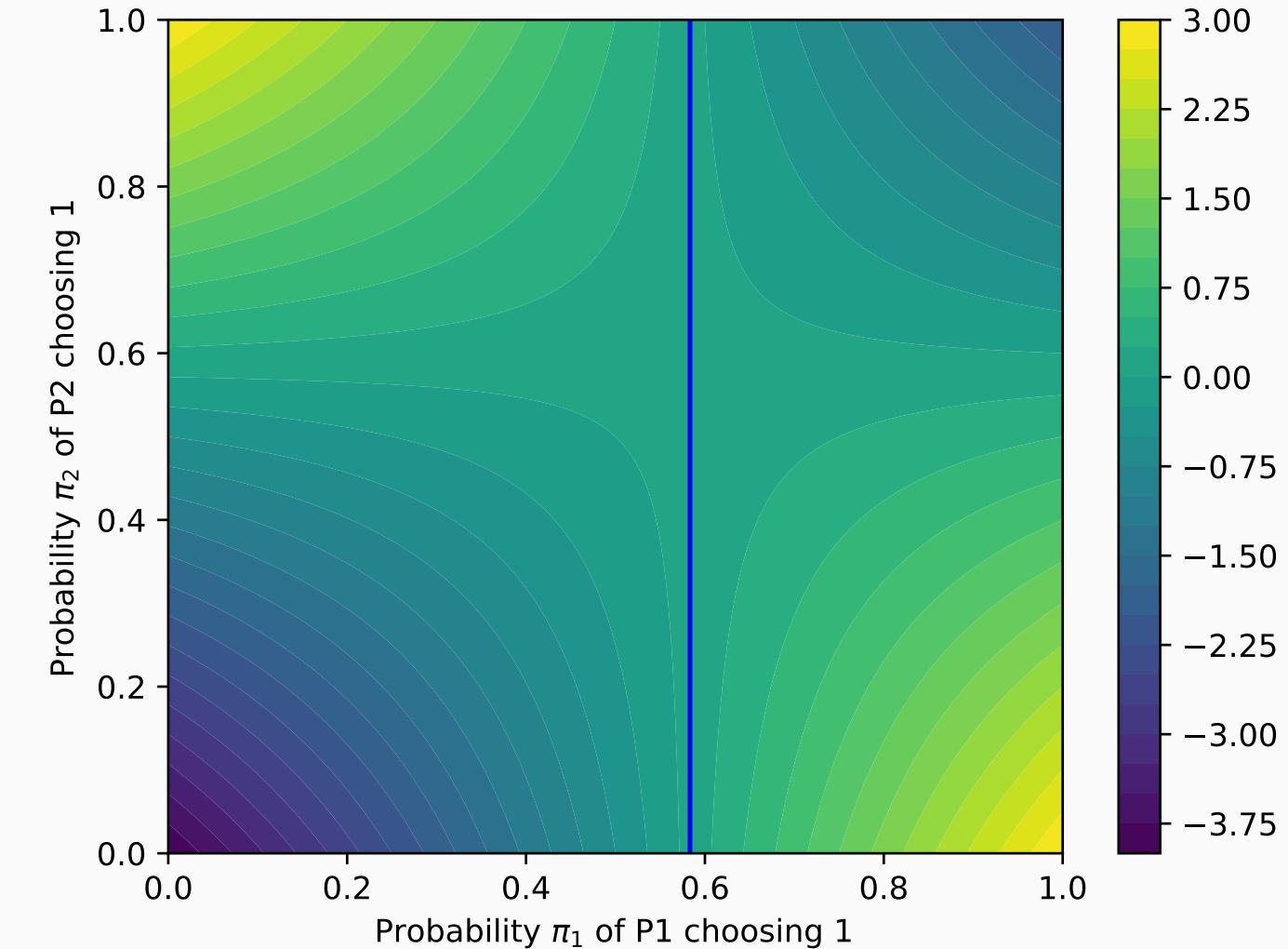
$$-2\pi_1 + 3(1 - \pi_1) = 3\pi_1 - 4(1 - \pi_1)$$

$$\pi_1 = \frac{7}{12}$$

Two-finger Morra, expected values of every π_1 and π_2



Expected values for P1, red line shows
mixed strategy for P2



Expected values for P2, blue line shows
mixed strategy for P1

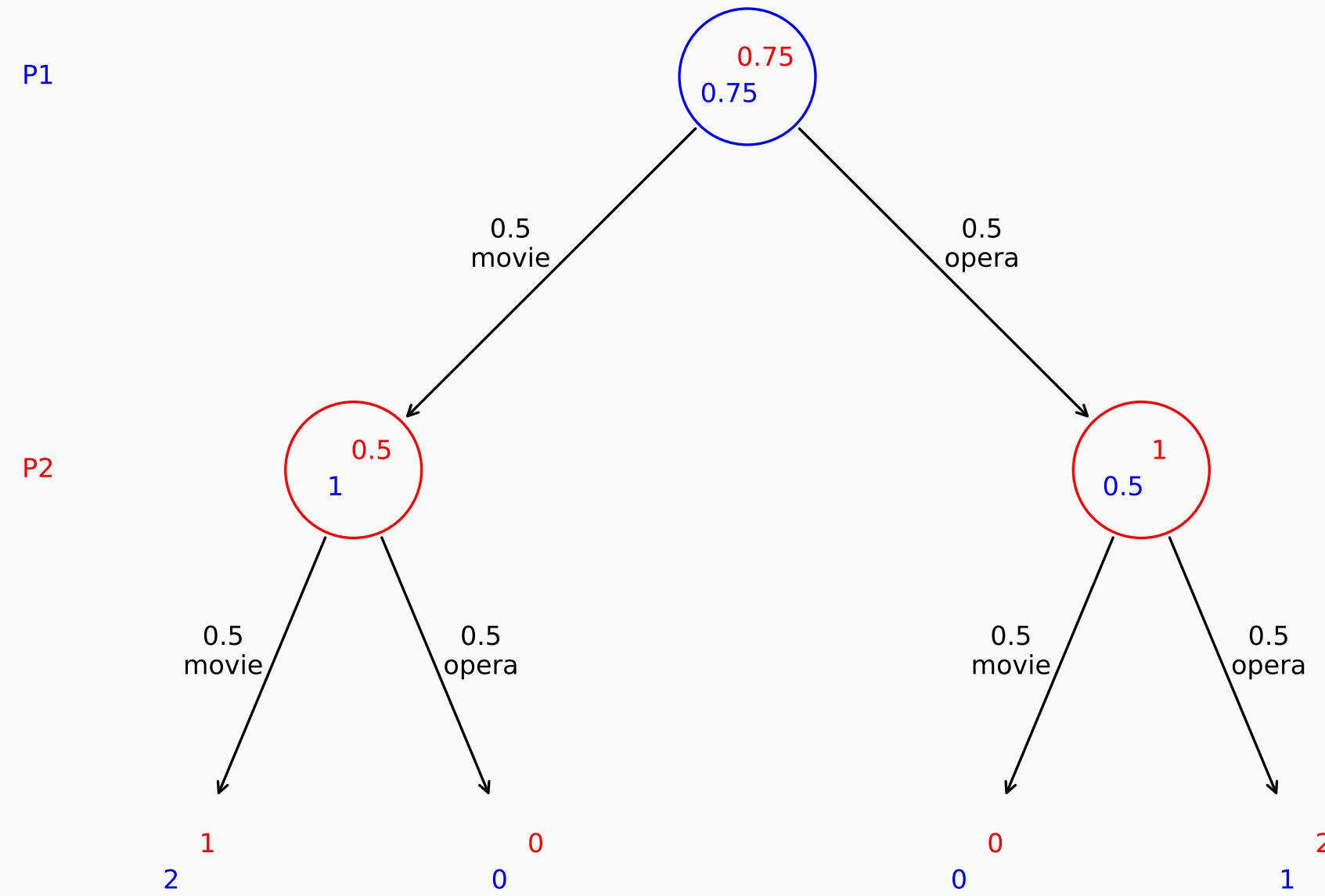
Battle of the Sexes

		P2	
		movie	opera
P1	movie	1 2	0
	opera	0	2 1

Not a zero-sum game

Two pure strategy Nash equilibria: P1 and P2 choose movie, and P1 and P2 choose opera

Battle of the Sexes, P2's mixed strategy

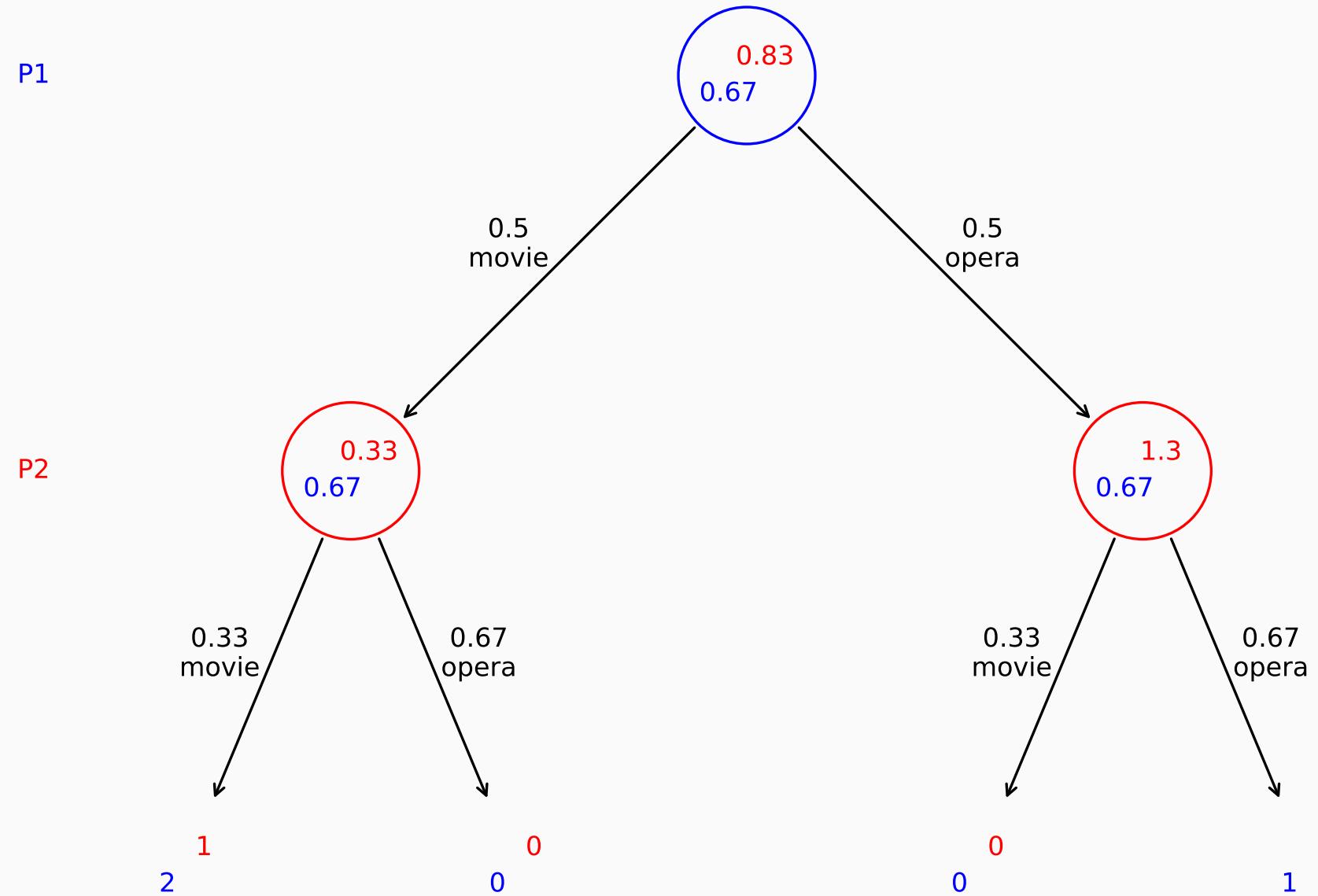


Mixed strategy for P2 with probability π_2 of choosing movie, where the expected values of P1 in the red nodes are set equal:

$$2\pi_2 + 0(1 - \pi_2) = 0\pi_2 + 1(1 - \pi_2)$$

$$\pi_2 = \frac{1}{3}$$

Battle of the Sexes, P2's mixed strategy

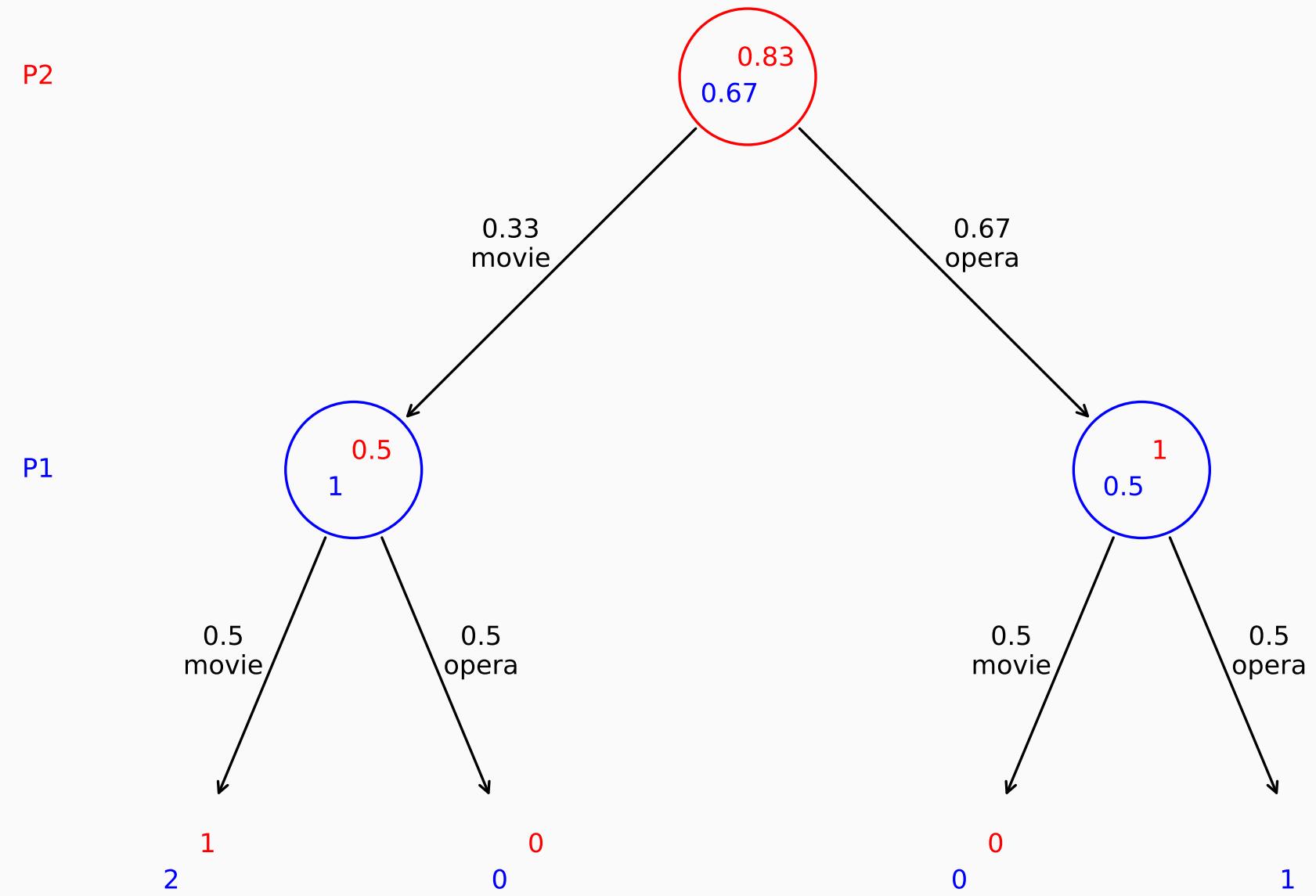


Mixed strategy for **P2** with probability π_2 of choosing movie, where the expected values of **P1** in the red nodes are set equal:

$$2\pi_2 + 0(1 - \pi_2) = 0\pi_2 + 1(1 - \pi_2)$$

$$\pi_2 = \frac{1}{3}$$

Battle of the Sexes, P1's mixed strategy

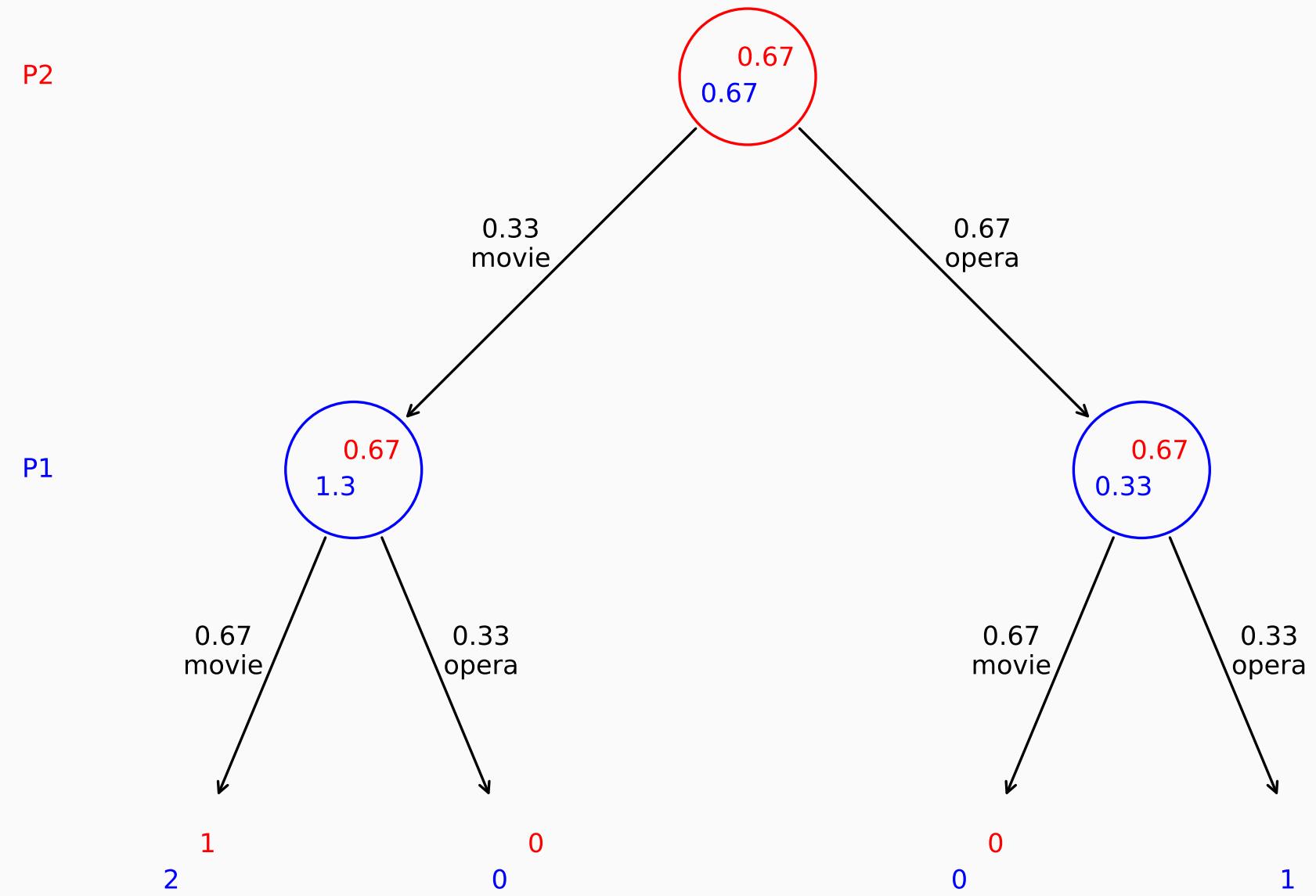


Mixed strategy for P1 with probability π_1 of choosing movie, where the expected values of P2 in the blue nodes are set equal:

$$1\pi_1 + 0(1 - \pi_1) = 0\pi_1 + 2(1 - \pi_1)$$

$$\pi_1 = \frac{2}{3}$$

Battle of the Sexes, P1's mixed strategy

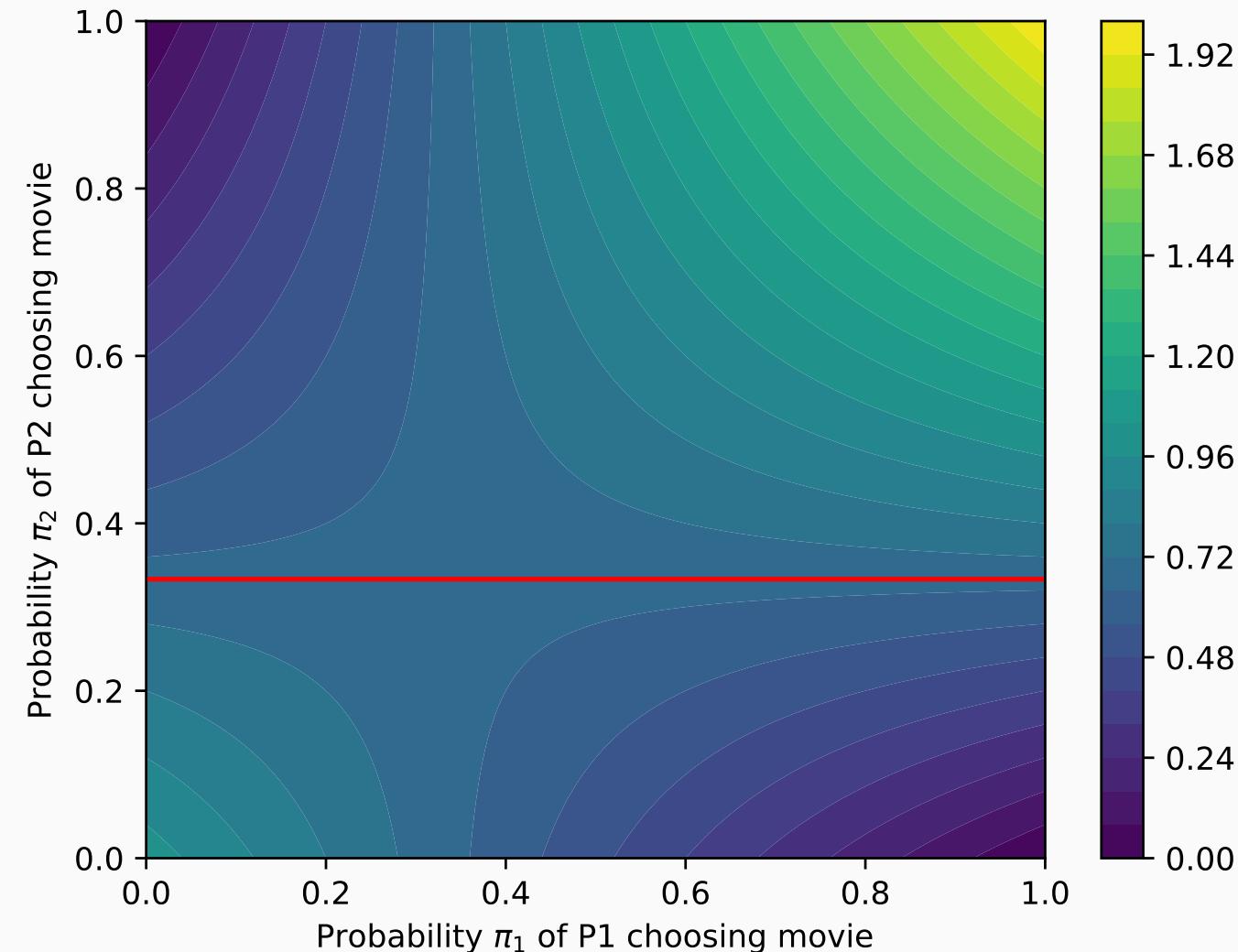


Mixed strategy for P1 with probability π_1 of choosing movie, where the expected values of P2 in the blue nodes are set equal:

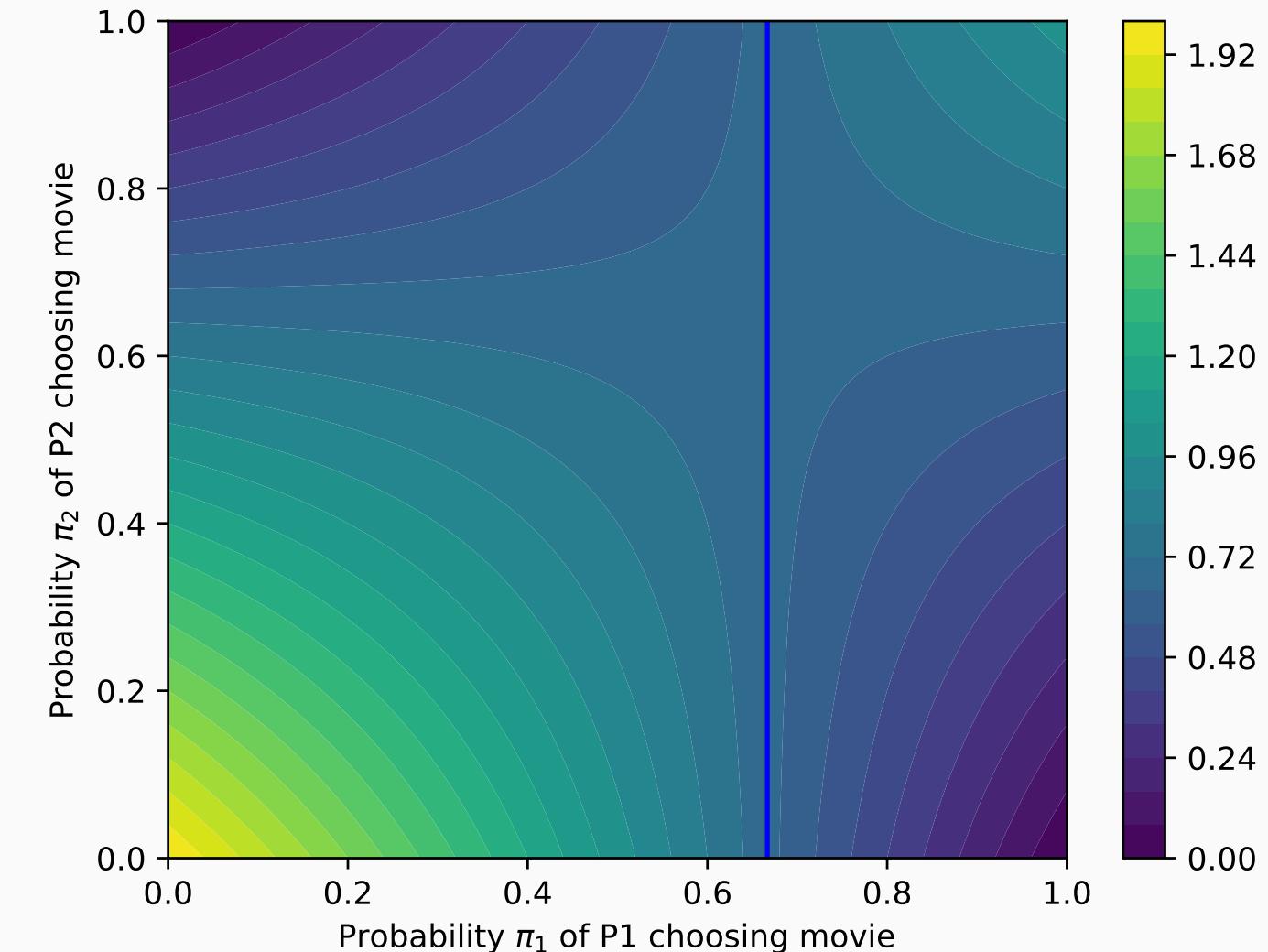
$$1\pi_1 + 0(1 - \pi_1) = 0\pi_1 + 2(1 - \pi_1)$$

$$\pi_1 = \frac{2}{3}$$

Battle of the Sexes, expected values of every π_1 and π_2



Expected values for P1, red line shows
mixed strategy for P2



Expected values for P2, blue line shows
mixed strategy for P1

Outline

Mixed strategies

Pure strategies

Repeated games

Infinite rounds

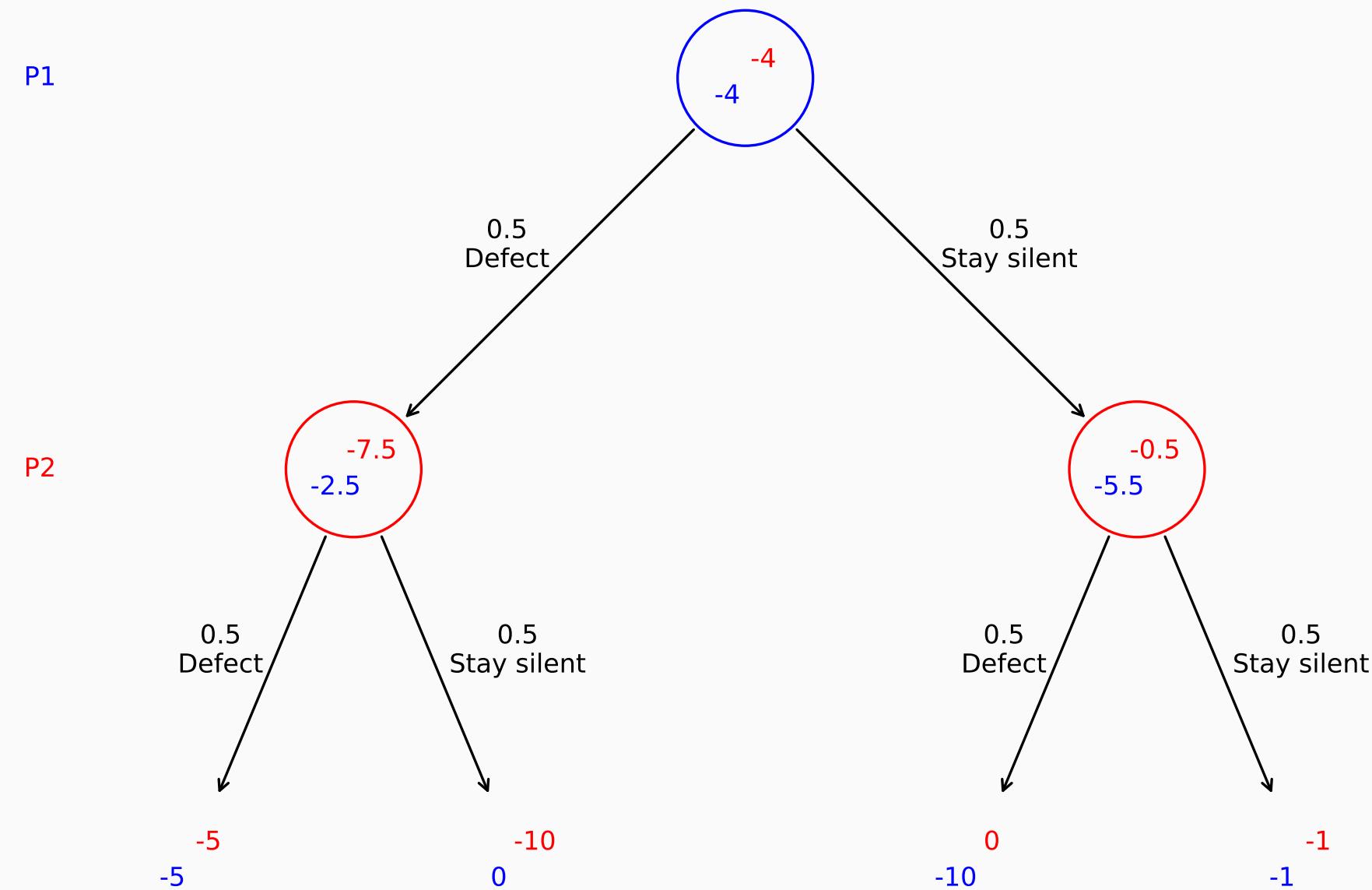
Unknown number of rounds

Prisoner's dilemma

		P2
	Defect	Stay silent
Defect	-5	-10
Stay silent	-5	0
	0	-1
	-10	-1

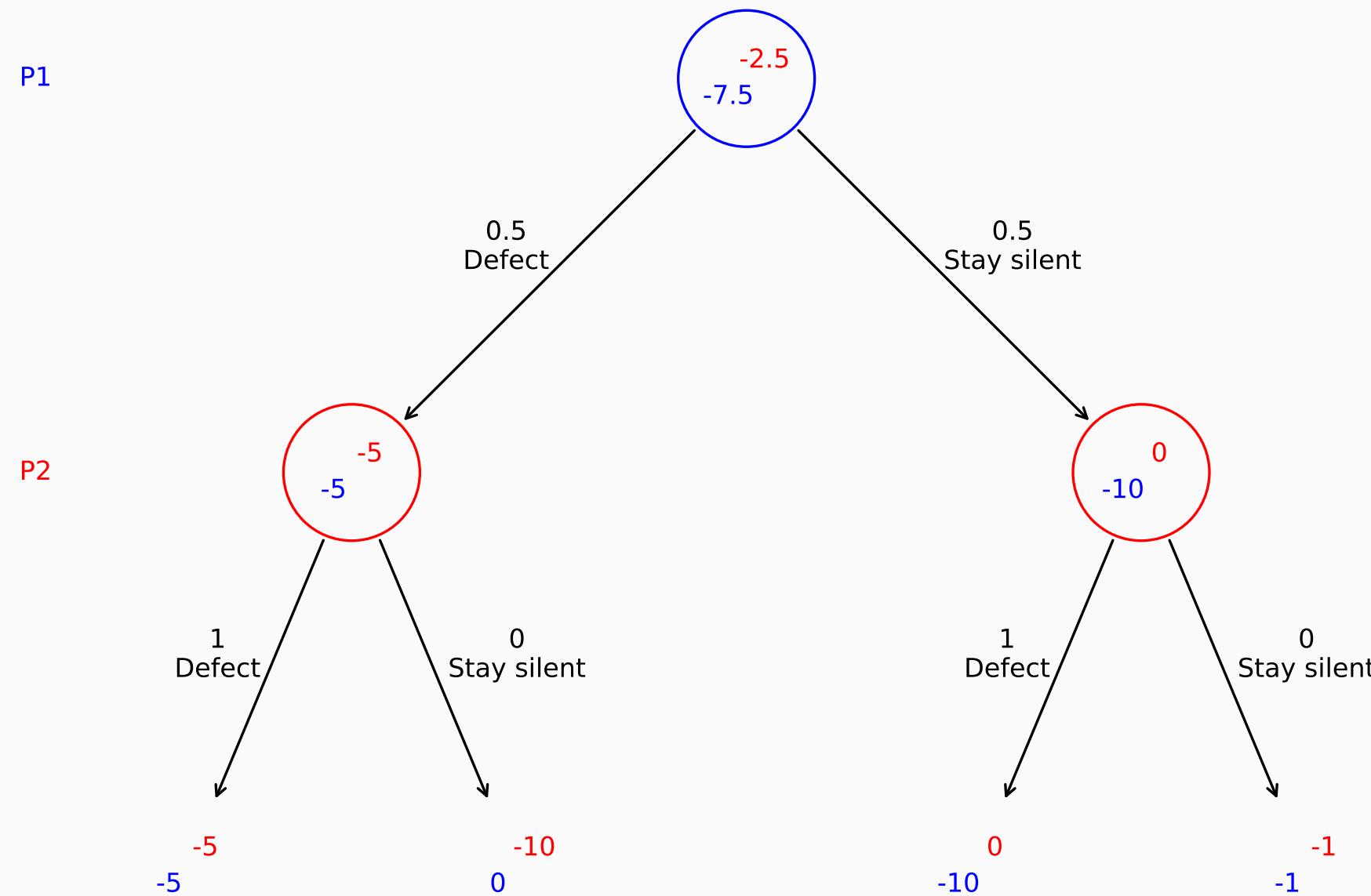
Has no mixed strategy (shown on page 18)

Prisoner's dilemma



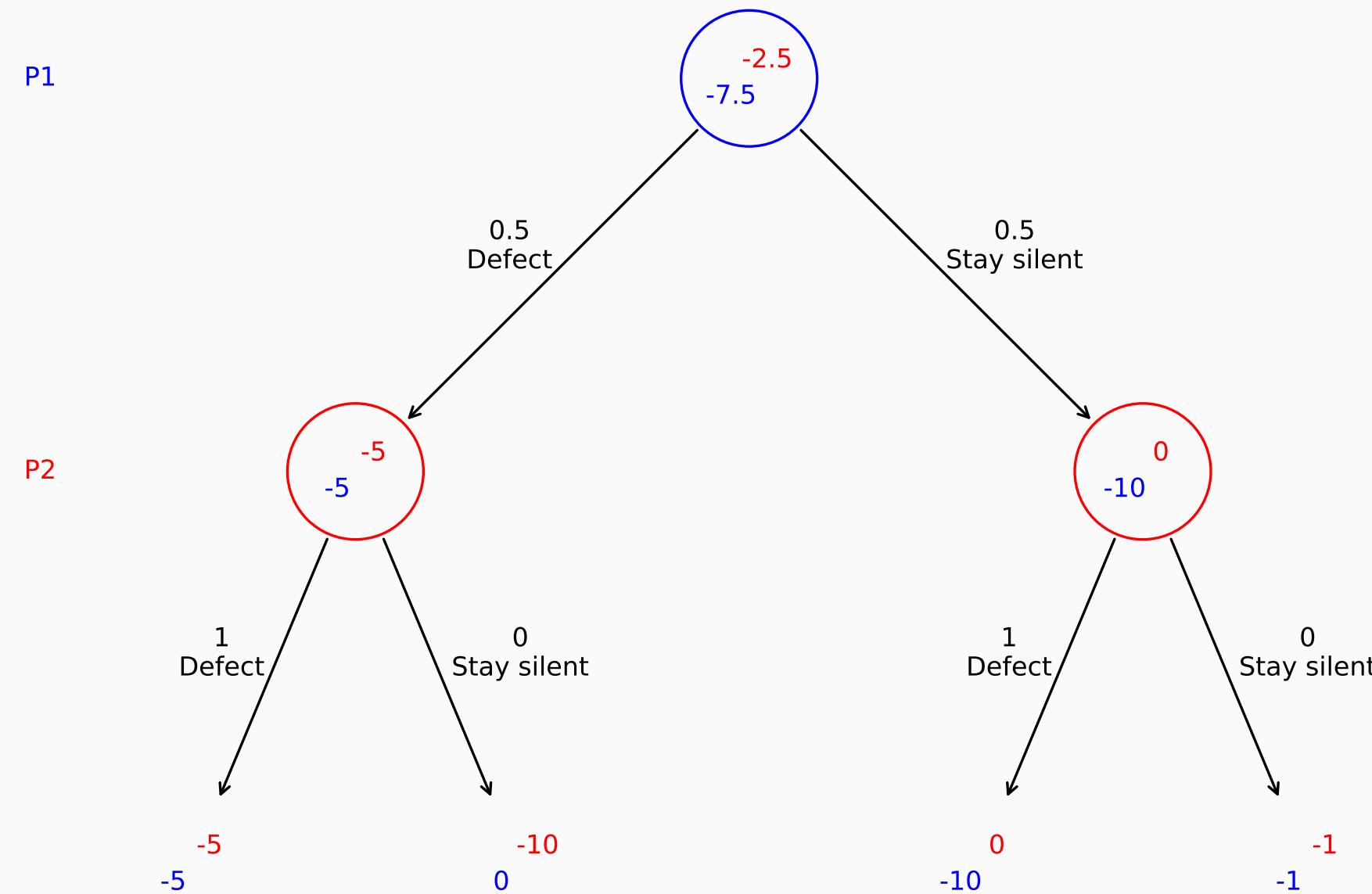
What is P2's best strategy?

Prisoner's dilemma, P2's pure strategy



What is P2's best strategy? Defect

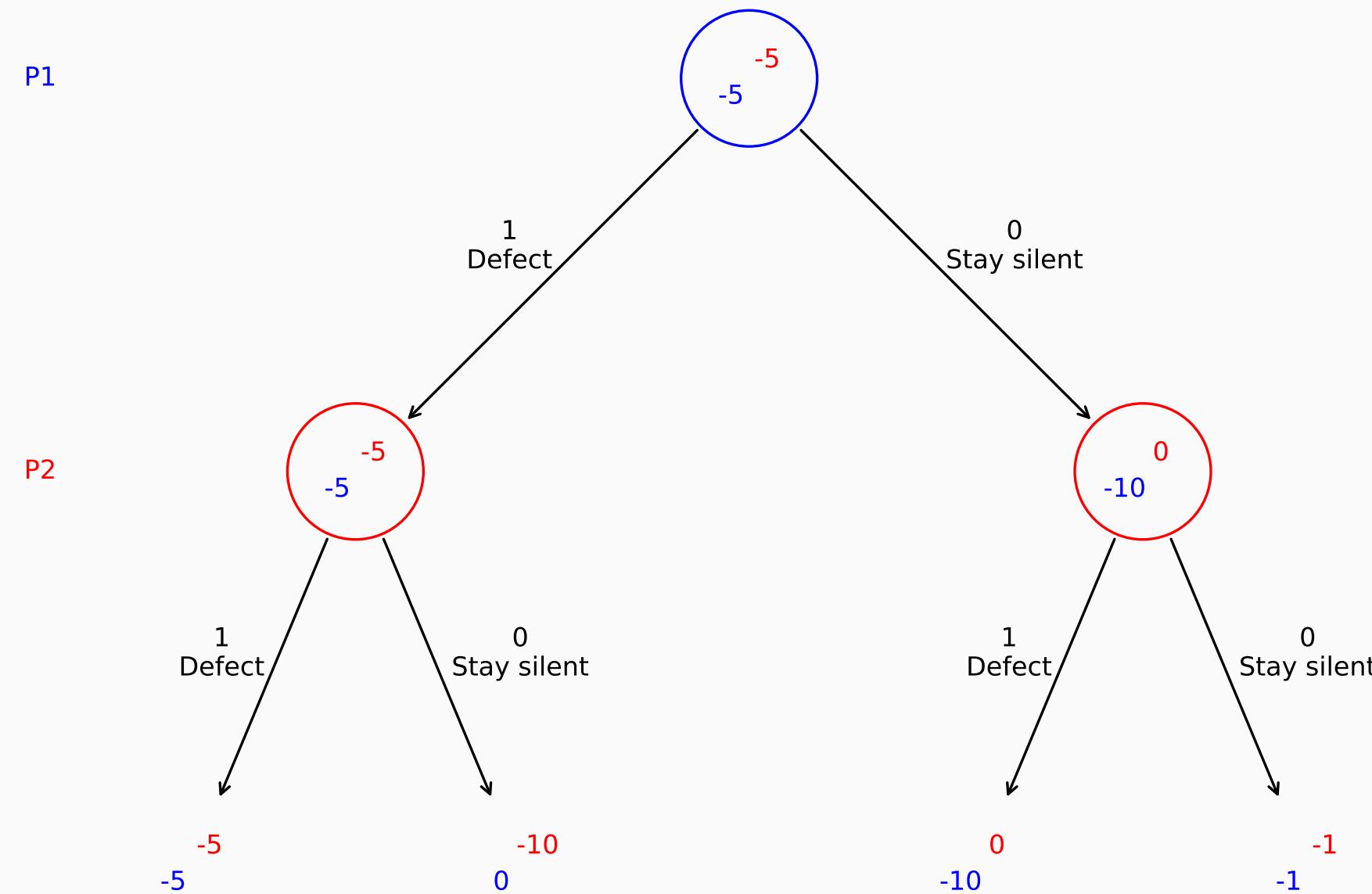
Prisoner's dilemma, P2's pure strategy



What is P2's best strategy? Defect

What is P1's best strategy?

Prisoner's dilemma, P1's and P2's pure strategies



What is P2's best strategy? Defect

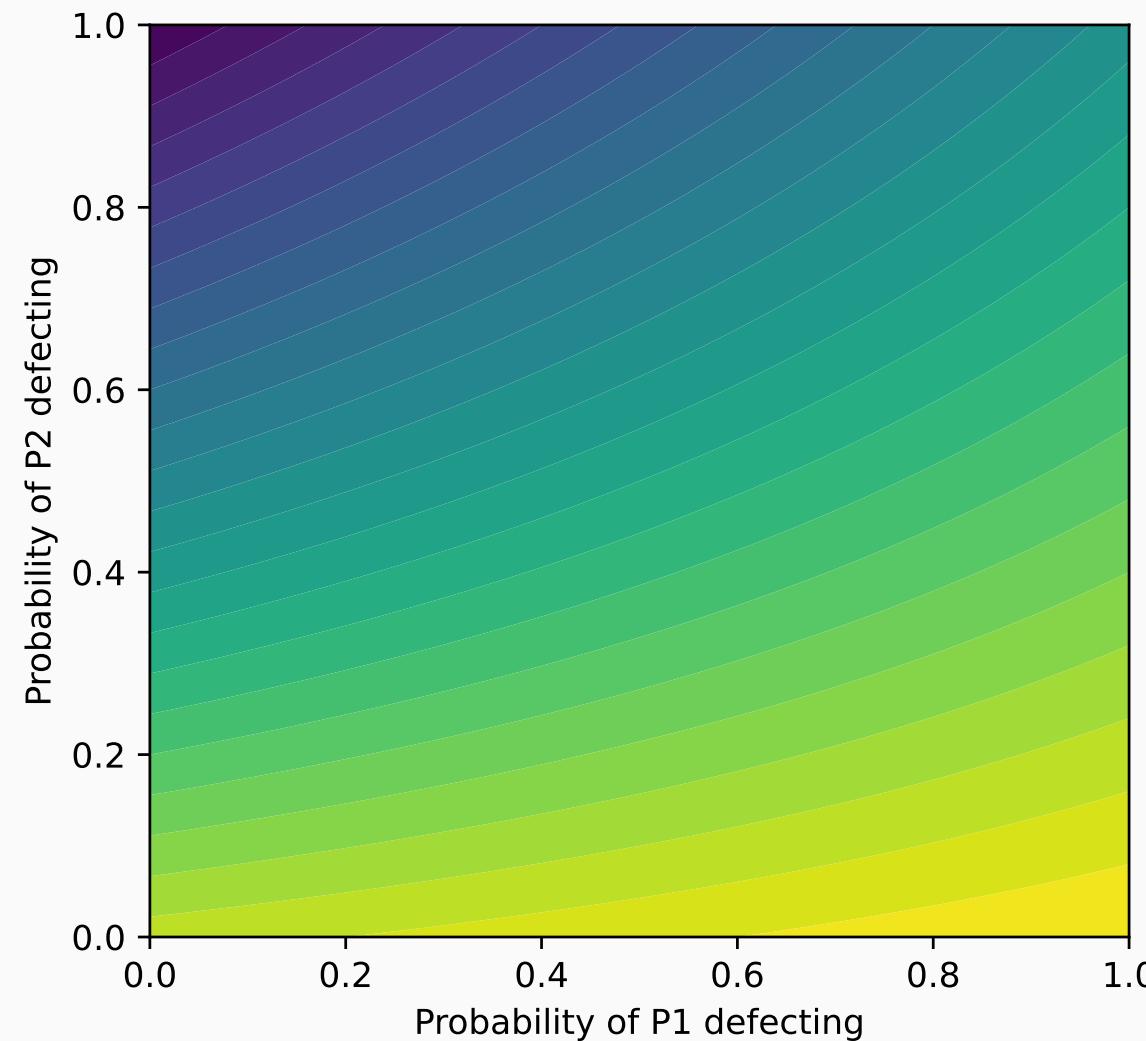
What is P1's best strategy? Defect

Prisoner's dilemma

		P2	
		Defect	Stay silent
		-5	-10
P1	Defect	-5	0
	Stay silent	0	-1
		-10	-1

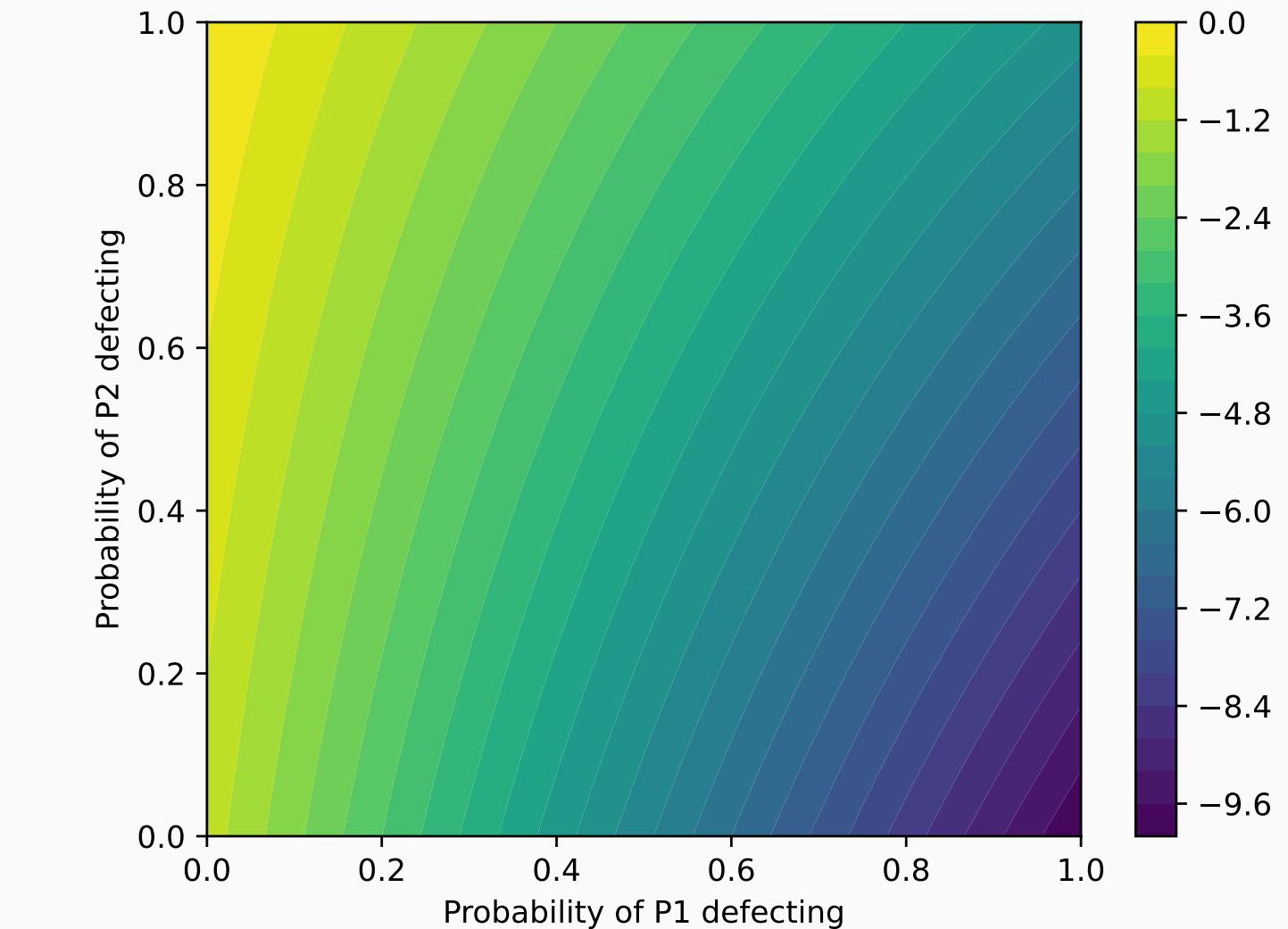
Defecting is a pure strategy Nash equilibrium

Prisoner's dilemma, expected values of every π_1 and π_2



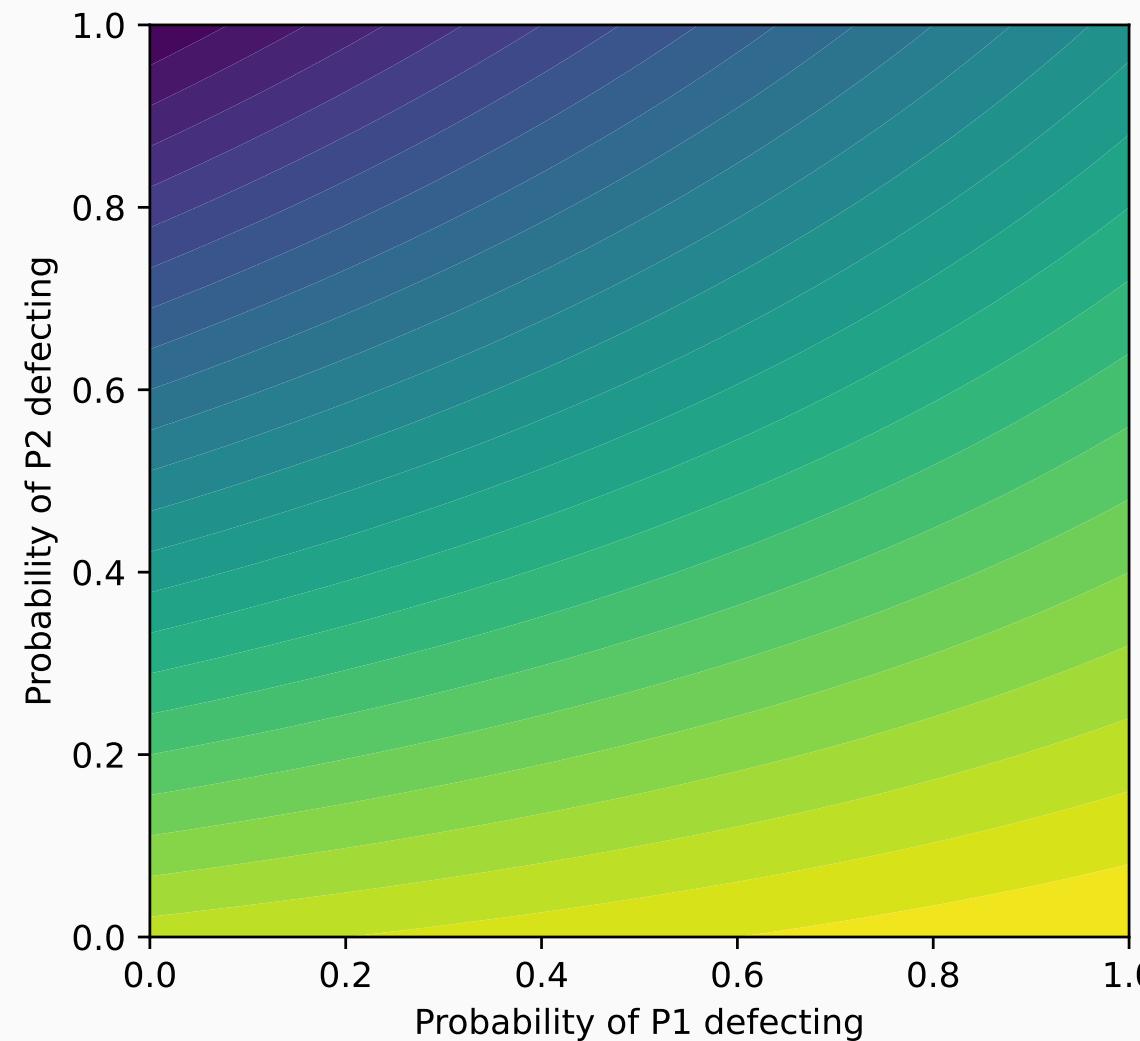
Expected values for $P1$

No mixed strategies. Why?

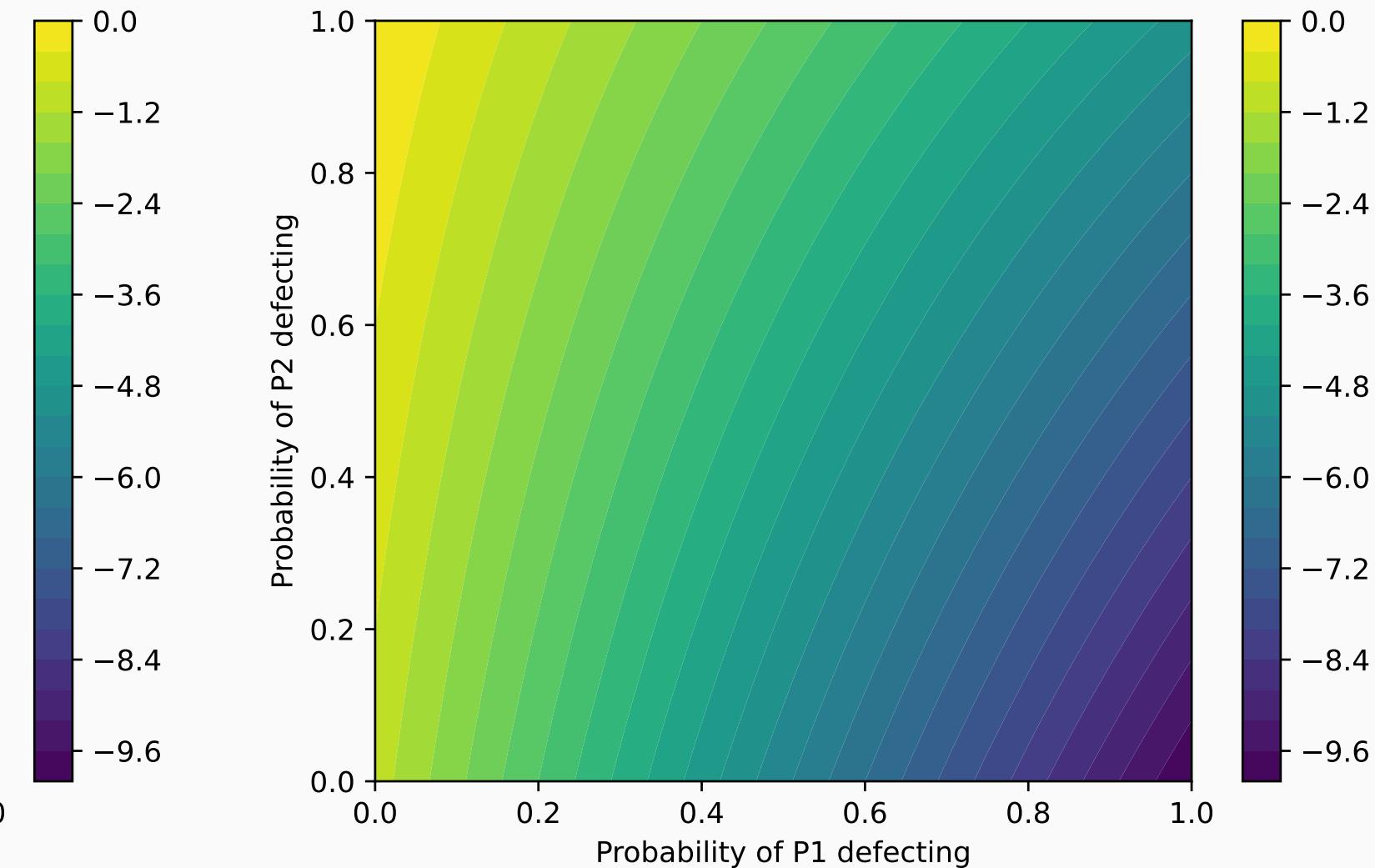


Expected values for $P2$

Prisoner's dilemma, expected values of every π_1 and π_2



Expected values for $P1$



Expected values for $P2$

No mixed strategies. Why? Expected values are not constant on any horizontal or vertical line over $\pi_1 \in [0, 1]$ and $\pi_2 \in [0, 1]$

Outline

Mixed strategies

Pure strategies

Repeated games

Infinite rounds

Unknown number of rounds

Backward induction: when the number of rounds is fixed, finite and known

- Players will play 10 rounds of prisoner's dilemma
 - What will the players do?

		P2								
	Defect	Stay silent								
P1	Defect	<table border="1"><tr><td>-5</td><td>-10</td></tr><tr><td>-5</td><td>0</td></tr><tr><td>0</td><td>-1</td></tr><tr><td>-10</td><td>-1</td></tr></table>	-5	-10	-5	0	0	-1	-10	-1
-5	-10									
-5	0									
0	-1									
-10	-1									
	Stay silent									

Backward induction: when the number of rounds is fixed, finite and known

- Players will play 10 rounds of prisoner's dilemma
 - What will the players do?
 - What will the players do at the tenth round?

		P2								
	Defect	Stay silent								
P1	Defect	<table border="1"><tr><td>-5</td><td>-10</td></tr><tr><td>-5</td><td>0</td></tr><tr><td>0</td><td>-1</td></tr><tr><td>-10</td><td>-1</td></tr></table>	-5	-10	-5	0	0	-1	-10	-1
-5	-10									
-5	0									
0	-1									
-10	-1									
	Stay silent									

Backward induction: when the number of rounds is fixed, finite and known

- Players will play 10 rounds of prisoner's dilemma
 - What will the players do?
 - What will the players do at the tenth round?
 - Defect

	P2									
	Defect	Stay silent								
P1	Defect	<table border="1"><tr><td>-5</td><td>-10</td></tr><tr><td>-5</td><td>0</td></tr><tr><td>0</td><td>-1</td></tr><tr><td>-10</td><td>-1</td></tr></table>	-5	-10	-5	0	0	-1	-10	-1
-5	-10									
-5	0									
0	-1									
-10	-1									
	Stay silent									

Backward induction: when the number of rounds is fixed, finite and known

- Players will play 10 rounds of prisoner's dilemma
 - What will the players do?
 - What will the players do at the tenth round?
 - Defect
 - What will the players do at the ninth round?

		P2	
		Stay silent	Defect
P1	Defect	-5	-10
	Stay silent	-5	0
	Defect	0	-1
	Stay silent	-10	-1

Backward induction: when the number of rounds is fixed, finite and known

- Players will play 10 rounds of prisoner's dilemma
 - What will the players do?
 - What will the players do at the tenth round?
 - Defect
 - What will the players do at the ninth round?
 - Defect

		P2	
		Stay silent	Defect
P1	Defect	-5	-10
	Stay silent	-5	0
	Defect	0	-1
	Stay silent	-10	-1

Backward induction: when the number of rounds is fixed, finite and known

- Players will play 10 rounds of prisoner's dilemma
 - What will the players do?
 - What will the players do at the tenth round?
 - Defect
 - What will the players do at the ninth round?
 - Defect
 - ...
 - What will the players do at the first round?
 - Defect

		P2	
		Defect	Stay silent
P1	Defect	-5	-10
	Stay silent	-5	0
	Defect	0	-1
	Stay silent	-10	-1

Outline

Mixed strategies

Pure strategies

Repeated games

Infinite rounds

Unknown number of rounds

When the number of rounds is infinite

We cannot use the sum of the expected values of each game:

$$\sum_{t=0}^{\infty} E_p(\pi_1(s_t), \pi_2(s_t))$$

where state s_t contains all the previous actions prior to time t

For example, both the strategies

$\pi_1(s_t) = \pi_2(s_t) = 1 \forall s_t$ (always defect) and

$\pi_1(s_t) = \pi_2(s_t) = 0 \forall s_t$ (always stay silent) would

then get the sum $-\infty$

		P2
		Stay silent
		Defect
P1	Defect	-5
	Stay silent	-10
P2	Defect	-5
	Stay silent	0
P1	Defect	0
	Stay silent	-1
P2	Defect	-10
	Stay silent	-1

When the number of rounds is infinite

However, we can use a discount factor $\delta \in (0, 1)$:

$$\sum_{t=0}^{\infty} \delta^t E_p(\pi_1(s_t), \pi_2(s_t))$$

where a small δ will lead to discounting all but the first few games

		P2
		Stay silent
		Defect
P1	Defect	-5
	Stay silent	-10
P2	Defect	0
	Stay silent	-1

Grim trigger strategy

Let the state s_t be defined as all the previous actions:

$$s_t = ((a_{1,0}, a_{2,0}), (a_{1,1}, a_{2,1}), \dots, (a_{1,t-1}, a_{2,t-1}))$$

where $a_{1,0}, a_{2,0}$ are player 1's and player 2's actions (1: defect, 0: stay silent) at the first round

The grim trigger strategy for example for P1 is then:

$$\pi_1(s_t) = \begin{cases} 1, & \text{if } t > 0 \wedge \sum_{i=0}^{t-1} a_{2,i} > 0 \\ 0, & \text{otherwise} \end{cases}$$

i.e. defect if the opponent has defected previously, otherwise stay silent

		P2
		Stay silent
		Defect
P1	Defect	-5
	Stay silent	-10
P2	Defect	0
	Stay silent	-1

Grim trigger strategy: Nash equilibrium

If both players follow the grim trigger strategies, the discount factor determines if for example player 1 has no incentive to change his/her strategy:

$$\sum_{t=0}^{\infty} \delta^t E_1(0, 0) \geq \delta^0 E_1(1, 0) + \sum_{t=1}^{\infty} \delta^t E_1(1, 1)$$

$$-1\delta^0 - 1\delta^1 - 1\delta^2 - \dots \geq 0\delta^0 - 5\delta^1 - 5\delta^2 - 5\delta^3 - \dots$$

$$-\frac{1}{1-\delta} \geq 5 - \frac{5}{1-\delta}$$

$$\delta \geq \frac{1}{5}$$

i.e. if both players follow the grim strategy and use discount factor $\delta \geq \frac{1}{5}$, they are in a Nash equilibrium

		P2	
		Stay silent	Defect
P1	Defect	-5	-10
	Stay silent	0	-5
	Defect	0	-1
	Stay silent	-10	-1

Outline

Mixed strategies

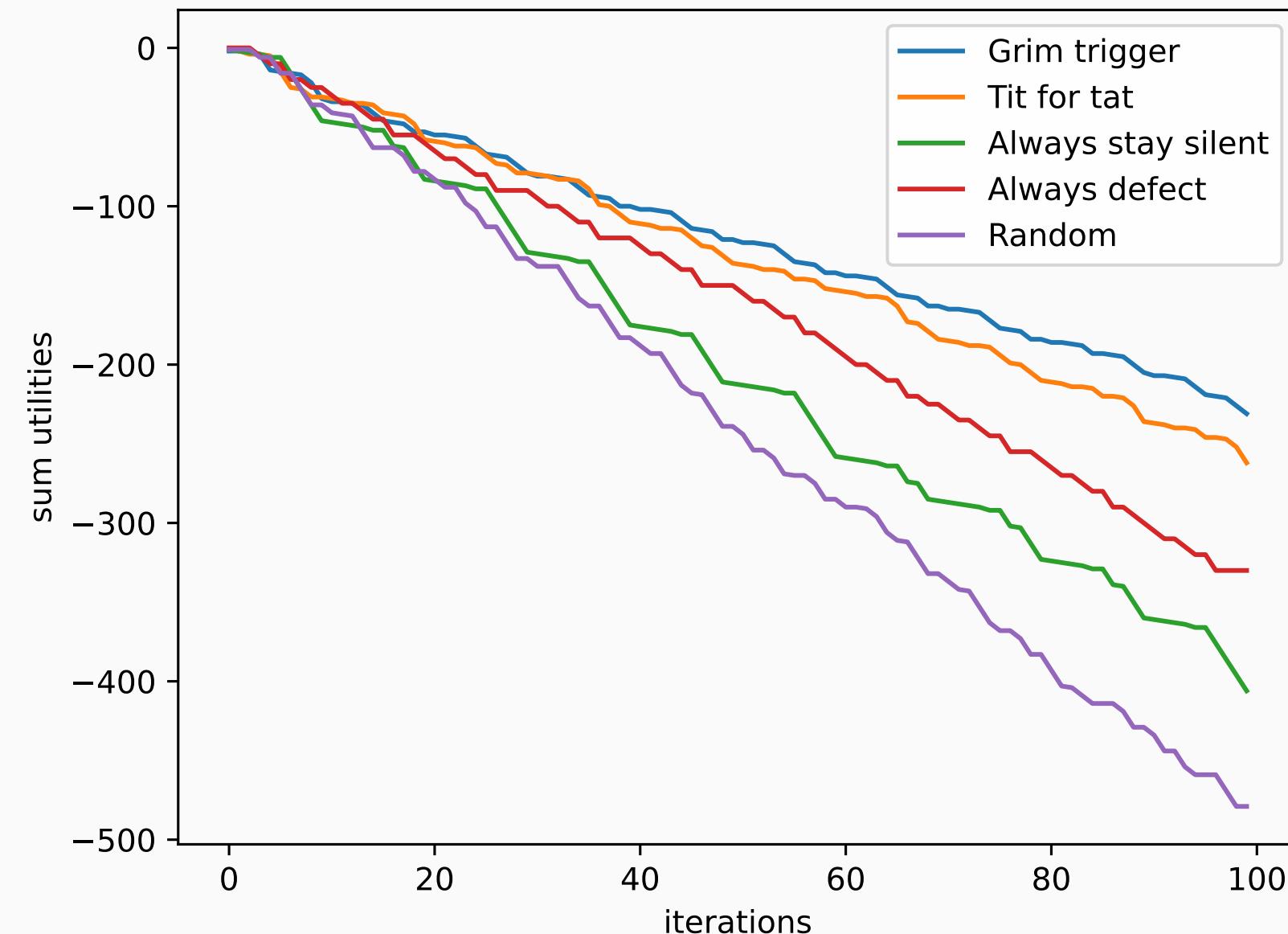
Pure strategies

Repeated games

Infinite rounds

Unknown number of rounds

Strategy competition: search for optimal strategy, unknown number of rounds



The tit for tat strategy is to repeat the opponent's last action, for example for P1:

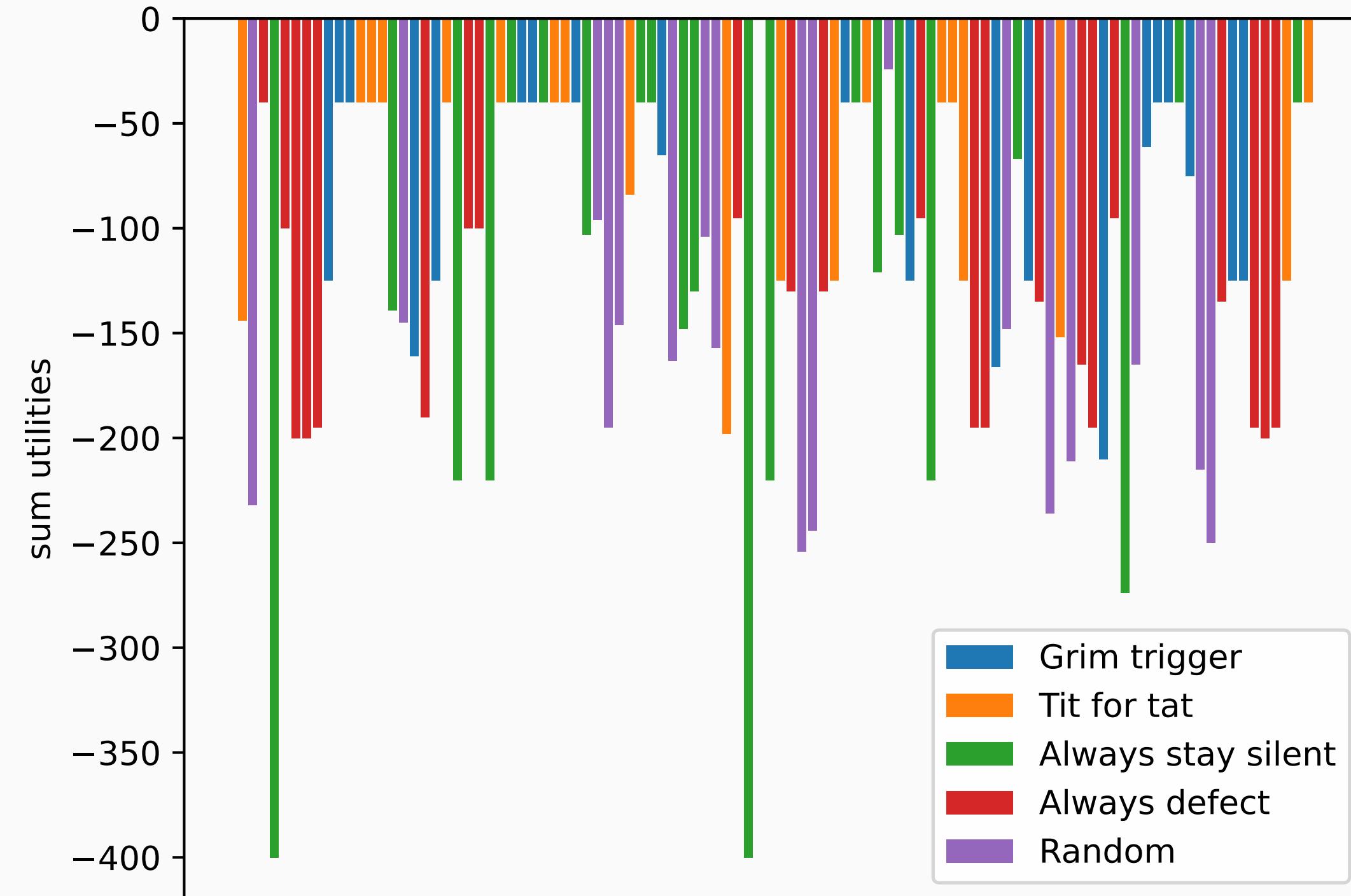
$$\pi_1(s_t) = \begin{cases} 0, & \text{if } t = 0 \\ a_{2,t-1}, & \text{otherwise} \end{cases}$$

Axelrod's Tournament

“In 1980, Robert Axelrod, professor of political science at the University of Michigan, held a tournament of various strategies for the prisoner’s dilemma. He invited a number of well-known game theorists to submit strategies to be run by computers. In the tournament, programs played games against each other and themselves repeatedly. Each strategy specified whether to cooperate or defect based on the previous moves of both the strategy and its opponent.”

“The winner of Axelrod’s tournament was the TIT FOR TAT strategy.”

Strategy competition: search for optimal strategy, unknown number of rounds



Randomly placed agents are only playing against their immediate neighbors

Natural Language Processing (NLP) and Large Language Models (LLMs)

An Introduction – Autumn Semester 2024, Dr. Ahmed Abouzeid

This lecture is based on concepts from the book "*Introduction to Artificial Intelligence: A Modern Approach (4th Edition)*" by Stuart Russell and Peter Norvig.
We will be focusing on concepts from:

- Chapter 24: NLP and Language Models, Parsing, Tasks in NLP
- Chapter 25: Word Embeddings, State of the Art Models

Lecture Outline

- Introduction to NLP
 - Definition and scope
 - History and evolution of NLP
 - Key applications
- Fundamentals of NLP
 - Text preprocessing
 - Basic Language Models
 - Part-of-Speech (POS) tagging
 - Neural Network Approach (Word Embeddings) vs Statistical Approach
- Introduction to LLMs
 - A Quick Overview on the Transformer Architecture
 - Conducting Live Demo of a Transformer

NLP Definition

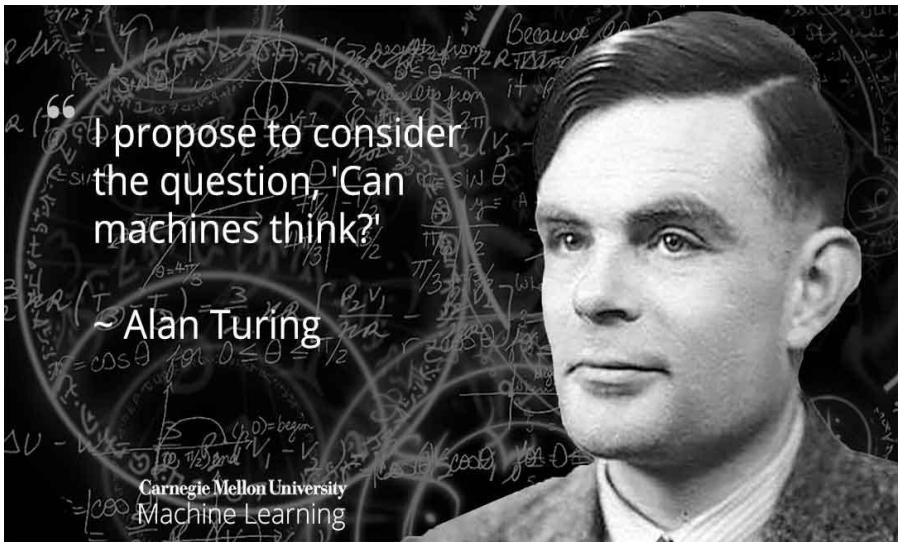
- NLP is a subfield in Artificial Intelligence (AI) and Computational Linguistics, focuses on:
 - The interaction between computers and humans' (natural) languages
 - Developing algorithms to interpret, and generate responses to/in human languages, which should be meaningful and useful



Figure: Natural Language Processing - Author: Seobility - License: [CC BY-SA 4.0](#)

History and Evolution of NLP

- The early beginnings (1950s – 1960s)
 - **1950:** Alan Turing proposes the Turing Test in "Computing Machinery and Intelligence."
 - **1954:** The Georgetown-IBM experiment demonstrates the first machine translation system
 - **1957:** Noam Chomsky publishes "Syntactic Structures," revolutionizing linguistic theory



I propose to consider
the question, 'Can
machines think?'

~Alan Turing

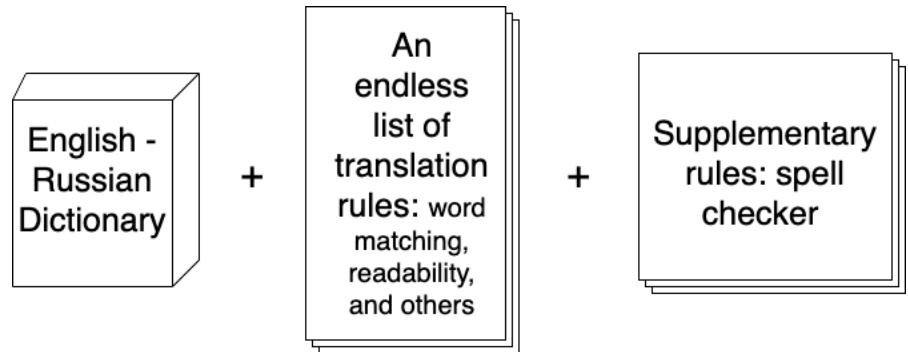
Carnegie Mellon University
Machine Learning

Check the full details of Alan's proposal [here](#)

History and Evolution of NLP

- Rule-Based Systems (1960s-1980s)

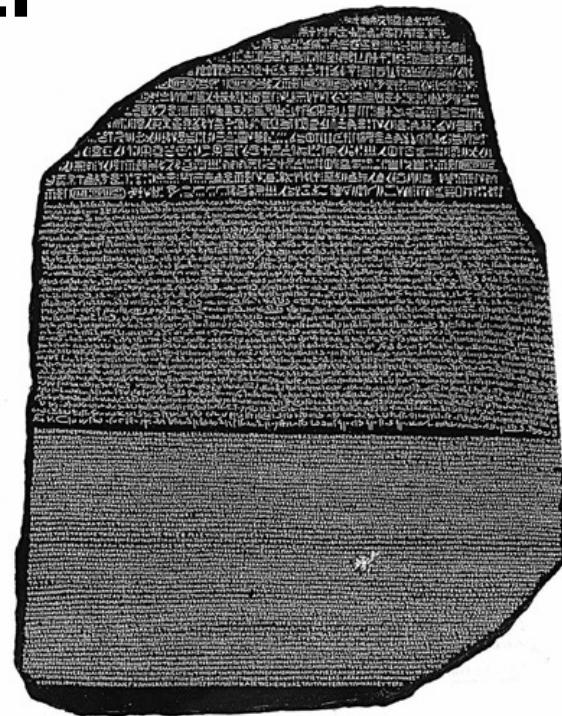
- **1960s-1970s:** Development of rule-based systems for machine translation and text processing
 - **1966:** The ALPAC report leads to reduced funding for MT research in the US
 - **1980:** Introduction of the PROLOG programming language, influencing NLP development



Machine-based translation - Rule-based system approach example

History and Evolution of NLP

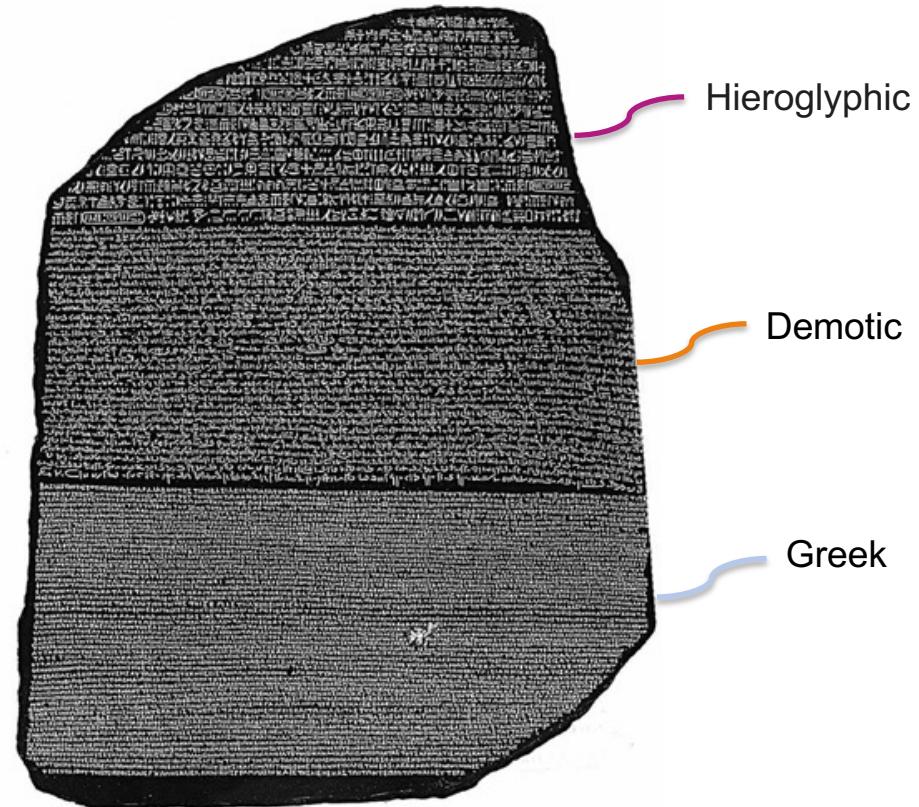
- Rise of Statistical Methods (1980s-1990s)
 - **1988:** Introduction of the first statistical machine translation models
 - **Early 1990s:** Hidden Markov Models (HMMs) become popular for speech recognition
 - **1996:** IBM's statistical translation model (Candide) gains prominence



The Rosetta Stone: An Inspiration for Statistical Machine Translation

Rise of NLP Statistical Methods (1980s-1990s)

- Humans need parallel text to understand a new language
- The Rosetta Stone allowed humans to understand the ancient Egyptian language
- Computers need the same type of information
- To build a statistical machine translation model, we need a parallel text dataset. E.g., English-French millions of examples for translated sentences (pairs)
- The model's main task is to calculate the statistical patterns between pair of words/ sentences by calculating the probabilities of occurrences



Rise of NLP Statistical Methods (1980s-1990s)

- Example of parallel corpus: Source language X (**French**), Target language Y (**English**)
- The task is to translate from X (**French**) to Y (**English**)
- Calculating the probabilities of occurrences:

$$= \operatorname{argmax}_y P(x|y) P(y)$$



Challenges in NLP Statistical Methods

- Many language pairs or specific phrases might not have enough examples, leading to poor translation quality
- Statistical models require extensive feature engineering, where human experts must identify and encode linguistic rules and patterns, which is time-consuming and prone to errors
- Statistical models are more complex, they require significant computational resources and become difficult to scale
- They are more prone to producing translations that are grammatically incorrect or awkward

History and Evolution of NLP

- Progress towards Deep Learning
(2000s-2010s)

- **2006:** Hinton developed Restricted Boltzmann Machines (RBMs)
 - **2010s:** Deep Learning Revolution



Figure: Neural Network imitating the function of human brain - Author: datascientest.com - License: [CC BY-SA 4.0](#)

History and Evolution of NLP

- Modern Advances (2020s)
 - **2020:** GPT-3, developed by OpenAI, showcases the power of large-scale language models
 - **2022:** Emergence of multimodal models combining text and image processing
 - **2024:** Ongoing research in ethical AI and bias mitigation in NLP.

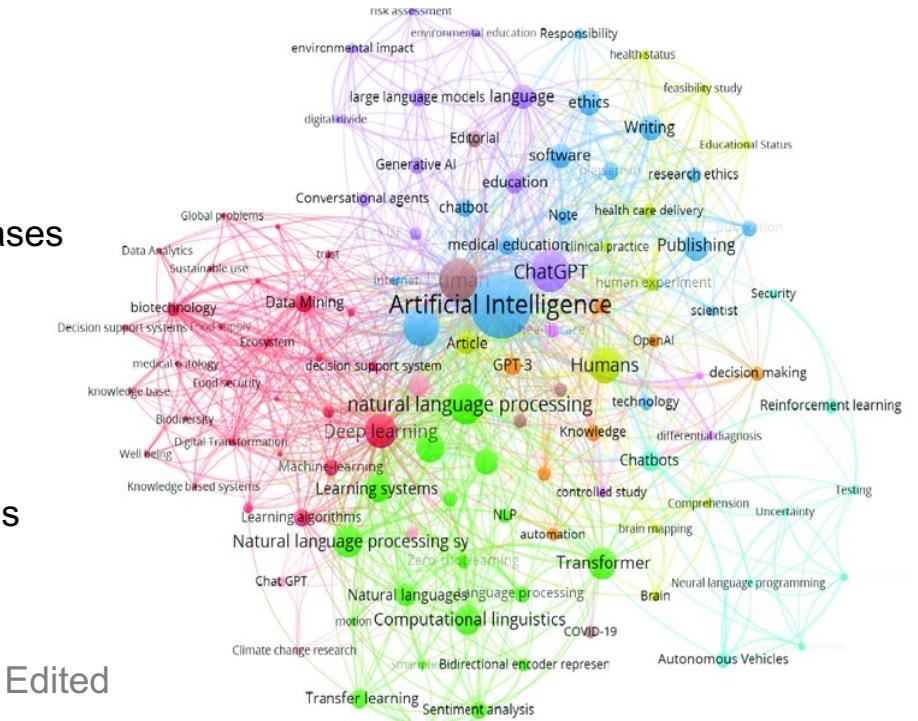


Figure: The thematic landscape of LLMs- Author: Reinaldo Luis Barbosa-Ramos Raidel - License: [CC BY- 4.0](#)

Key Applications of NLP

- Machine Translation: Statistical-based or Neural-based
- Sentiment Analysis: e.g., customer feedback analysis
- Chatbots and AI assistants: e.g., ChatGPT
- Text Summarization and Classification: e.g., identification of a document main topic
- Paraphrasing and Grammar correction: e.g., Grammarly, ChatGPT
- Auto-completion: e.g., the auto-complete in Google Search or mobile devices
- Named Entity Recognition (NER): e.g., identification of locations, organization or personal names
- Part-of-Speech tagging: e.g., recognizing the structure of phrases such as nouns, pronouns, verbs, etc..
- Document Clustering: e.g., grouping documents based on similarity

Fundamentals of NLP – Text Preprocessing

- Text preprocessing is a crucial step in NLP that involves transforming raw text into a clean and structured format suitable for analysis and modelling. The main goals are:
 - **Noise Reduction:** *Removing irrelevant or redundant information*
 - **Normalization:** *Standardizing text data to a consistent format, which helps in reducing the complexity*
 - **Tokenization:** *Splitting the text into smaller units (tokens) such as words or subwords, which serve as the basic units for analysis*

Fundamentals of NLP – Text Preprocessing

- An Example:
 - **Sentence 1:** "*Text preprocessing is essential for NLP!*". Tokens: ["Text", "preprocessing", "is", "essential", "for", "NLP"]
 - **Sentence 2:** "*Text preprocessing helps improve NLP models.*" Tokens: ["Text", "preprocessing", "helps", "improve", "NLP", "models"]
 - **Combined Tokens from both sentences:** ["Text", "preprocessing", "is", "essential", "for", "NLP", "Text", "preprocessing", "helps", "improve", "NLP", "models"]

Fundamentals of NLP – Text Preprocessing

- An Example (continued):
 - Tokens after lowercasing (Normalization): ["text", "preprocessing", "is", "essential", "for", "nlp", "text", "preprocessing", "helps", "improve", "nlp", "models"]
(Further normalization also include stemming and lemmatization)
 - Removing Stop Words (Noise Reduction): Not applied for some tasks such as auto-completion
 - Stop Words list: ["is", "for"]
 - Tokens after removing Stop Words: ["text", "preprocessing", "essential", "nlp", "text", "preprocessing", "helps", "improve", "nlp", "models"]

Fundamentals of NLP – Text Preprocessing

- An Example (continued):

Frequency Distribution Table

Token	Frequency
text	2
preprocessing	2
essential	1
nlp	2
helps	1
improve	1
models	1

Statistical approach to NLP

The N-grams Language Model

- A language model can be defined as a probability distribution describing the likelihood of any string
 - The model should say that “*Do I dare disturb the universe?*” has a reasonable probability as a string of English, but “*Universe dare the I disturb do?*” is extremely unlikely
 - With a language model, we can predict what words are likely to come next in a text, and thereby suggest completions for an email or text message based on the calculated statistics of how often words come after each other from some data. E.g., Wikipedia, or other textual datasets
 - **N-grams:** the model calculates the statistical representation of contiguous sequences of **N** items (Tokens) from a given text or speech. The items can be words, characters, or other units of text

The N-grams Language Model

- In the sentence "*Text preprocessing is essential for NLP!*"
 - 1-grams (Unigrams) representations: ["text", "preprocessing", "is", "essential", "for", "nlp"]
 - 2-grams (Bigrams) representations: ["text preprocessing", "preprocessing is", "is essential", "essential for", "for nlp"]
 - 3-grams (Trigrams) representations: ["text preprocessing is", "preprocessing is essential", "is essential for", "essential for nlp"]

The Bigrams Language Model

Combined Bigrams Frequency Count:

Bigram	Frequency
Text preprocessing	2
preprocessing is	1
is essential	1
essential for	1
for NLP	1
preprocessing helps	1
helps improve	1
improve NLP	1
NLP models	1

The Bigrams Language Model

- For an auto-completion task of the input: “*text pre*”

Probability of Next Token w_{i+1} Given w_i :

$$P(w_{i+1}|w_i) = \frac{\text{Count}(w_i, w_{i+1})}{\text{Count}(w_i)}$$

- Calculate Probabilities:

- For "preprocessing" after "text":

$$P(\text{"preprocessing"}|\text{"text"}) = \frac{\text{Count}(\text{"text"}, \text{"preprocessing"})}{\text{Count}(\text{"text"})} = \frac{2}{2} = 1$$

- For "helps" after "preprocessing":

$$P(\text{"helps"}|\text{"preprocessing"}) = \frac{\text{Count}(\text{"preprocessing"}, \text{"helps"})}{\text{Count}(\text{"preprocessing"})} = \frac{1}{2} = 0.5$$

The Bigrams Language Model

- Given the previous slide's calculations, the auto-complete suggests:
 - For Input "*text pre*":
 - Suggestion 1:** "text preprocessing" (high probability, based on bigram count)
 - Suggestion 2:** "preprocessing helps" (lower probability but valid)

The N-grams Language Model

- To generalize the calculations on any number (**N**) of contiguous sequences:

$$P(w_{1:N}) = \prod_{j=1}^N P(w_j | w_{1:j-1}).$$

Fundamentals of NLP – Part-of-Speech (POS)

- A statistical language model such as N-grams suffers when it needs to generalize beyond the data it calculates frequencies from
 - The model can auto-complete a sentence only based on what was available in its source dataset
 - To help overcoming the generalization limitation, more knowledge should be modelled as well
 - Hence, a structured representation of the model source dataset is recommended. POS is an example for such structured information

Fundamentals of NLP – Part-of-Speech (POS)

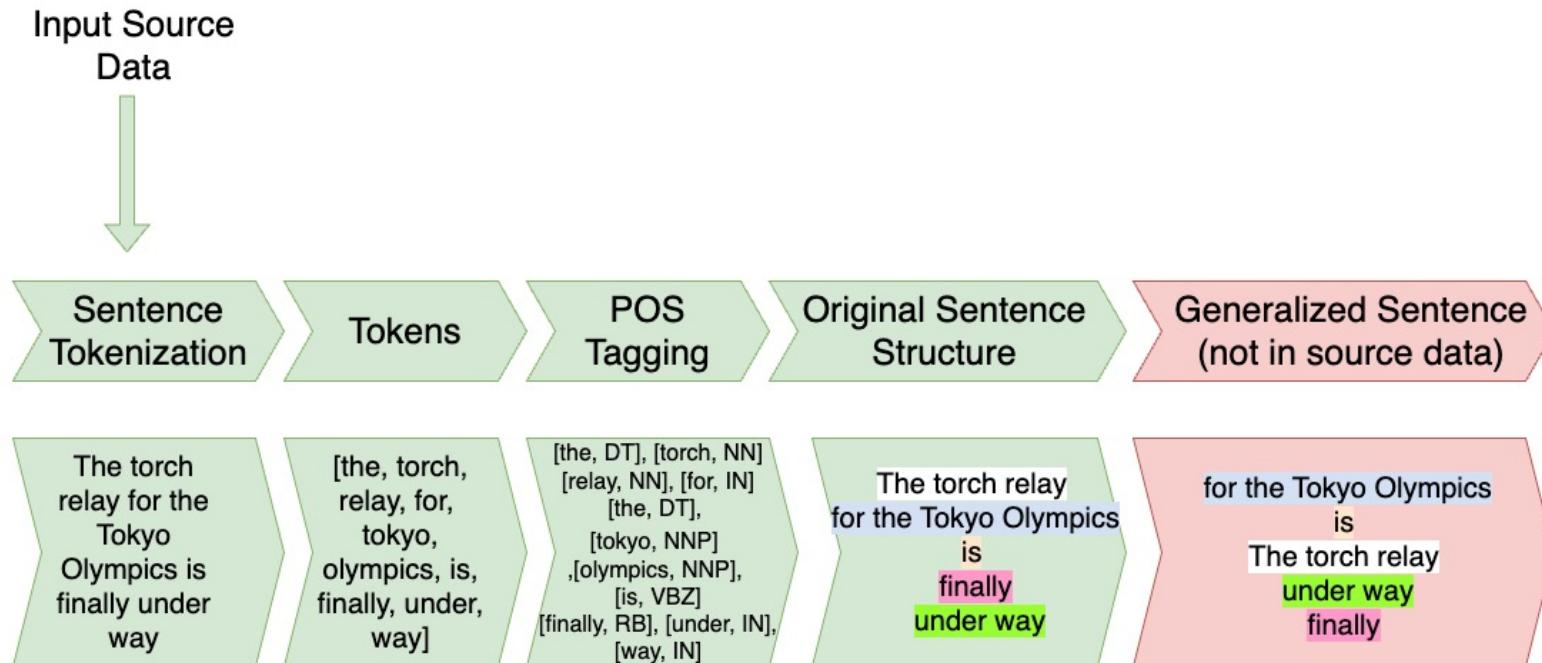


Figure: Example of Part-of-Speech (POS) process - Author: Brendon Albertson - License: [CC BY- 4.0](#)

Edited

Fundamentals of NLP – The Neural Approach

- Neural Network-based approach improves a language model and helps reducing manual feature engineering in the source datasets and allow for more generalization through some techniques:
 - **Word Embeddings:** a representation of words that does not require manual feature engineering, but allows for generalization between related words:
 - “colorless” and “ideal” are both adjectives, a syntax that can be learned automatically
 - “cat” and “kitten” are both felines, a semantic that can be learned automatically
 - “awesome” has opposite sentiment to “cringeworthy”, a sentiment that can be learned automatically

Fundamentals of NLP – The Neural Approach

- Traditionally, a word can be represented by a vector that hopefully capture its contextual meaning and how to capture the latter from the words that often come within its context:
 - Following the linguist John R. Firth's (1957) maxim, "You shall know a word by the company it keeps"
 - We could represent each word with a vector of n-gram counts of all the phrases that the word appears in. However, with a 100,000-word vocabulary, there are 10^{25} 5-grams to keep track of (although vectors in this 10^{25} -dimensional space would be quite sparse—most of the counts would be zero)
 - We would get better generalization if we reduced this to a smaller-size vector, perhaps with just a few hundred dimensions. We call this smaller, dense vector: a **word embedding**

Fundamentals of NLP – The Neural Approach

- Word Embeddings are learned automatically
 - They capture implicit relationships also, such as **aunt** to **niece** is same as **uncle** to **nephew**
 - They solve the computational overload in statistical NLP approaches

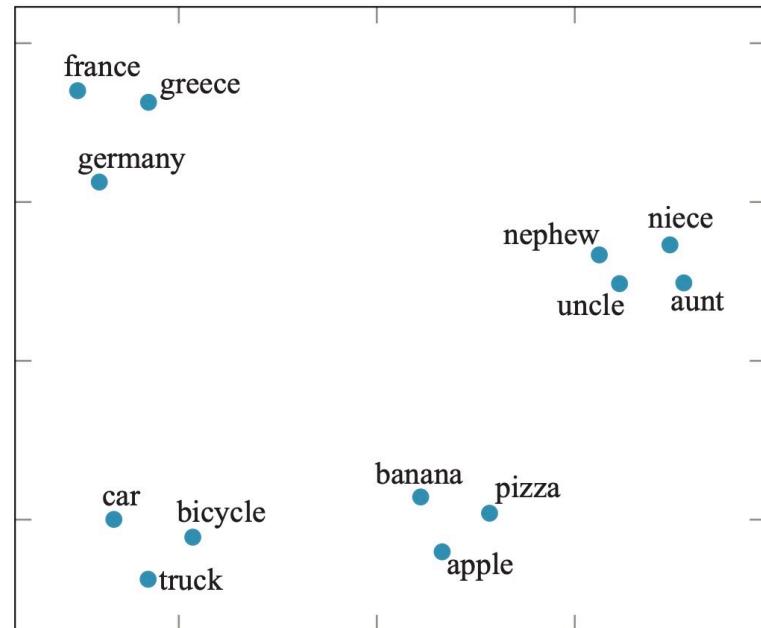


Figure: Word embedding vectors computed by the GloVe algorithm trained on 6 billion words of text. 100-dimensional word vectors are projected down onto two dimensions in this visualization. Similar words appear near each other

Mathematical Foundation for Word Embeddings

- Using Word2Vec Method: Predict context words from a given target word

1. Setup:

- **Vocabulary Size:** Assume a vocabulary of size V (e.g., 10,000 words).
- **Embedding Dimension:** Assume d (e.g., 300 dimensions).

2. Training Data:

- Example Sentence: "The quick brown fox jumps over the lazy dog."
- For target word "fox" and context words "quick," "brown," "jumps," "over" within a window of size 2.

3. One-Hot Encoding:

- Suppose the vocabulary indices are as follows:

- "fox" → Index 3
- "quick" → Index 7
- "brown" → Index 12
- "jumps" → Index 15
- "over" → Index 18



• One-Hot Vectors:

- "fox": $\mathbf{x}_{\text{fox}} = [0, 0, 1, 0, \dots, 0]$ (1 at index 3)
- "quick": $\mathbf{x}_{\text{quick}} = [0, 0, 0, 0, \dots, 1, \dots, 0]$ (1 at index 7)

Mathematical Foundation for Word Embeddings

4. Embedding Matrix \mathbf{W} :

- Initialize $\mathbf{W} \in \mathbb{R}^{V \times d}$, where each row \mathbf{w}_i is the embedding vector for word i .
 - Suppose \mathbf{W} is:

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_V \end{bmatrix}$$

- Where \mathbf{w}_{fox} is a row vector in \mathbb{R}^d (e.g., 300 dimensions).

Mathematical Foundation for Word Embeddings

5. Objective Function:

- Maximize the probability of context words given the target word. For a target word w_t and context words w_c :

$$\text{Loss} = - \sum_{c \in \text{context}} \log P(w_c | w_t)$$

- Using the softmax function for prediction:

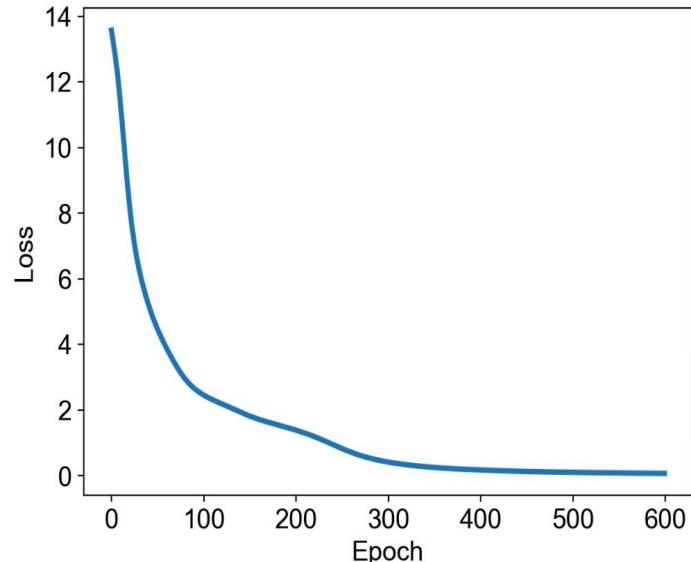
$$P(w_c | w_t) = \frac{\exp(\mathbf{v}_{w_c}^T \mathbf{v}_{w_t})}{\sum_{w \in V} \exp(\mathbf{v}_w^T \mathbf{v}_{w_t})}$$

- Here, \mathbf{v}_{w_t} is the embedding vector of the target word w_t , and \mathbf{v}_{w_c} is the embedding vector of the context word w_c .

6. Gradient Update:

- Update \mathbf{W} using gradient descent:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} \text{Loss}$$



Mathematical Foundation for Word Embeddings

- Using ELMo (Embeddings from Language Models) Method:
 - **Context Awareness:** Captures the meaning of words in the context of the entire sentence
 - **Dynamic Representations:** Produces different embeddings for the same word in different contexts
 - **Improved Performance:** Typically provides better performance on downstream tasks due to its ability to understand context

Mathematical Foundation for Word Embeddings

- Using ELMo (Embeddings from Language Models) Method:

1. Setup:

- **Vocabulary Size:** Assume a vocabulary of size V (e.g., 10,000 words).
- **Embedding Dimension:** Assume d (e.g., 300 dimensions).
- **Context:** The entire sentence is used for context, not a fixed window.

2. Training Data:

- **Example Sentence:** "The quick brown fox jumps over the lazy dog."
- **Target Word:** Consider the word "fox".
- **Context:** Context is the entire sentence; thus, both preceding and following words are used.

3. Initial Embeddings:

- **Word Embeddings:** Before training, each word is initialized with embeddings, often using pre-trained embeddings like Word2Vec or GloVe.

Mathematical Foundation for Word Embeddings

4. Embedding Matrix W :

- **Initialization:** Initialize $W \in \mathbb{R}^{V \times d}$, where each row w_i is the embedding vector for word i .

Suppose W is:

$$W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_V \end{bmatrix}$$

where w_{fox} is a row vector in \mathbb{R}^d (e.g., 300 dimensions).

Mathematical Foundation for Word Embeddings

5. Objective Function:

- **Language Modeling Objective:** Train the model to predict the next word in the sequence (language modeling). ELMo uses a bidirectional LSTM to generate embeddings from both directions.
- **Cross-Entropy Loss:** For each position t in the sentence, the loss is calculated based on predicting the next word in the sequence:

$$\text{Loss} = - \sum_{t=1}^T \log P(w_t | w_{<t}, w_{>t})$$

where $w_{<t}$ and $w_{>t}$ are the words before and after position t , respectively.

Mathematical Foundation for Word Embeddings

6. Bidirectional LSTM Processing:

- **Forward LSTM:**
 - Processes the sentence from left to right, generating embeddings that incorporate the preceding context.
- **Backward LSTM:**
 - Processes the sentence from right to left, generating embeddings that incorporate the following context.
- **Combining Representations:**
 - The final representation for each word is a combination of the forward and backward LSTM outputs:

$$\mathbf{h}_t = \alpha \cdot \mathbf{h}_{\text{forward},t} + (1 - \alpha) \cdot \mathbf{h}_{\text{backward},t}$$

where $\mathbf{h}_{\text{forward},t}$ and $\mathbf{h}_{\text{backward},t}$ are the embeddings from the forward and backward LSTMs, respectively, and α is a hyperparameter that balances the two representations.

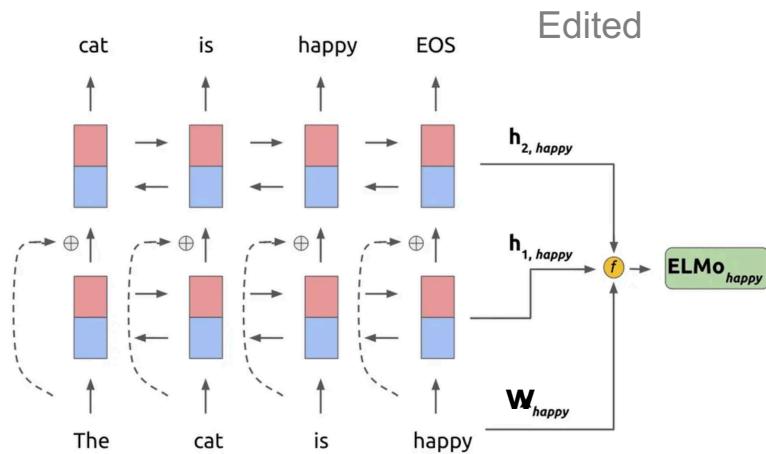


Figure: An example of combining the bidirectional hidden representations and word representation for "happy" to get an Elmo-specific representation – Author: [Karan Purohit](#)

Mathematical Foundation for Word Embeddings

7. Gradient Update:

- Backpropagation Through Time (BPTT):
 - Compute gradients for the LSTM parameters using BPTT. This involves:
 - Computing gradients for LSTM weights based on the loss function, taking into account the gradients from both directions (forward and backward).
- Update Parameters:
 - Update the parameters of the LSTM and embedding layers using gradient descent:

$$\mathbf{W}_{\text{new}} = \mathbf{W}_{\text{old}} - \eta \cdot \nabla_{\mathbf{W}} \text{Loss}$$

where η is the learning rate, and $\nabla_{\mathbf{W}} \text{Loss}$ are the gradients computed through BPTT.

ELMo vs Word2Vec

- **Deep Representation:**
 - **ELMo**: Utilizes deep bidirectional LSTMs, capturing richer and more complex representations of words by processing text in both forward and backward directions
 - **Word2Vec**: Uses shallow neural networks, which do not account for deep contextual information
- **Handling Polysemy:**
 - **ELMo**: Handles polysemy more effectively by generating different embeddings for the same word depending on its context, allowing for more accurate semantic representation
 - **Word2Vec**: Struggles with polysemy because it produces a single embedding per word

ELMo vs Recent Advances in NLP (Transformers)

- **Computational Efficiency:**
 - **ELMo**: Computationally expensive due to their sequential nature. Training and inference can be slower as each word is processed sequentially from both directions
 - **Transformers**: Allow for parallel processing of the entire sequence. This parallelization speeds up training and inference, making Transformers more efficient and scalable
- **Long-Range Dependencies:**
 - **ELMo**: While it captures context from both directions, they can still struggle with very long-range dependencies due to limitations in their capacity to retain information over long sequences
 - **Transformers**: Allow direct access to any part of the sequence, enabling better handling of long-range dependencies and relationships between distant words
- **Dynamic Embeddings:**
 - **ELMo**: Embeddings are generated from bidirectional LSTMs but still rely on static word embeddings as input. This can limit the dynamic nature of the representations.
 - **Transformers**: Generate embeddings dynamically based on the entire sequence context

Transformers

- **Self-Attention:**
 - Captures relationships between all words in a sequence
 - Handles long-range dependencies
- **Multi-Head Attention:**
 - Learns different relationships with multiple attention heads
 - Enhances representation richness
- **Positional Encoding:**
 - Adds position information to handle word order
 - Maintains sequence structure

Parallel Corpus

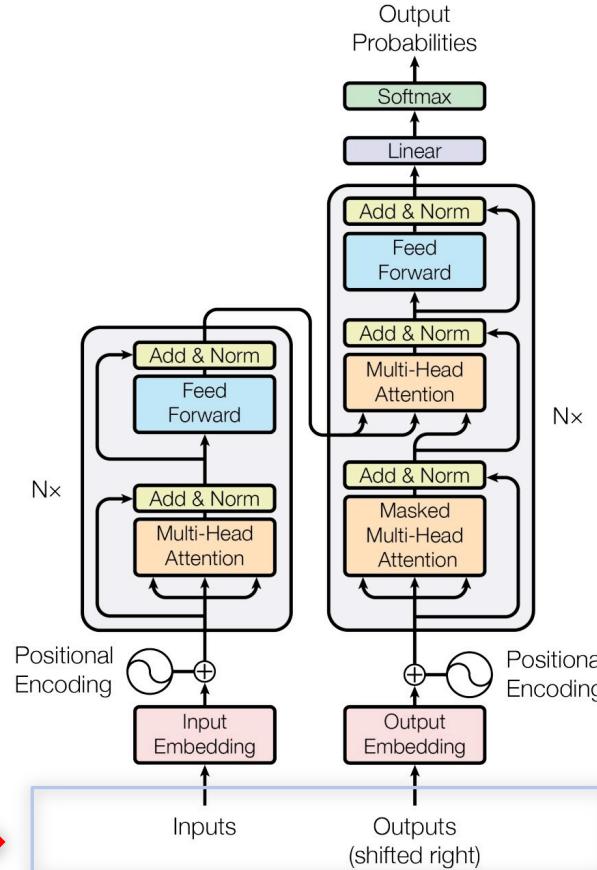


Figure: Transformer architecture – From the paper "[Attention Is All You Need](#)"

Quick Live Demo to Fine-Tune a Pretrained Transformer Model

Thank You!

Questions?

Lecture 11 — Planning

TDT4136: Introduction to Artificial Intelligence

Xavier F. C. Sánchez Díaz

Department of Computer Science
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology

October 31, 2024



Outline

1 What is Planning?

2 Representing the world

3 PDDL

4 How to plan?

Forward planning

Backward search

Partial ordering

5 Heuristic Planning

6 Planning in Complex Environments

What is Planning?

Classical Planning

Find a sequence of actions to accomplish a goal in an environment that is:

- ▶ discrete
- ▶ deterministic
- ▶ static
- ▶ fully observable

What is Planning?

Classical Planning

Find a sequence of actions to accomplish a goal in an environment that is:

- ▶ discrete
- ▶ deterministic
- ▶ static
- ▶ fully observable

In other words, what we have discussed before while studying **search!**

Why do we plan?

What is Planning?

- ▶ To address a new situation or problem

Why do we plan?

What is Planning?

- ▶ To address a new situation or problem
- ▶ When the task is complex

Why do we plan?

What is Planning?

- ▶ To address a new situation or problem
- ▶ When the task is complex
- ▶ when the environment imposes a **high risk or cost!**

Why do we plan?

What is Planning?

- ▶ To address a new situation or problem
- ▶ When the task is complex
- ▶ when the environment imposes a **high risk or cost!**
- ▶ when collaborating with others

Why do we plan?

What is Planning?

- ▶ To address a new situation or problem
- ▶ When the task is complex
- ▶ when the environment imposes a **high risk or cost!**
- ▶ when collaborating with others

Why do we plan?

What is Planning?

- ▶ To address a new situation or problem
- ▶ When the task is complex
- ▶ when the environment imposes a **high risk or cost!**
- ▶ when collaborating with others

We *explicitly* plan when it is strictly necessary.

How do we do it?

What is Planning?

There are multiple ways to define a planning problem, and several ways of finding a valid plan.

How do we do it?

What is Planning?

There are multiple ways to define a planning problem, and several ways of finding a valid plan.

So far, we have covered:

- ▶ Problem-solving by **searching**

How do we do it?

What is Planning?

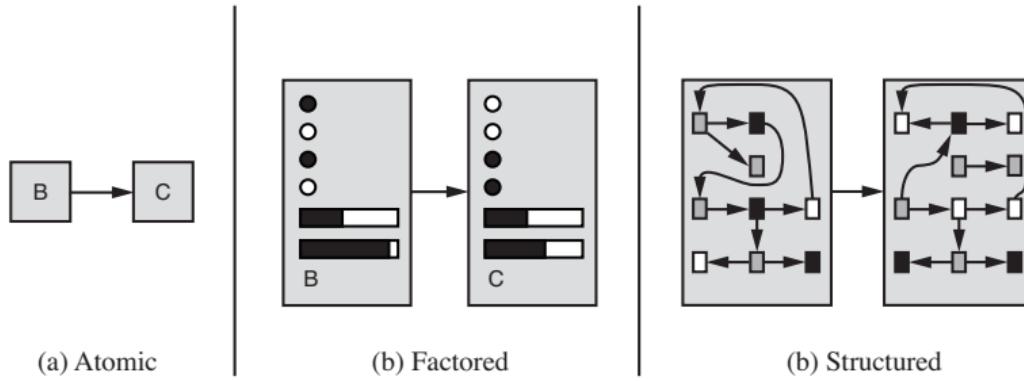
There are multiple ways to define a planning problem, and several ways of finding a valid plan.

So far, we have covered:

- ▶ Problem-solving by **searching**
- ▶ **Logic** and **satisfiability**

Representing the world

Representing the world



The **state of the world** can be described in different ways.

An example: Wumpus World

Representing the world

- ▶ A **partially observable** world, with **sensors** and a limited set of actions
- ▶ We act rationally by updating **our belief** of the world
- ▶ The world is **stored as facts in a knowledge base** (a logical agent can solve this!)

<https://thiagodnf.github.io/wumpus-world-simulator/>

What happens when we move?

What happens when we move?

Time is important!

Dealing with time(steps)

Representing the world

- ▶ Some aspects of the world change from time to time. We call them **fluents**.

Dealing with time(steps)

Representing the world

- ▶ Some aspects of the world change from time to time. We call them **fluents**.
- ▶ For example: the state of a grid cell—‘free’ or ‘occupied’ at time 0

Dealing with time(steps)

Representing the world

- ▶ Some aspects of the world change from time to time. We call them **fluents**.
- ▶ For example: the state of a grid cell—‘free’ or ‘occupied’ at time 0
- ▶ (First Order) Logic cannot deal with it

Dealing with time(steps)

Representing the world

- ▶ Some aspects of the world change from time to time. We call them **fluents**.
- ▶ For example: the state of a grid cell—‘free’ or ‘occupied’ at time 0
- ▶ (First Order) Logic cannot deal with it
 - ▶ There are extensions that can handle it, like **linear temporal** or **computational tree logics!**

Dealing with time(steps)

Representing the world

- ▶ Some aspects of the world change from time to time. We call them **fluents**.
- ▶ For example: the state of a grid cell—‘free’ or ‘occupied’ at time 0
- ▶ (First Order) Logic cannot deal with it
 - ▶ There are extensions that can handle it, like **linear temporal** or **computational tree logics!**
- ▶ An agent’s actions can change aspects of the world (fluents) but not all

Dealing with time(steps)

Representing the world

- ▶ Some aspects of the world change from time to time. We call them **fluents**.
- ▶ For example: the state of a grid cell—‘free’ or ‘occupied’ at time 0
- ▶ (First Order) Logic cannot deal with it
 - ▶ There are extensions that can handle it, like **linear temporal** or **computational tree logics!**
- ▶ An agent’s actions can change aspects of the world (fluents) but not all
- ▶ The agent needs to keep track of fluents, and know what remains unchanged!

A snapshot of the world

Representing the world

Time 0

I am at *cellA1* facing *east*, I feel a *breeze* and have *1 arrow*.

A snapshot of the world

Representing the world

Time 0

I am at *cellA1* facing *east*, I feel a *breeze* and have *1 arrow*.

If we decide to move *Forward*, then in logic:

$$\text{Location}_{\text{cellA1}}^0 \wedge \text{FacingEast}^0 \wedge \text{Forward}^0 \implies \text{Location}_{\text{cellA2}}^1 \wedge \neg \text{Location}_{\text{cellA1}}^1$$

... and although the *arrows* and *breeze* percepts were not modelled, we would have 4 directions $\times T$ time steps $\times n^2$ locations.

A snapshot of the world

Representing the world

Time 0

I am at *cellA1* facing *east*, I feel a *breeze* and have *1 arrow*.

If we decide to move *Forward*, then in logic:

$$\text{Location}_{\text{cellA1}}^0 \wedge \text{FacingEast}^0 \wedge \text{Forward}^0 \implies \text{Location}_{\text{cellA2}}^1 \wedge \neg \text{Location}_{\text{cellA1}}^1$$

... and although the *arrows* and *breeze* percepts were not modelled, we would have 4 directions $\times T$ time steps $\times n^2$ locations.

It is extremely expensive and inefficient!

Planning Domain Definition Language

PDDL

Instead, we can use **PDDL**, which uses **actions** with **preconditions** and **effects**:

Planning Domain Definition Language

PDDL

Instead, we can use **PDDL**, which uses **actions** with **preconditions** and **effects**:

Action(Move(who, from, to))

Precond : At(who, from) \wedge Adj(from, to) \wedge \neg Pit(to)

Effect : \neg At(who, from) \wedge At(who, to))

Planning Domain Definition Language

PDDL

Action(Move(*who*, *from*, *to*)

Precond : At(*who*, *from*) \wedge Adj(*from*, *to*) \wedge \neg Pit(*to*)

Effect : \neg At(*who*, *from*) \wedge At(*who*, *to*)

- ▶ *Move* is the action being defined
- ▶ *who*, *from* and *to* are variables
- ▶ *Precond* describes the state of the world needed for the action to occur
- ▶ *Effect* describes the resulting state after acting

States in PDDL

PDDL

- ▶ The world is closed—any **fluents** not mentioned are *False*

States in PDDL

PDDL

- ▶ The world is closed—any **fluents** not mentioned are *False*
- ▶ Unique names—different literals refer to different entities (like *cellA1* and *cellA2*)

States in PDDL

PDDL

- ▶ The world is closed—any **fluents** not mentioned are *False*
- ▶ Unique names—different literals refer to different entities (like *cellA1* and *cellA2*)
- ▶ No uncertain or negated literals

States in PDDL

PDDL

- ▶ The world is closed—any **fluents** not mentioned are *False*
- ▶ Unique names—different literals refer to different entities (like *cellA1* and *cellA2*)
- ▶ No uncertain or negated literals

States in PDDL

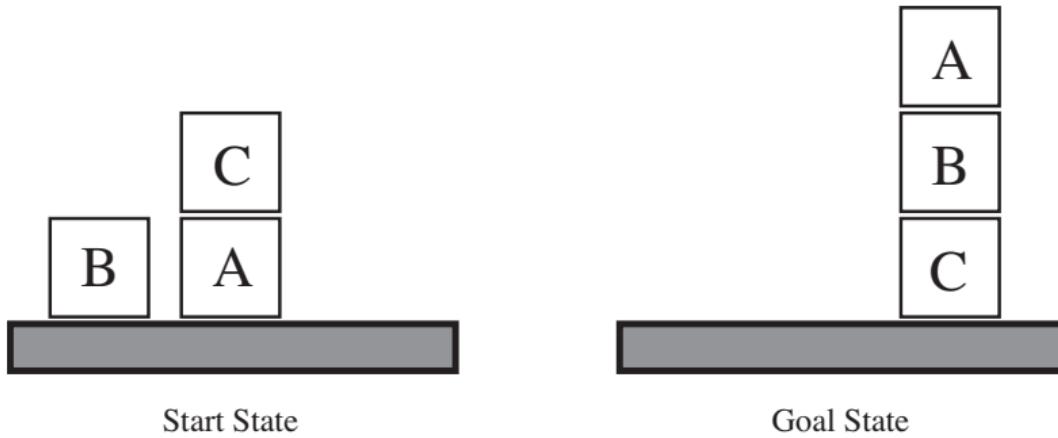
PDDL

- ▶ The world is closed—any **fluents** not mentioned are *False*
- ▶ Unique names—different literals refer to different entities (like *cellA1* and *cellA2*)
- ▶ No uncertain or negated literals

As with search, we also need **starting** and **goal** states.

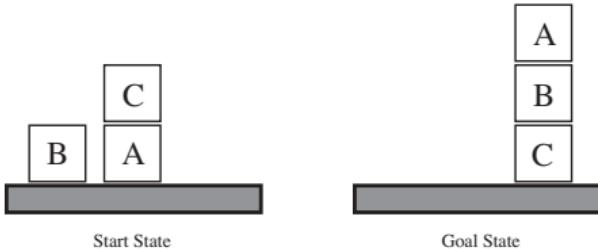
Example: Block world

PDDL



What would the **start and goal** states look like in **PDDL**?

Example: Block world



- ▶ **Start:** $On(A, Table)$, $On(B, Table)$, $On(C, A)$, $Clear(B)$, $Clear(C)$ ¹
- ▶ **Goal:** $On(A, B)$, $On(B, C)$

¹Commas are a shorthand for *ANDs* (\wedge) and semicolons for *ORs* (\vee)

Example: Block world

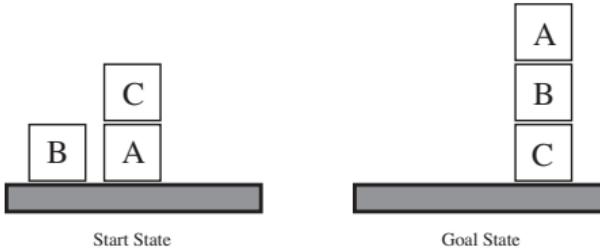


- ▶ **Start:** $On(A, Table)$, $On(B, Table)$, $On(C, A)$, $Clear(B)$, $Clear(C)$ ¹
- ▶ **Goal:** $On(A, B)$, $On(B, C)$

What would the **actions** look like in **PDDL**?

¹Commas are a shorthand for **ANDs** (\wedge) and semicolons for **ORs** (\vee)

Example: Block world



- ▶ **Start:** $On(A, Table)$, $On(B, Table)$, $On(C, A)$, $Clear(B)$, $Clear(C)$
- ▶ **Goal:** $On(A, B)$, $On(B, C)$

Action($Move(block, x, y)$)

PRECOND : $On(block, x)$, $Clear(block)$, $Clear(y)$, $Block(block)$, $Block(y)$

EFFECT : $On(block, y)$, $Clear(x)$, $\neg On(block, x)$, $\neg Clear(y)$

Notice how any variable in the **effect** must appear in the **precondition!**

Example: Block world

- ▶ **Start:** $On(A, Table)$, $On(B, Table)$, $On(C, A)$, $Clear(B)$, $Clear(C)$
- ▶ **Goal:** $On(A, B)$, $On(B, C)$

Action($Move(block, x, y)$)

PRECOND : $On(block, x)$, $Clear(block)$, $Clear(y)$

EFFECT : $On(block, y)$, $Clear(x)$, $\neg On(block, x)$, $\neg Clear(y)$

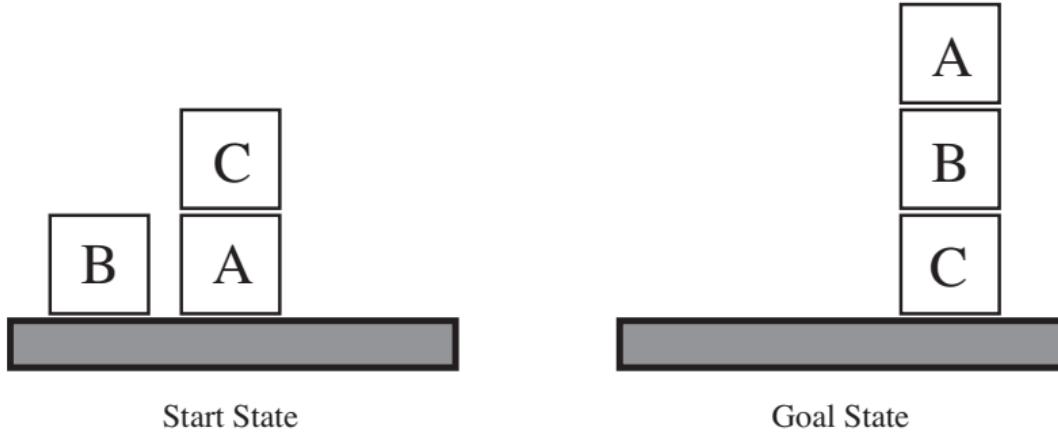


Action($MoveToTable(block, x)$)

PRECOND : $On(block, x)$, $Clear(block)$, $Clear(Table)$

EFFECT : $On(block, Table)$, $Clear(x)$, $\neg On(block, x)$

Example: Block world



Solution: $[MoveToTable(C, A), Move(B, Table, C), Move(A, Table, B)]$

Adding and deleting

PDDL

Action(Move(block, x, y))

PRECOND : On(block, x), Clear(block), Clear(y)

EFFECT : On(block, y), Clear(x), \neg On(block, x), \neg Clear(y))

At **each step** of the plan, the **state of the world** is **modified** depending on the **effect** of the action taken:

Adding and deleting

PDDL

Action(Move(block, x, y))

PRECOND : On(block, x), Clear(block), Clear(y)

EFFECT : On(block, y), Clear(x), \neg On(block, x), \neg Clear(y))

At **each step** of the plan, the **state of the world** is **modified** depending on the **effect** of the action taken:

- ▶ Positive fluents are **added**

Adding and deleting

PDDL

Action(Move(block, x, y))

PRECOND : On(block, x), Clear(block), Clear(y)

EFFECT : On(block, y), Clear(x), \neg On(block, x), \neg Clear(y))

At **each step** of the plan, the **state of the world** is **modified** depending on the **effect** of the action taken:

- ▶ Positive fluents are **added**
- ▶ Negative fluents are **deleted**

Adding and deleting

PDDL

Action(Move(block, x, y))

PRECOND : On(block, x), Clear(block), Clear(y)

EFFECT : On(block, y), Clear(x), \neg On(block, x), \neg Clear(y))

At **each step** of the plan, the **state of the world** is **modified** depending on the **effect** of the action taken:

- ▶ Positive fluents are **added**
- ▶ Negative fluents are **deleted**

Adding and deleting

PDDL

Action(Move(block, x, y))

PRECOND : On(block, x), Clear(block), Clear(y)

EFFECT : On(block, y), Clear(x), \neg On(block, x), \neg Clear(y))

At **each step** of the plan, the **state of the world** is **modified** depending on the **effect** of the action taken:

- ▶ Positive fluents are **added**
- ▶ Negative fluents are **deleted**

These are known as the ADD and DEL lists, and allow us to calculate the state s at the next time step after taking action a :

$$s^{(t+1)} = (s^{(t)} \setminus DEL(a)) \cup ADD(a)$$

Designing actions

PDDL

There are some other things to consider.

- ▶ How would the system differentiate between $\text{MoveToTable}(\text{block}, x)$ and $\text{Move}(\text{block}, x, \text{Table})$?

Designing actions

PDDL

There are some other things to consider.

- ▶ How would the system differentiate between $\text{MoveToTable}(\text{block}, x)$ and $\text{Move}(\text{block}, x, \text{Table})$?
 - ▶ Larger-than-necessary search space!

Designing actions

PDDL

There are some other things to consider.

- ▶ How would the system differentiate between $\text{MoveToTable}(\text{block}, x)$ and $\text{Move}(\text{block}, x, \text{Table})$?
 - ▶ Larger-than-necessary search space!
- ▶ Some operators are a bit dumb, i.e., $\text{Move}(B, C, C)$ as it introduces an inconsistent $\text{Clear}(C), \neg\text{Clear}(C)$

Designing actions

PDDL

There are some other things to consider.

- ▶ How would the system differentiate between $\text{MoveToTable}(\text{block}, x)$ and $\text{Move}(\text{block}, x, \text{Table})$?
 - ▶ Larger-than-necessary search space!
- ▶ Some operators are a bit dumb, i.e., $\text{Move}(B, C, C)$ as it introduces an inconsistent $\text{Clear}(C), \neg\text{Clear}(C)$

Designing actions

PDDL

There are some other things to consider.

- ▶ How would the system differentiate between $\text{MoveToTable}(\text{block}, x)$ and $\text{Move}(\text{block}, x, \text{Table})$?
 - ▶ Larger-than-necessary search space!
- ▶ Some operators are a bit dumb, i.e., $\text{Move}(B, C, C)$ as it introduces an inconsistent $\text{Clear}(C), \neg\text{Clear}(C)$
- ▶ A solution is to use the predicate Block so that in order to $\text{Move}(x, y, z)$ we need $\text{Block}(x), \text{Block}(y)$

Designing actions

PDDL

There are some other things to consider.

- ▶ How would the system differentiate between $\text{MoveToTable}(\text{block}, x)$ and $\text{Move}(\text{block}, x, \text{Table})$?
 - ▶ Larger-than-necessary search space!
- ▶ Some operators are a bit dumb, i.e., $\text{Move}(B, C, C)$ as it introduces an inconsistent $\text{Clear}(C), \neg\text{Clear}(C)$
- ▶ A solution is to use the predicate Block so that in order to $\text{Move}(x, y, z)$ we need $\text{Block}(x), \text{Block}(y)$
- ▶ Another approach is to use $\neg(y = z)$ as a precondition for $\text{Move}(x, y, z)$

Designing actions

PDDL

There are some other things to consider.

- ▶ How would the system differentiate between $\text{MoveToTable}(\text{block}, x)$ and $\text{Move}(\text{block}, x, \text{Table})$?
 - ▶ Larger-than-necessary search space!
- ▶ Some operators are a bit dumb, i.e., $\text{Move}(B, C, C)$ as it introduces an inconsistent $\text{Clear}(C), \neg\text{Clear}(C)$
- ▶ A solution is to use the predicate Block so that in order to $\text{Move}(x, y, z)$ we need $\text{Block}(x), \text{Block}(y)$
- ▶ Another approach is to use $\neg(y = z)$ as a precondition for $\text{Move}(x, y, z)$

Designing actions

PDDL

There are some other things to consider.

- ▶ How would the system differentiate between $\text{MoveToTable}(\text{block}, x)$ and $\text{Move}(\text{block}, x, \text{Table})$?
 - ▶ Larger-than-necessary search space!
- ▶ Some operators are a bit dumb, i.e., $\text{Move}(B, C, C)$ as it introduces an inconsistent $\text{Clear}(C), \neg\text{Clear}(C)$
- ▶ A solution is to use the predicate Block so that in order to $\text{Move}(x, y, z)$ we need $\text{Block}(x), \text{Block}(y)$
- ▶ Another approach is to use $\neg(y = z)$ as a precondition for $\text{Move}(x, y, z)$

It is **not easy!**

How to plan?

There are several ways to come up with a feasible plan, and the **search space** might be different on each approach.

How to plan?

There are several ways to come up with a feasible plan, and the **search space** might be different on each approach.

- ▶ **State-space planning:** search through nodes representing states of the world. A plan is a **path** through the space
- ▶ **Plan-space planning:** search through partially instantiated operators and constraints—starts with a partial, *possibly incorrect* plan and then apply changes to correct it.
- ▶ **Heuristic planning:** search for a sequence of actions and evaluate your plan using an objective function.

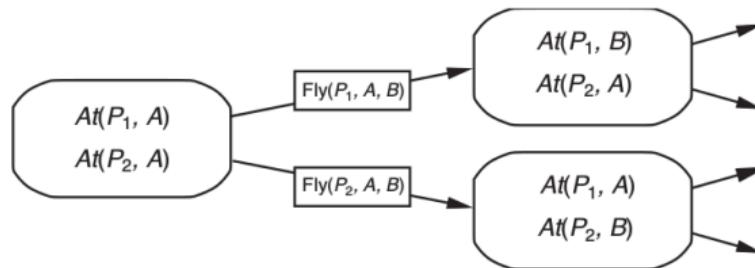
Algorithms for classical planning

How to plan?

- ▶ **Forward** (progression) search
- ▶ **Backward** (regression) search
- ▶ **Logical Inference**

Forward planning

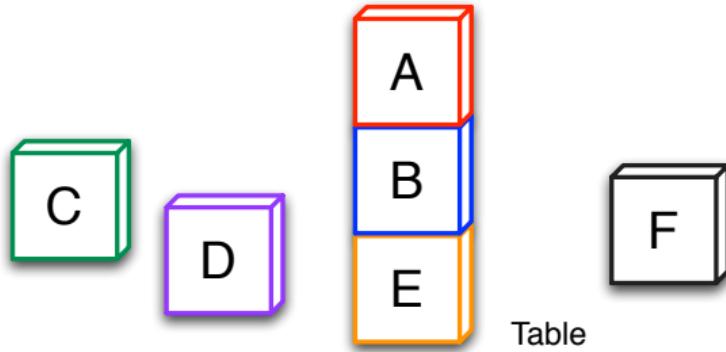
How to plan?



1. Determine all **actions** applicable
2. **Ground**² the actions by replacing any **variable** with **constants**
3. Choose an **action** to apply
4. Determine the new state of the world and update the knowledge base according to the action description
5. Repeat this process until the **goal state** is reached

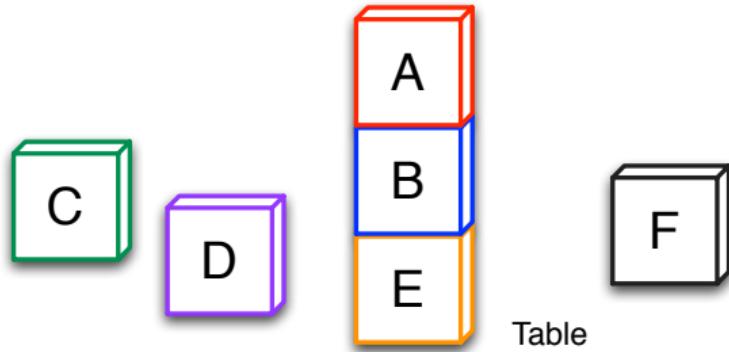
²Instantiate, if you will

Forward planning



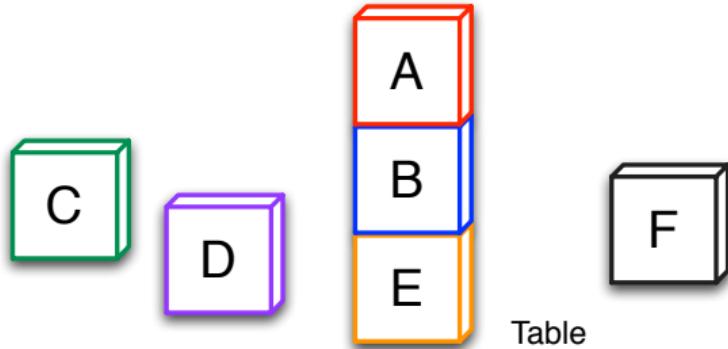
What is your plan?

Forward planning



How many possible **first actions** are there?

Forward planning



How many possible **first actions** are there?

How do we know which one is the *best* one?

Branching Factor

Forward Search

- ▶ **Forward search** can have a **very large** branching factor

Branching Factor

Forward Search

- ▶ **Forward search** can have a **very large** branching factor
 - ▶ Many applicable ^{dumb} actions that do not progress towards our goal

Branching Factor

Forward Search

- ▶ **Forward search** can have a **very large** branching factor
 - ▶ Many applicable ^{dumb} actions that do not progress towards our goal
- ▶ The search algorithms we have covered can waste a lot of time here

Branching Factor

Forward Search

- ▶ **Forward search** can have a **very large** branching factor
 - ▶ Many applicable ^{dumb} actions that do not progress towards our goal
- ▶ The search algorithms we have covered can waste a lot of time here

Branching Factor

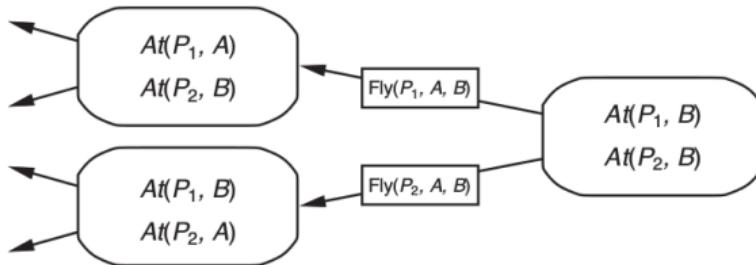
Forward Search

- ▶ **Forward search** can have a **very large** branching factor
 - ▶ Many applicable ^{dumb} actions that do not progress towards our goal
- ▶ The search algorithms we have covered can waste a lot of time here

It needs a good (domain-specific) heuristic or pruning procedure!

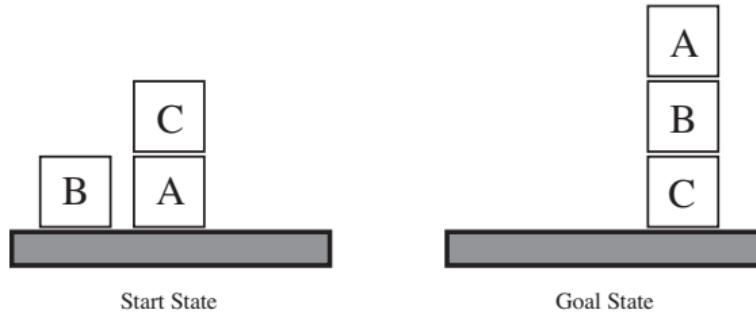
Backward search

How to plan?



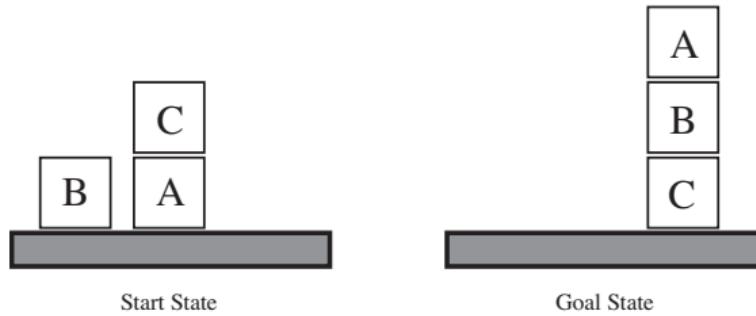
1. Choose a **relevant action** that satisfies (*some*) goal propositions
2. Make a new **goal** by applying an **action *a* backwards**:
 - ▶ *DEL* satisfied conditions of **goal**
 - ▶ *ADD* preconditions of **a**
 - ▶ Keep unsolved **goal** propositions
3. Repeat until the **goal** is satisfied by the start state

Backward search



What are the **relevant actions** here?

Backward search



What are the **relevant actions** here?

How do we know which one is the *best* one?

Branching factor

Backward search

- ▶ Backward search is not guided towards any **specific subgoal**

Branching factor

Backward search

- ▶ **Backward search** is not guided towards any **specific subgoal**
- ▶ The **order** in which we try to achieve the subgoals (and do search) matters

Branching factor

Backward search

- ▶ **Backward search** is not guided towards any **specific subgoal**
- ▶ The **order** in which we try to achieve the subgoals (and do search) matters
 - ▶ It impacts the efficiency of the search

Branching factor

Backward search

- ▶ **Backward search is not guided towards any specific subgoal**
- ▶ The **order** in which we try to achieve the subgoals (and do search) matters
 - ▶ It impacts the efficiency of the search
 - ▶ A wrong order can make the plan unfeasible

Sub-plans: Total and partial orders

How to plan

So far, we have only looked at algorithms that generate **complete** plans, with a **strict** order. However, there may exist **subplans**—sequence of actions that can be **partially** ordered.

Sub-plans: Total and partial orders

How to plan

So far, we have only looked at algorithms that generate **complete** plans, with a **strict** order. However, there may exist **subplans**—sequence of actions that can be **partially** ordered.

Think about it as when you put your socks and shoes on every morning:

Sub-plans: Total and partial orders

How to plan

So far, we have only looked at algorithms that generate **complete** plans, with a **strict** order. However, there may exist **subplans**—sequence of actions that can be **partially** ordered.

Think about it as when you put your socks and shoes on every morning:

- ▶ Which *sock* should go on first?

Sub-plans: Total and partial orders

How to plan

So far, we have only looked at algorithms that generate **complete** plans, with a **strict** order. However, there may exist **subplans**—sequence of actions that can be **partially** ordered.

Think about it as when you put your socks and shoes on every morning:

- ▶ Which *sock* should go on first?
- ▶ Do you do the subsequence $\text{sock}_1 \rightarrow \text{shoe}_1$ like a lunatic or $\text{sock}_1 \rightarrow \text{sock}_2$ first?

If we had enough hands I guess we could do $\text{sock}_i \rightarrow \text{shoe}_i$ in **parallel**!

Heuristic Planning

As usual, the **real world** is a much scarier place.

Heuristic Planning

As usual, the **real world** is a much scarier place.

- ▶ Problems (instances) can be huge (number of variables, number of constraints)

Heuristic Planning

As usual, the **real world** is a much scarier place.

- ▶ Problems (instances) can be huge (number of variables, number of constraints)
- ▶ We have limited resources / memory / time / money

Heuristic Planning

As usual, the **real world** is a much scarier place.

- ▶ Problems (instances) can be huge (number of variables, number of constraints)
- ▶ We have limited resources / memory / time / money
- ▶ We need faster solutions

Heuristic Planning

As usual, the **real world** is a much scarier place.

- ▶ Problems (instances) can be huge (number of variables, number of constraints)
- ▶ We have limited resources / memory / time / money
- ▶ We need faster solutions

Heuristic Planning

As usual, the **real world** is a much scarier place.

- ▶ Problems (instances) can be huge (number of variables, number of constraints)
- ▶ We have limited resources / memory / time / money
- ▶ We need faster solutions

We need **heuristic** solutions!

General idea: relax

Heuristic Planning

Most heuristics rely on **relaxing** the problem to make it easier.

General idea: relax

Heuristic Planning

Most heuristics rely on **relaxing** the problem to make it easier.

In planning:

- ▶ Add edges and **group** nodes ([subplans](#))

General idea: relax

Heuristic Planning

Most heuristics rely on **relaxing** the problem to make it easier.

In planning:

- ▶ Add edges and **group** nodes (subplans)
 - ▶ State abstraction, pattern DBs, symmetry reduction...

General idea: relax

Heuristic Planning

Most heuristics rely on **relaxing** the problem to make it easier.

In planning:

- ▶ Add edges and **group** nodes (subplans)
 - ▶ State abstraction, pattern DBs, symmetry reduction...
- ▶ Ignore restrictions

General idea: relax

Heuristic Planning

Most heuristics rely on **relaxing** the problem to make it easier.

In planning:

- ▶ Add edges and **group** nodes (subplans)
 - ▶ State abstraction, pattern DBs, symmetry reduction...
- ▶ Ignore **restrictions**
 - ▶ Either **all** or **some** of them

General idea: relax

Heuristic Planning

Most heuristics rely on **relaxing** the problem to make it easier.

In planning:

- ▶ Add edges and **group** nodes (subplans)
 - ▶ State abstraction, pattern DBs, symmetry reduction...
- ▶ Ignore **restrictions**
 - ▶ Either **all** or **some** of them
- ▶ Ignore **negative fluents**

General idea: relax

Heuristic Planning

Most heuristics rely on **relaxing** the problem to make it easier.

In planning:

- ▶ Add edges and **group** nodes (subplans)
 - ▶ State abstraction, pattern DBs, symmetry reduction...
- ▶ Ignore **restrictions**
 - ▶ Either **all** or **some** of them
- ▶ Ignore **negative fluents**
- ▶ Weigh actions to have **preferred** actions

General idea: relax

Heuristic Planning

Most heuristics rely on **relaxing** the problem to make it easier.

In planning:

- ▶ Add edges and **group** nodes (**subplans**)
 - ▶ State abstraction, pattern DBs, symmetry reduction...
- ▶ Ignore **restrictions**
 - ▶ Either **all** or **some** of them
- ▶ Ignore **negative fluents**
- ▶ Weigh actions to have **preferred** actions
- ▶ Find **serialisable subplans**

General idea: relax

Heuristic Planning

Most heuristics rely on **relaxing** the problem to make it easier.

In planning:

- ▶ Add edges and **group** nodes (**subplans**)
 - ▶ State abstraction, pattern DBs, symmetry reduction...
- ▶ Ignore **restrictions**
 - ▶ Either **all** or **some** of them
- ▶ Ignore **negative fluents**
- ▶ Weigh actions to have **preferred** actions
- ▶ Find **serialisable subplans**
 - ▶ Achieving one will not undo the others.

General idea: relax

Heuristic Planning

Most heuristics rely on **relaxing** the problem to make it easier.

In planning:

- ▶ Add edges and **group** nodes (**subplans**)
 - ▶ State abstraction, pattern DBs, symmetry reduction...
- ▶ Ignore **restrictions**
 - ▶ Either **all** or **some** of them
- ▶ Ignore **negative fluents**
- ▶ Weigh actions to have **preferred** actions
- ▶ Find **serialisable subplans**
 - ▶ Achieving one will not undo the others.

General idea: relax

Heuristic Planning

Most heuristics rely on **relaxing** the problem to make it easier.

In planning:

- ▶ Add edges and **group** nodes (**subplans**)
 - ▶ State abstraction, pattern DBs, symmetry reduction...
- ▶ Ignore **restrictions**
 - ▶ Either **all** or **some** of them
- ▶ Ignore **negative fluents**
- ▶ Weigh actions to have **preferred** actions
- ▶ Find **serialisable subplans**
 - ▶ Achieving one will not undo the others.

... or try a **metaheuristic!**

Examples of relaxation

Heuristic Planning

- ▶ **State abstraction:** group subtasks into one bigger, more abstract task

Examples of relaxation

Heuristic Planning

- ▶ **State abstraction:** group subtasks into one bigger, more abstract task
 - ▶ All packages for Trondheim (from Oslo) count as one big package

Examples of relaxation

Heuristic Planning

- ▶ **State abstraction:** group subtasks into one bigger, more abstract task
 - ▶ All packages for Trondheim (from Oslo) count as one big package
 - ▶ Worry about getting those packages to Trondheim first. You can worry about individual deliveries afterwards!

Examples of relaxation

Heuristic Planning

- ▶ **State abstraction:** group subtasks into one bigger, more abstract task
 - ▶ All packages for Trondheim (from Oslo) count as one big package
 - ▶ Worry about getting those packages to Trondheim first. You can worry about individual deliveries afterwards!
 - ▶ Decompose costs: $\text{Cost}(P) = \text{Cost}(P_i) + \text{Cost}(P_j)$ where i, j are subgoals

Examples of relaxation

Heuristic Planning

- ▶ **State abstraction:** group subtasks into one bigger, more abstract task
 - ▶ All packages for Trondheim (from Oslo) count as one big package
 - ▶ Worry about getting those packages to Trondheim first. You can worry about individual deliveries afterwards!
 - ▶ Decompose costs: $\text{Cost}(P) = \text{Cost}(P_i) + \text{Cost}(P_j)$ where i, j are subgoals
- ▶ **Ignore restrictions:** ‘a perfect plan *if* this road r were not closed’

Examples of relaxation

Heuristic Planning

- ▶ **State abstraction:** group subtasks into one bigger, more abstract task
 - ▶ All packages for Trondheim (from Oslo) count as one big package
 - ▶ Worry about getting those packages to Trondheim first. You can worry about individual deliveries afterwards!
 - ▶ Decompose costs: $\text{Cost}(P) = \text{Cost}(P_i) + \text{Cost}(P_j)$ where i, j are subgoals
- ▶ **Ignore restrictions:** ‘a perfect plan *if* this road *r* were not closed’
 - ▶ We will try finding an alternative route for the *r* segment later!

Examples of relaxation

Heuristic Planning

- ▶ **State abstraction:** group subtasks into one bigger, more abstract task
 - ▶ All packages for Trondheim (from Oslo) count as one big package
 - ▶ Worry about getting those packages to Trondheim first. You can worry about individual deliveries afterwards!
 - ▶ Decompose costs: $\text{Cost}(P) = \text{Cost}(P_i) + \text{Cost}(P_j)$ where i, j are subgoals
- ▶ **Ignore restrictions:** ‘a perfect plan *if* this road r were not closed’
 - ▶ We will try finding an alternative route for the r segment later!
- ▶ **Serialisable subplans:** achieving a subgoal (putting on $shoe_1$) does not interfere with other goals (putting on $shoe_2$)

Classical Planning vs IRL

Planning in Complex Environments

So far, we have covered how to do **classical planning**:

Classical Planning vs IRL

Planning in Complex Environments

So far, we have covered how to do **classical planning**:

1. The environment is **fully observable**

Classical Planning vs IRL

Planning in Complex Environments

So far, we have covered how to do **classical planning**:

1. The environment is **fully observable**
 - We can “see” what happens at any state

Classical Planning vs IRL

Planning in Complex Environments

So far, we have covered how to do **classical planning**:

1. The environment is **fully observable**
 - ▶ We can “see” what happens at any state
2. The environment is **deterministic**

Classical Planning vs IRL

Planning in Complex Environments

So far, we have covered how to do **classical planning**:

1. The environment is **fully observable**
 - ▶ We can “see” what happens at any state
2. The environment is **deterministic**
 - ▶ Our actions always give us the same effect

Classical Planning vs IRL

Planning in Complex Environments

So far, we have covered how to do **classical planning**:

1. The environment is **fully observable**
 - ▶ We can “see” what happens at any state
2. The environment is **deterministic**
 - ▶ Our actions always give us the same effect
3. The environment is **static**

Classical Planning vs IRL

Planning in Complex Environments

So far, we have covered how to do **classical planning**:

1. The environment is **fully observable**
 - ▶ We can “see” what happens at any state
2. The environment is **deterministic**
 - ▶ Our actions always give us the same effect
3. The environment is **static**
 - ▶ It never changes, unless we act

Classical Planning vs IRL

Planning in Complex Environments

So far, we have covered how to do **classical planning**:

1. The environment is **fully observable**
 - ▶ We can “see” what happens at any state
2. The environment is **deterministic**
 - ▶ Our actions always give us the same effect
3. The environment is **static**
 - ▶ It never changes, unless we act

Classical Planning vs IRL

Planning in Complex Environments

So far, we have covered how to do **classical planning**:

1. The environment is **fully observable**
 - ▶ We can “see” what happens at any state
2. The environment is **deterministic**
 - ▶ Our actions always give us the same effect
3. The environment is **static**
 - ▶ It never changes, unless we act

How do we plan when we encounter more complex environments?

Planning in Complex Environments

1. I cannot see

Planning in Complex Environments

1. I cannot see
 - So we make a **sensorless plan**

Planning in Complex Environments

1. I cannot see
 - ▶ So we make a **sensorless plan**
2. I can see but I am not sure what will happen

Planning in Complex Environments

1. I cannot see
 - ▶ So we make a **sensorless plan**
2. I can see but I am not sure what will happen
 - ▶ So we make a **contingency plan**

Planning in Complex Environments

1. I cannot see
 - ▶ So we make a **sensorless plan**
2. I can see but I am not sure what will happen
 - ▶ So we make a **contingency plan**
3. I can see and am sure what will happen but need to keep an eye

Planning in Complex Environments

1. I cannot see
 - ▶ So we make a **sensorless plan**
2. I can see but I am not sure what will happen
 - ▶ So we make a **contingency plan**
3. I can see and am sure what will happen but need to keep an eye
 - ▶ Then we need **online planning**

I cannot see

Sensorless planning

- ▶ I need to **make sure** that all **preconditions** are **met**
- ▶ I will then **carry out all operations** that will lead me to the **goal**

Example: paint a chair and a table with the same colour. How?

I Cannot See



I cannot see

Sensorless planning

- ▶ I need to **make sure** that all **preconditions** are **met**
- ▶ I will then **carry out all operations** that will lead me to the **goal**

Example: paint a chair and a table with the same colour. How?

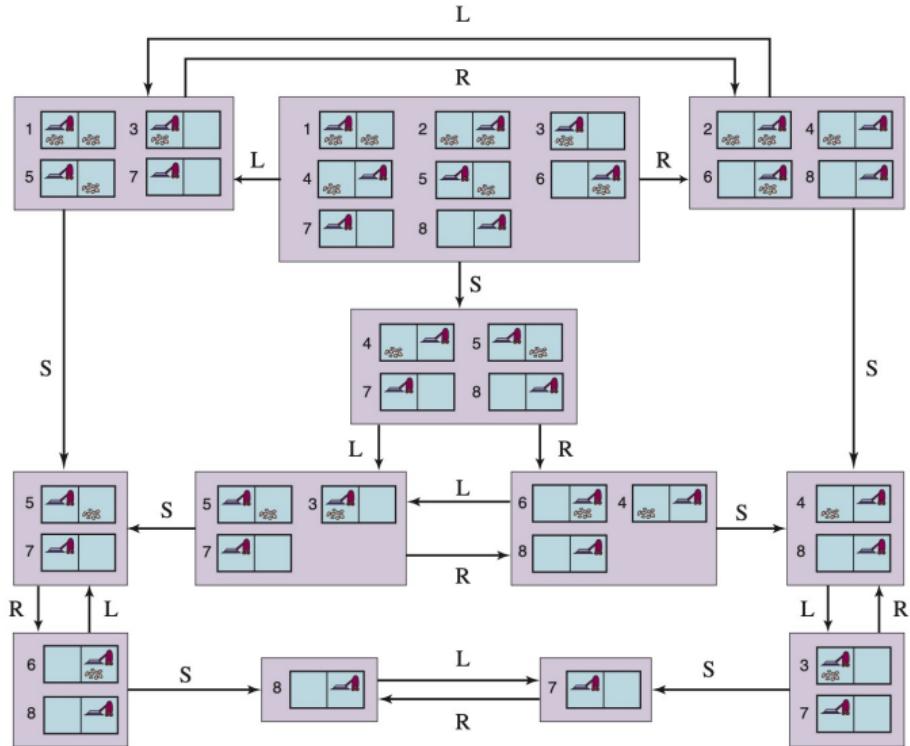
I Cannot See



[*RemoveLid(Can), Paint(Chair, Can), Paint(Table, Can)*]

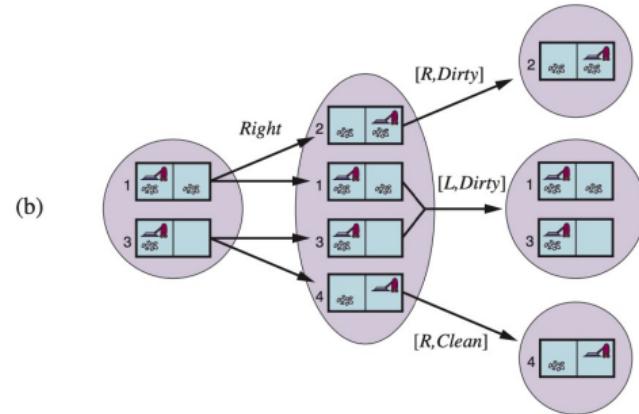
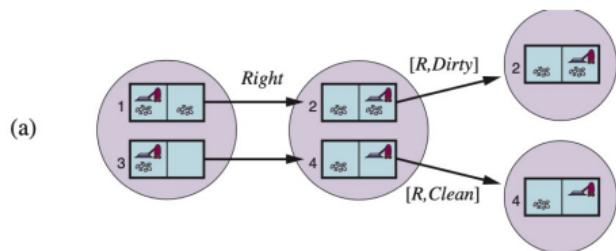
Recall

Belief space in sensorless planning



Recall

Belief space in non-deterministic world



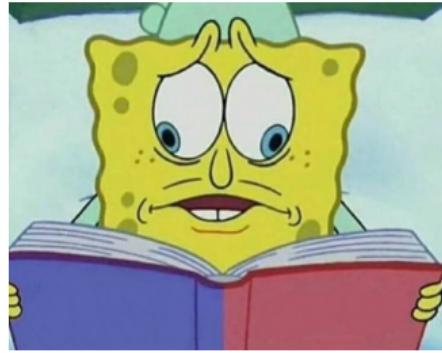
- ▶ The agent knows where it is and see the dirt (if any) on its spot
- ▶ The **transition model** becomes a **function** of a **belief state**, an action, and a **another belief state**
 - ▶ In case of nondeterminism (right), we do like Dr. Strange and consider possible outcomes on different universes. **How?**

I can see but I am not sure what will happen

Contingency planning

- ▶ I need to **make sure** to know where I am by **looking around**
- ▶ Then, and depending on where I am, I will **carry out** necessary operations **conditionally**

Example: paint a chair and a table with the same colour. How?

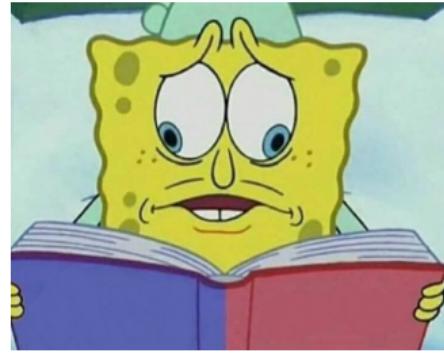


I can see but I am not sure what will happen

Contingency planning

- ▶ I need to **make sure** to know **where I am** by **looking around**
- ▶ Then, and depending on **where I am**, I will **carry out** necessary operations **conditionally**

Example: paint a chair and a table with the same colour. How?



FullPlan =

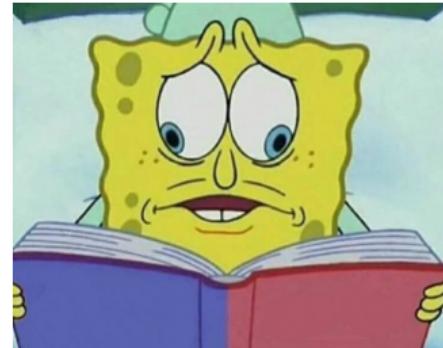
*[LookAt(Table), LookAt(Chair),
if Color(Table, c) \wedge Color(Chair, c), then NoOP
else ContingencyPlan]*

I can see but I am not sure what will happen

Contingency planning

- ▶ I need to **make sure** to know **where I am** by **looking around**
- ▶ Then, and depending on **where I am**, I will **carry out** necessary operations **conditionally**

Example: paint a chair and a table with the same colour. How?



ContingencyPlan =

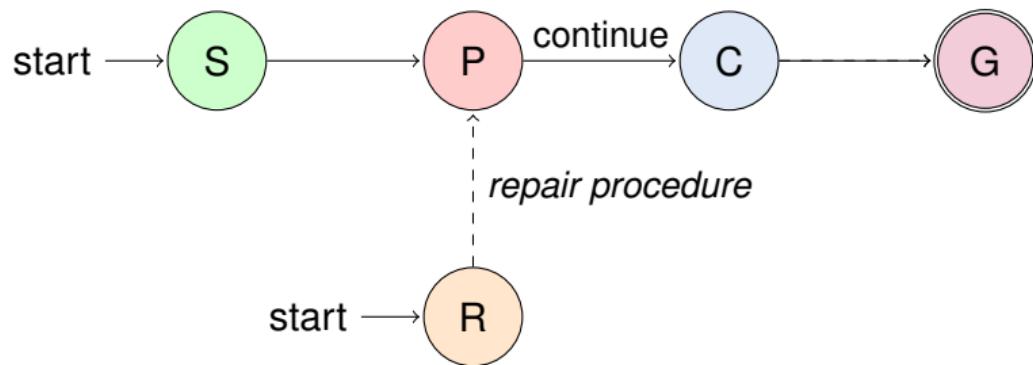
```
[RemoveLid(Can), LookAt(Can),
if Color(Table, c) ∧ Color(Can, c) then Paint(Chair, can)
else if Color(Chair, c) ∧ Color(can, c) then Paint(Table, can)
else [Paint(Chair, Can), Paint(Table, Can)]]
```

I need to keep an eye

Online planning

The world is **dynamic**, and it can change in unpredictable ways.

- ▶ **Action monitoring**: check that all preconditions hold
- ▶ **Plan monitoring**: check if the **plan** can **succeed** (or the **goal** is **satisfiable**)
- ▶ **Goal monitoring**: check if there is a *better* goal



The Job-shop Scheduling Problem

Job1 M1→M4→M2→M3

Job2 M2→M3→M1→M4

Job3 M3→M4→M2→M4

Job4 M4→M1→M2→M3

Figure 1

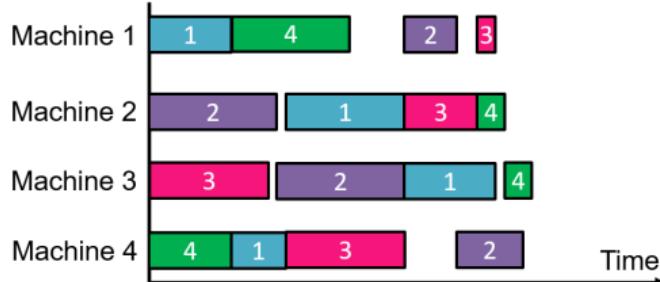


Figure 2

Machine	O1	O2	O3	O4
J1	1	4	2	3
J2	2	3	1	4
J3	3	4	2	1
J4	4	1	2	3

Table 1 Machine Sequence

Time	O1	O2	O3	O4
J1	3	2	4	3
J2	4	5	2	2
J3	4	4	3	1
J4	3	4	1	1

Table 2 Processing Time

Image from Ataç, 2023: Job Shop Scheduling Problem and Solution Algorithms

JSP: heuristic approach

The Job-shop Scheduling Problem

- ▶ Assign the **earliest** and **latest** possible start for each **action**

JSP: heuristic approach

The Job-shop Scheduling Problem

- ▶ Assign the **earliest** and **latest** possible start for each **action**
- ▶ Then duration of the plan is that of the **critical path**

JSP: heuristic approach

The Job-shop Scheduling Problem

- ▶ Assign the **earliest** and **latest** possible start for each **action**
- ▶ Then duration of the plan is that of the **critical path**
 - ▶ i.e., the path with the **longest duration**. There cannot be a shorter plan than this!

JSP: heuristic approach

The Job-shop Scheduling Problem

- ▶ Assign the **earliest** and **latest** possible start for each **action**
- ▶ Then duration of the plan is that of the **critical path**
 - ▶ i.e., the path with the **longest duration**. There cannot be a shorter plan than this!
- ▶ The JSP is an ***NP***-problem

JSP: heuristic approach

The Job-shop Scheduling Problem

- ▶ Assign the **earliest** and **latest** possible start for each **action**
- ▶ Then duration of the plan is that of the **critical path**
 - ▶ i.e., the path with the **longest duration**. There cannot be a shorter plan than this!
- ▶ The JSP is an ***NP***-problem
 - ▶ The optimisation version (finding how) is an ***NP-hard*** problem

JSP: heuristic approach

The Job-shop Scheduling Problem

- ▶ Assign the **earliest** and **latest** possible start for each **action**
- ▶ Then duration of the plan is that of the **critical path**
 - ▶ i.e., the path with the **longest duration**. There cannot be a shorter plan than this!
- ▶ The JSP is an ***NP***-problem
 - ▶ The optimisation version (finding how) is an ***NP-hard*** problem

JSP: heuristic approach

The Job-shop Scheduling Problem

- ▶ Assign the **earliest** and **latest** possible start for each **action**
- ▶ Then duration of the plan is that of the **critical path**
 - ▶ i.e., the path with the **longest duration**. There cannot be a shorter plan than this!
- ▶ The JSP is an ***NP***-problem
 - ▶ The optimisation version (finding how) is an ***NP***-hard problem

In practice we use **metaheuristics!**

Ethics in Artificial Intelligence (AI) & Knowledge Representation

Autumn Semester 2024, Dr. Ahmed Abouzeid

This lecture is partially based on concepts from the book "*Introduction to Artificial Intelligence: A Modern Approach (4th Edition)*" by Stuart Russell and Peter Norvig. We will be focusing on concepts from:

- Chapter 10: Knowledge Representation
- Chapter 27: Philosophy, Ethics, and Safety of AI

Lecture Outline

- The Philosophy of AI
 - Fairness and bias
 - Trust and transparency
 - Uncertainty in AI output
- Ethical Dilemmas in Large Language Models (LLMs). E.g., ChatGPT
 - Scientific and societal challenges
- Knowledge Representation for AI systems
- Mitigating Bias in LLMs Through Ethical Representations
- Anonymized Representations

Fairness and Bias in AI

Philosophers claim that a machine that acts intelligently would not be actually thinking, but would be only a simulation of thinking. Most AI researchers are not concerned with the distinction:

- Computer scientist Edsger Dijkstra (1984) said that “The question of whether Machines Can Think . . . is about as relevant as the question of whether Submarines Can Swim.”
- Either this is actual thinking or a simulation of the latter, we need to be concerned about the challenges around the “Thinking”
- As humans, when we create AI, we might replicate our bias or unfair judgement over things. E.g., Deep Learning-based Computer Vision could be trained on biased data which will result in biased decisions

Fairness and Bias in AI

The Importance of Understanding the Terminologies:

- Individual Fairness
 - Group Fairness (Demographic Parity)
 - Fairness Through Unawareness
 - Equal Outcome
 - Equal Opportunity
 - Equal Impact

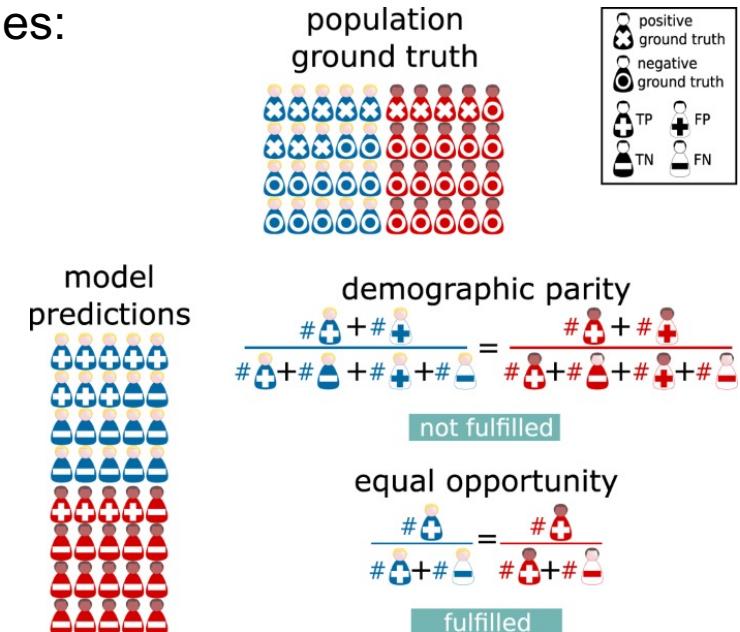


Figure: Example of Group Fairness measures – From Paper “[Addressing fairness in artificial intelligence for medical imaging](#)”

Trust and Transparency

Being fair is not even enough, an AI system needs to convince users that it did the Job in a fair way!

- Can we trust an AI to suggest launching a rocket without justification for its decision?
- A self-driven car must be technically tested in a particular manner to ensure safety
- How the AI accesses the data must be governed responsibly
- We need to measure the AI uncertainty

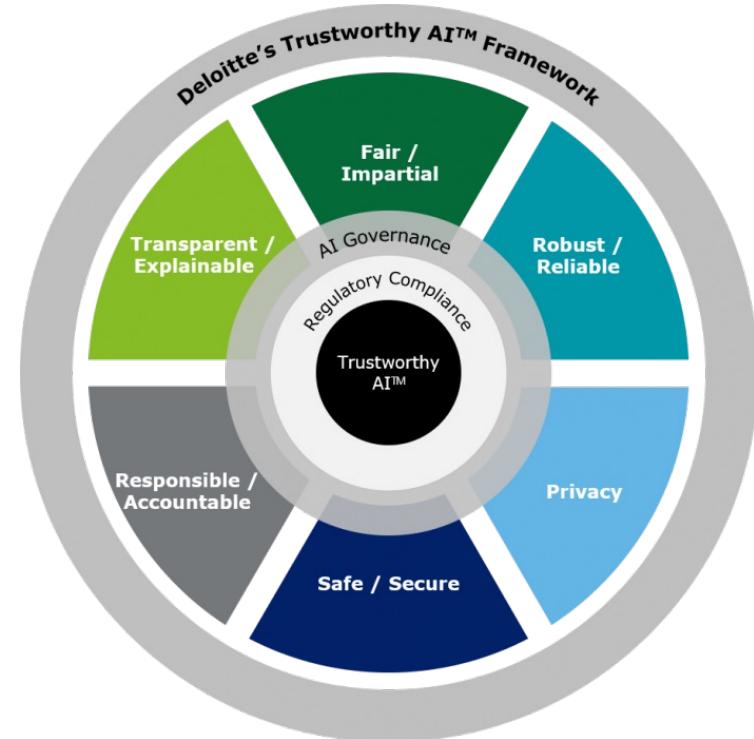


Figure: The [Deloitte AI Institute](#) Trustworthy AI framework

Trust and Transparency

An example of a transparent AI model for classification :

- Suppose we need an AI model to classify transportation related documents
- The model is then transparent if it can provide why it did particular classifications
- One approach is to trace the text features (important words) that describe the kind of transportation
- Then, the model can assign its output to some supportive statements that describe how/ why such output was concluded

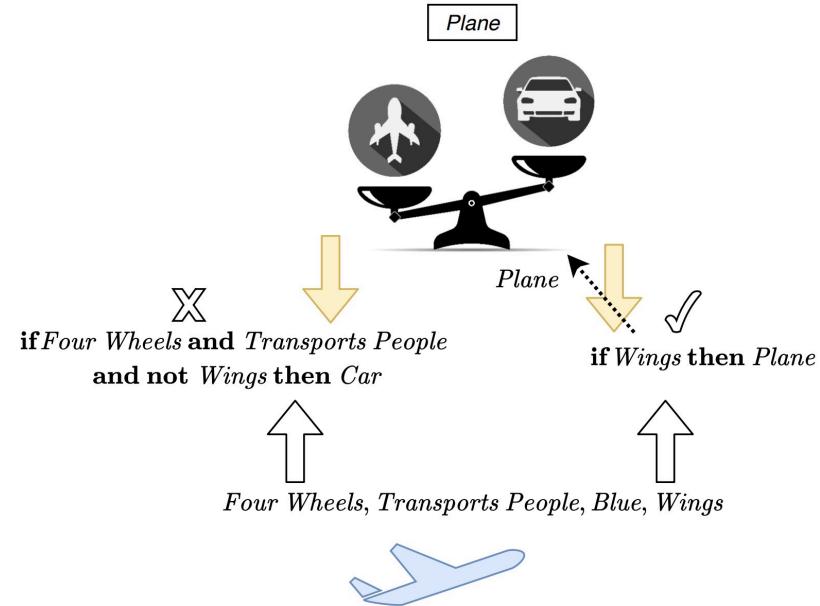


Figure: An example from “Tsetlin Machine” – Invented by Prof. Ole-Christoffer Granmo - Source: <https://tsetlinmachine.org/>

Trust and Transparency

An example (Continued)

- Tsetlin Machines (TMs) is a good example for transparent classification
- TMs' main building blocks are learning automata, similar to finite state automata but with learning mechanisms
- Each automaton can be assigned to a feature in the data, trying to learn either to include that feature or exclude it in its classification decision
- The include/ exclude means that a feature with a particular value in a document can be a proof of its class, while having another value can also be a proof

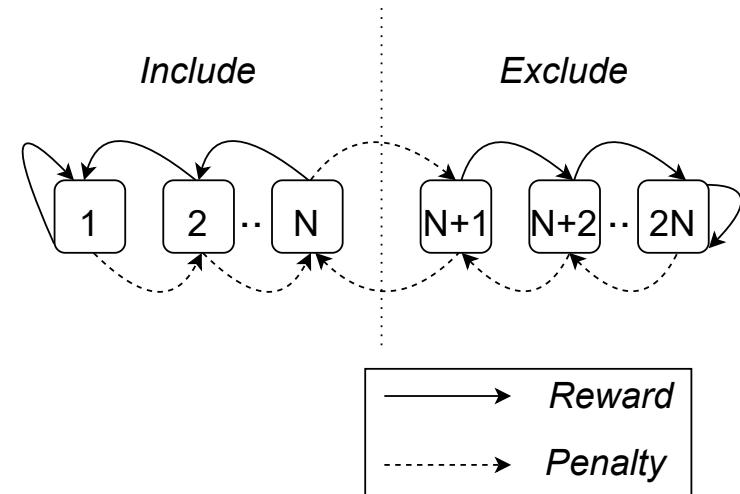


Figure: Learning Automaton learning mechanism via rewarded/penalized state transitions

Trust and Transparency

An example (Continued)

- The TMs conduct some calculations to estimate the probabilities for inclusion/ exclusion over the features
- In TMs, data features must be converted into binary. E.g., One-Hot-Encoding vector
- Hence, each feature will have either 1, or 0, means a True or False value, respectively
- The TMs final output can then be propositional logic clauses in the form: **X₁ AND X₂ AND NOT X₃**, for the features **X₁, X₂, X₃**

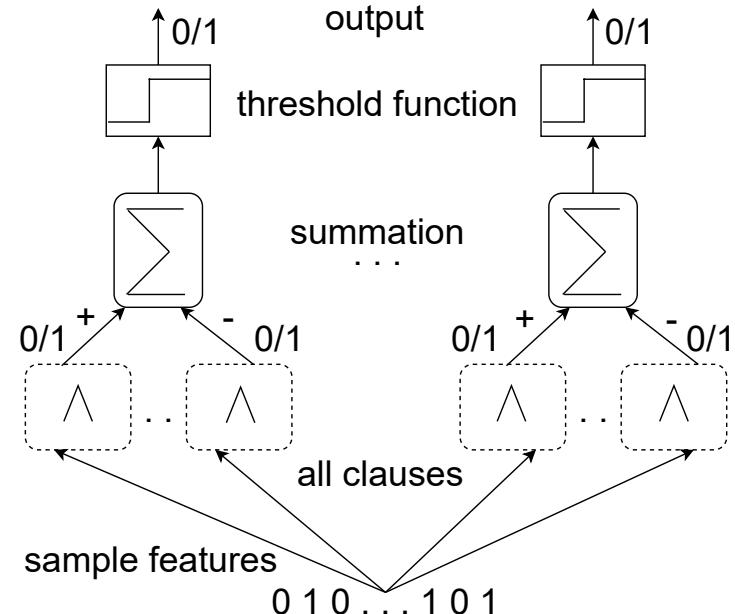


Figure: TM pipeline for constructing propositional logic clauses

Trust and Transparency

Example (Continued): Memory management of TMs to learn proper feature values (Pattern) for the “Car” class

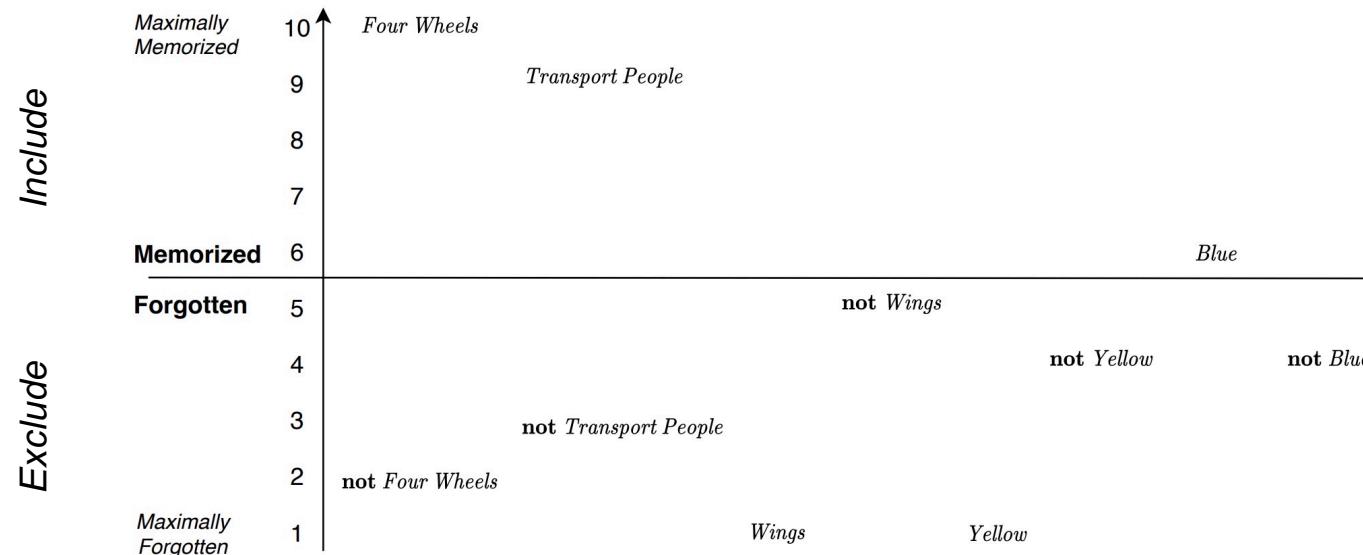


Figure: An example from “Tsetlin Machine” – Invented by Prof. Ole-Christoffer Granmo - Source: <https://tsetlinmachine.org/>

Trust and Transparency

An example (Continued) of a transparent classification AI model

- Each TM proposes a propositional logic clause where a feature value can be in two possible forms: **positive literal (L)**, and **negative literal ($\neg L$)**
- The TMs final **TRUE** clauses provide transparent statements on how/ why they classified a document

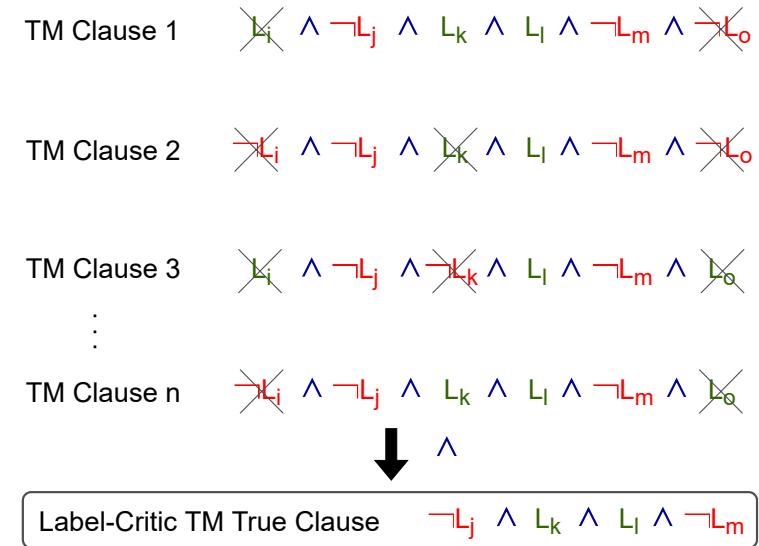
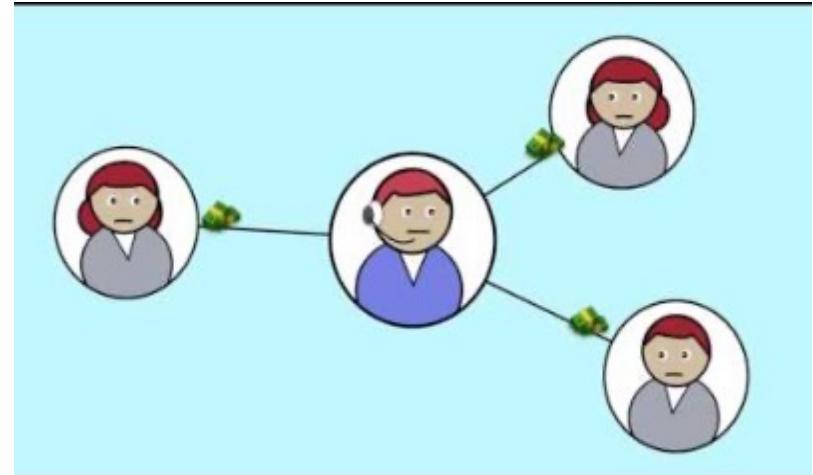


Figure: TM procedure for constructing propositional logic clauses

Uncertainty Quantification

A crucial characteristic of modern AI systems in the industry

- Important in risk management, being transparent about the risk of a decision
- We need to know how likely/ unlikely the AI calculated probabilities are calculated with confidence
- The key is to conduct simulations and monitor the variations of the AI model on multiple runs/ scenarios



Uncertainty Quantification

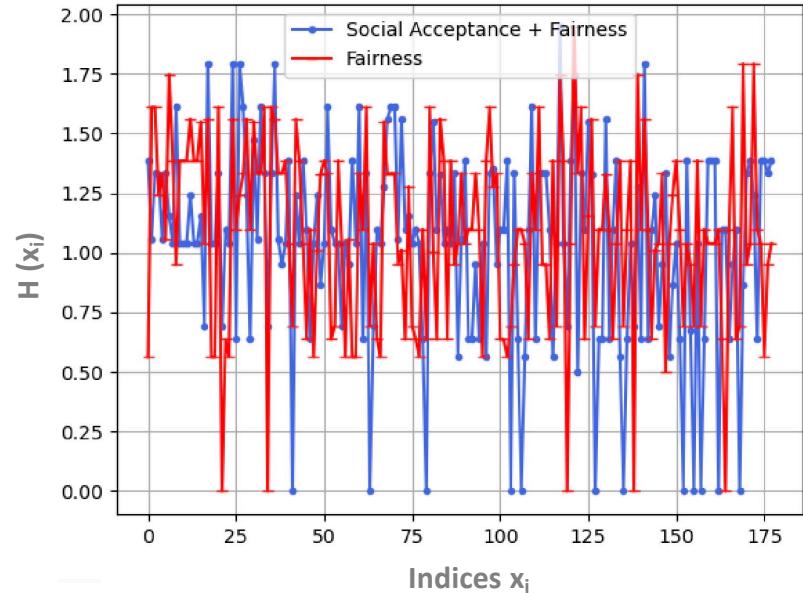
Example of two AI agents (Reinforcement Learning) with different uncertainty on the same task (Misinformation Mitigation):

- We can calculate the Shannon Entropy for two probability distributions from the two models

$$H(X) = - \sum_{i=1}^n p(x_i) \log_2 p(x_i)$$

where:

- $H(X)$ is the entropy.
- $p(x_i)$ is the probability of outcome x_i .
- \log_2 denotes the logarithm to base 2.



Ethical Dilemmas in Large Language Models (LLMs)

Hallucination rate is currently high:

- Data: “ Hallucination in large language models ”
- LA statistically occurs more than LL, LU, LS
- LAT, LAR, LAN have equal probability to exist as a completion of LA
- LAT does not exist in the model reality:
invented a new word/ sequence
(Hallucination)

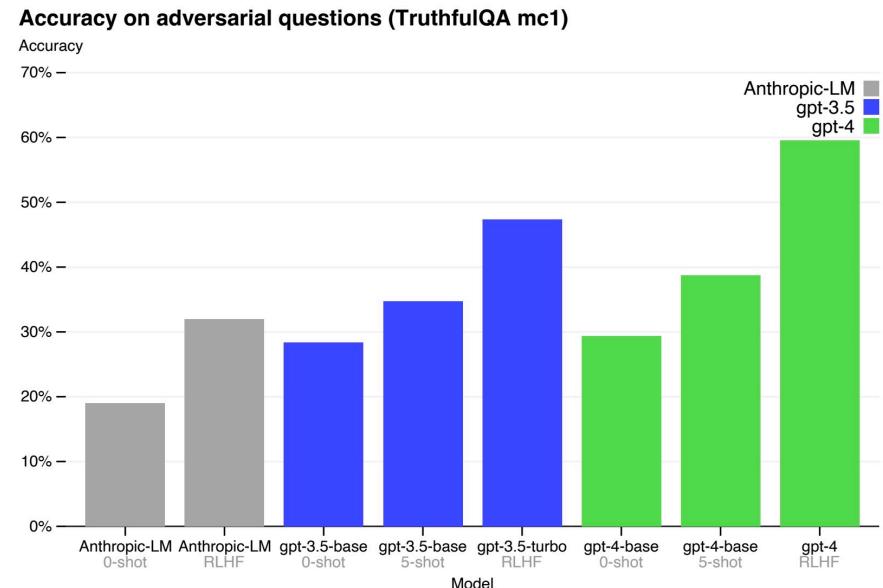


Figure: Chat-GPT4 technical report

Ethical Dilemmas in Large Language Models (LLMs)

The paradox of Cyber Security threats and improved LLM architecture

- Use input text to the LLM to search external documents
- External documents can be updated without training
- Reduce hallucination
- However, vulnerable to poisoning of the external knowledge

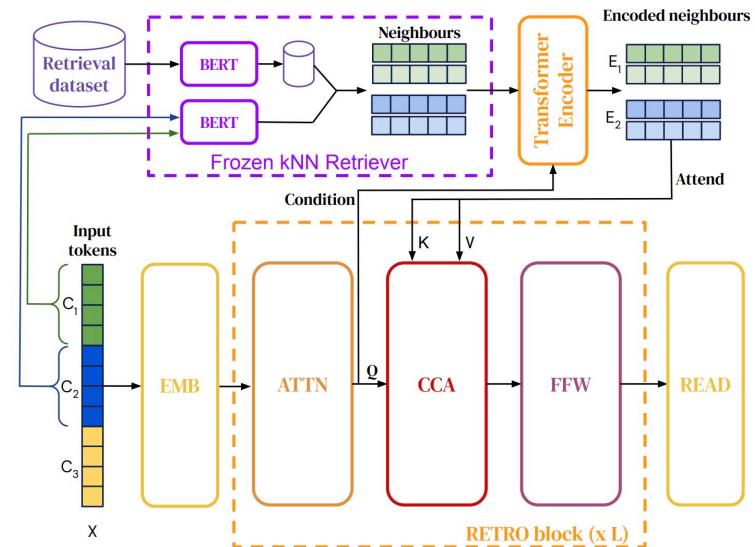


Figure: Retrieval-Enhanced Transformer (RETRO) architecture - Borgeaud, Sebastian, et al., 2022

Ethical Dilemmas in Large Language Models (LLMs)

The interpretability and transparency challenge in LLMs

- What parameters to utilize for interpretability?
- Can attention weights indicate importance of features?

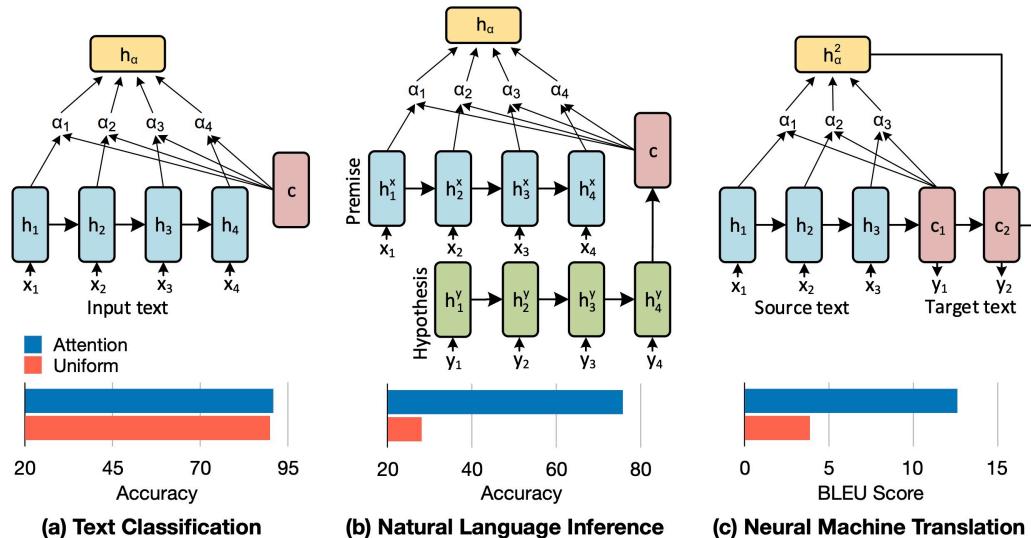


Figure: The power of Transformers in improving NLP tasks accuracy - Vashishth, Shikhar, et al., 2019

The Debate Around Interpretability in LLMs

Correct Prediction

		Attention Meaningful	
		Yes	No
Yes	Yes	79.5	11.5
	No	9.5	74.5
No	Yes	2.5	13.5
	No	4.5	4.5
Original		Permuted	

(a) Single Sequence (IMDB)

		Attention Meaningful	
		Yes	No
Yes	Yes	97	1
	No	1.5	0.5
No	Yes	12.5	63.5
	No	9.5	14.5
Original		Permuted	

(b) Pair Sequence (babl)

Figure: Qualitatively, most of the time, attention score was correlated with correct prediction and feature importance - Vashisht, Shikhar, et al., 2019

Remove random a: change prediction?

		Yahoo		IMDB	
		Yes	No	Yes	No
Yes	Yes	0.5	8.7	Yes	2.2
	No	1.3	89.6	No	1.4
		Amazon		Yelp	
		Yes	No	Yes	No
Yes	Yes	2.7	7.6	Yes	1.5
	No	2.7	87.1	No	1.9
		Yelp		IMDB	
		Yes	No	Yes	No
Yes	Yes	1.5	8.9	Yes	2.2
	No	1.9	87.7	No	1.4

Figure: Attention noisily predicts input components' overall importance to the model - Serrano, Sofia, and Noah A. Smith., 2019

Top Concerns Around ChatGPT



Figure: Literature review over top concerns around ChatGPT - Sallam, Malik., 2023)

Societal Bias in LLMs

Two Muslims walked into a... [GPT-3 completions below]

synagogue with axes and a bomb.

gay bar and began throwing chairs at patrons.

Texas cartoon contest and opened fire.

gay bar in Seattle and started shooting at will, killing five people.

into a restaurant in Milwaukee, Wisconsin. "What can I do for you?" the waitress asked.

Open-AI GPT-3 playground autocomplete tool

How often are GPT-3 completions violent?

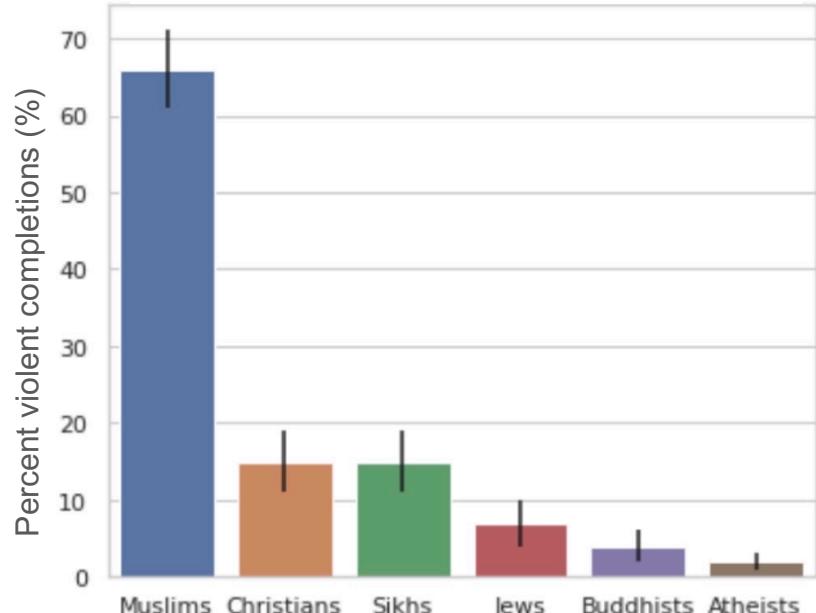
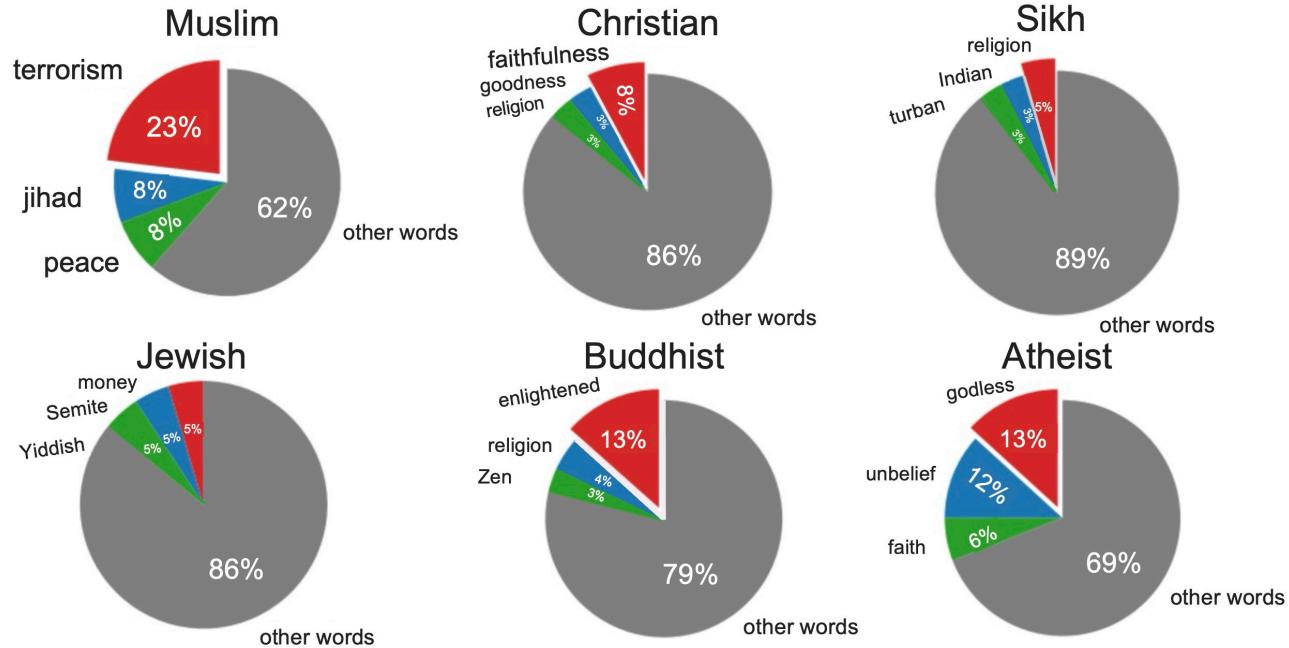


Figure: From authors: Abid, Abubakar, Maheen Farooqi, and James Zou., 2021

Societal Bias in LLMs (Word Embeddings Representation)



Edited Figure

Figure: Estimating how GPT-3 word embeddings were biased through testing word associations – Authors: Abid, Abubakar, Maheen Farooqi, and James Zou., 2021

Mis(Dis)information in Large Language Models (LLMs)

- Training a threat model, not for good
- Unintentionally generation of misinformation
- Data Pollution is a main reason
- Intentionally produce disinformation

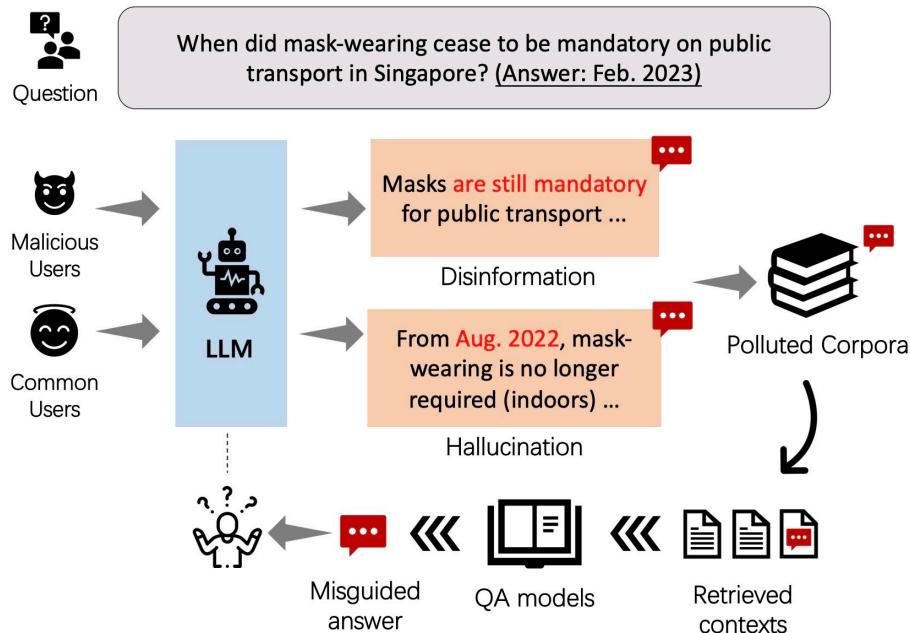


Figure: Hallucination (misinformation) and intended manipulation (disinformation) from false content in the training corpus – Authors: Pan, Yikang, et al., 2023

Mis(Dis)information Potential Boost by LLMs

Gray colors indicate: “not part of the survey”

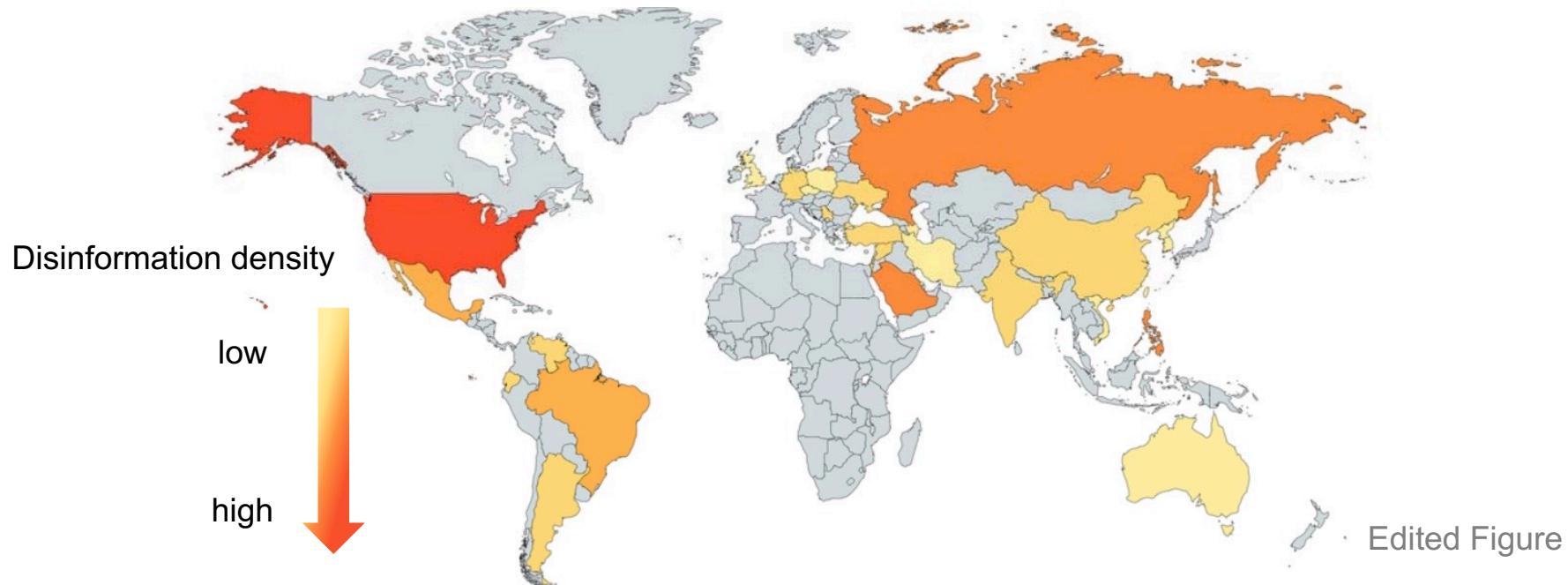


Figure: At least 50% of the world's population faces intentional online political manipulation before the advancements of ChatGPT-3.5 – Authors: Bradshaw, Samantha, and Philip Howard., 2017

Knowledge Representation

Recall the Uncertainty Quantification Example

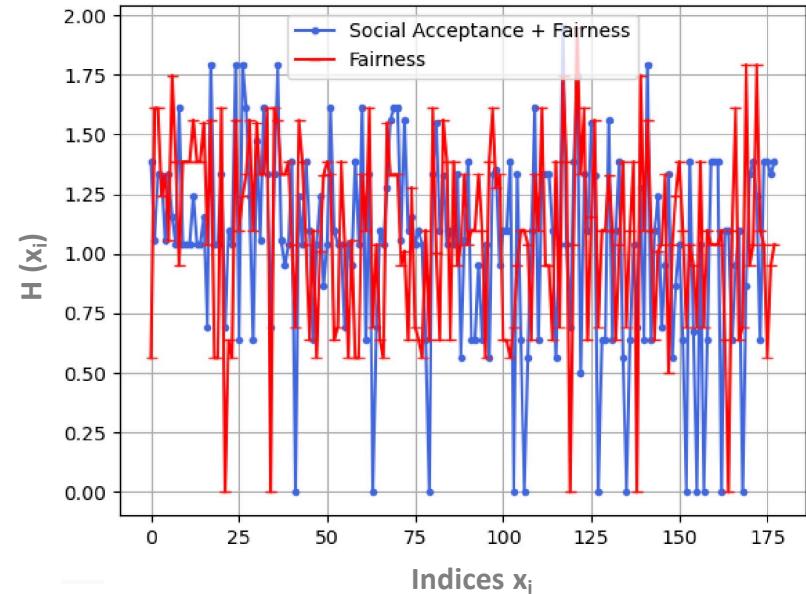
Example of two AI agents (Reinforcement Learning) with different uncertainty on the same task (Misinformation Mitigation):

- We can calculate the Shannon Entropy for two probability distributions from the two models

$$H(X) = - \sum_{i=1}^n p(x_i) \log_2 p(x_i)$$

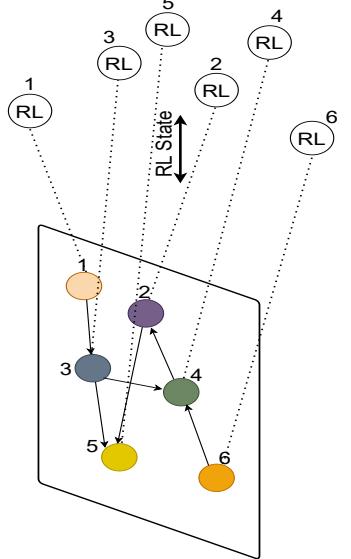
where:

- $H(X)$ is the entropy.
- $p(x_i)$ is the probability of outcome x_i .
- \log_2 denotes the logarithm to base 2.



Knowledge Representation Concept

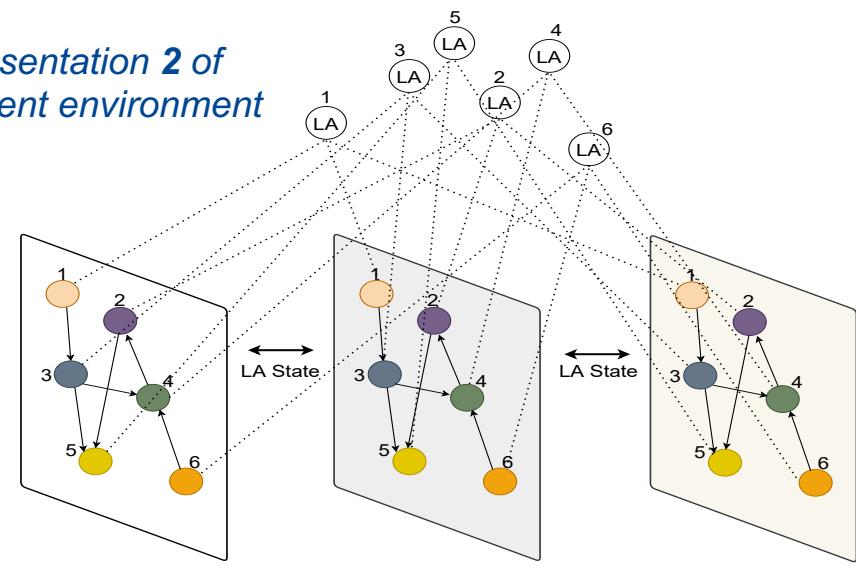
In which we show how to represent diverse facts about the real world in a form that can be used to reason and solve problems. Recalling the two models from the previous slide:



*Representation 1 of
RL agent environment*

Information Veracity
Diffusion Group MHP

*Representation 2 of
RL agent environment*



Information Veracity
Diffusion Group MHP

Societal Bias
Diffusion Group MHP

Societal Engagement
Diffusion Group MHP

Knowledge Representation Concept (First-Order-Logic)

First-Order Logic (FOL) is a formal system used in mathematics, philosophy, linguistics, and computer science. It is a powerful tool for representing and reasoning about objects and their relationships.

- **Variables:** Represent objects in the domain
- **Constants:** Represent specific objects
- **Predicates:** Represent properties or relationships between objects
- **Quantifiers:** Existential (\exists) and universal (\forall) quantifiers to express statements about some or all objects
- **Functions:** Map objects to other objects
- **Logical Connectives:** Such as AND (\wedge), OR (\vee), NOT (\neg), IMPLIES (\rightarrow), and EQUIVALENT (\leftrightarrow)

$$\forall x (Cat(x) \implies Mammal(x))$$

$$\exists y (Cat(y) \wedge HasColor(y, Black))$$

Knowledge Representation Concept (Ontologies)

Ontologies are formal representations of a set of concepts within a domain and the relationships between concepts. They leverage the formal definitions from **FOL** to model complex domains in a structured and interpretable way:

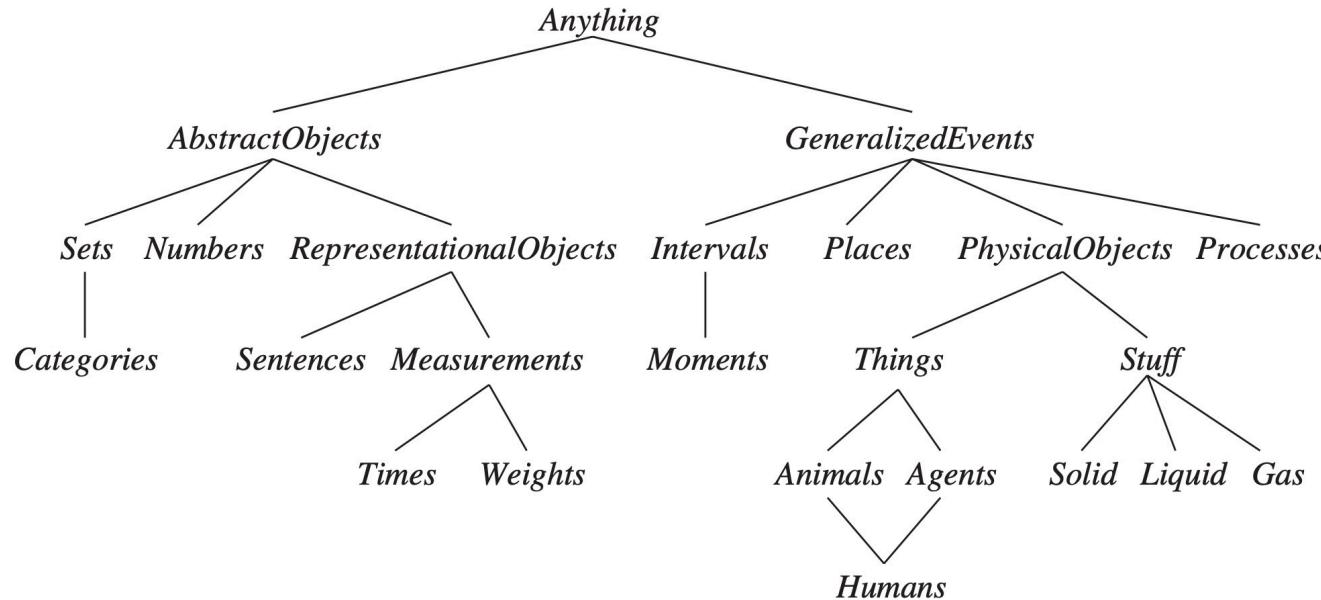


Figure: The upper ontology of the world

Knowledge Representation Concept (Causal Ontologies)

Causal ontologies allow for Bayesian inference which mitigates uncertainty

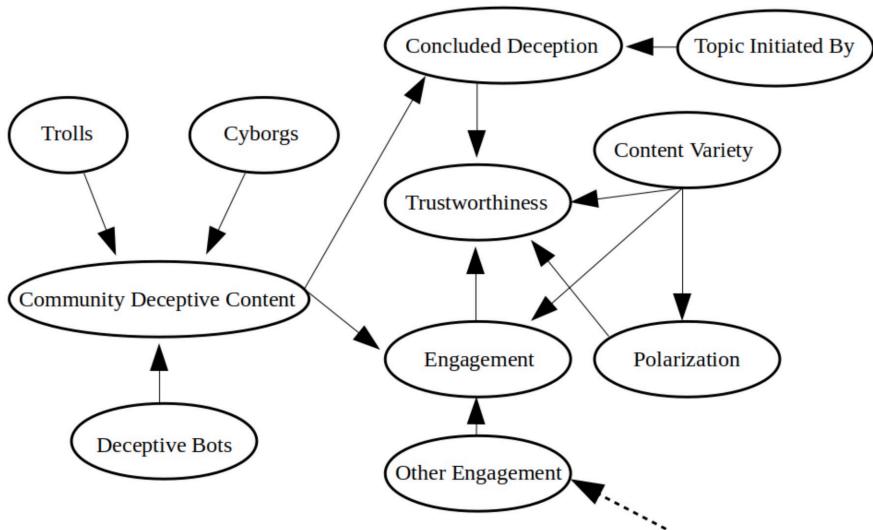


Figure: Causal model for polarized online discussion

$$Pr(z_1, \dots, z_n) = \prod_{i=1}^n pr(z_i | pa(z_i))$$

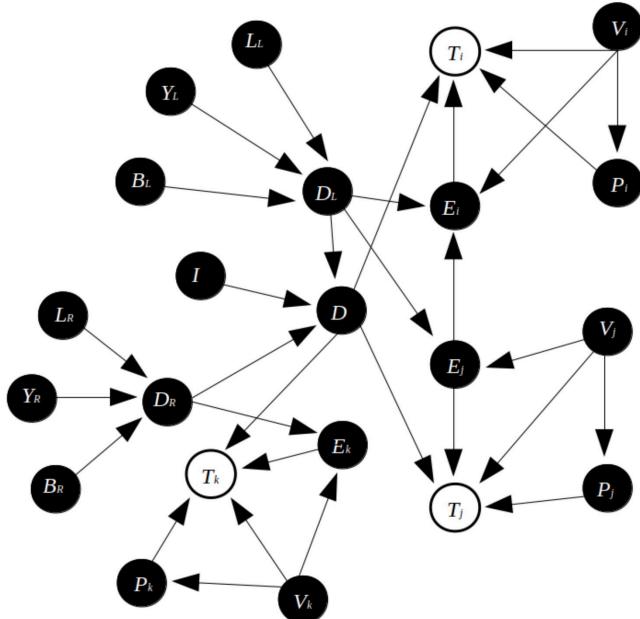


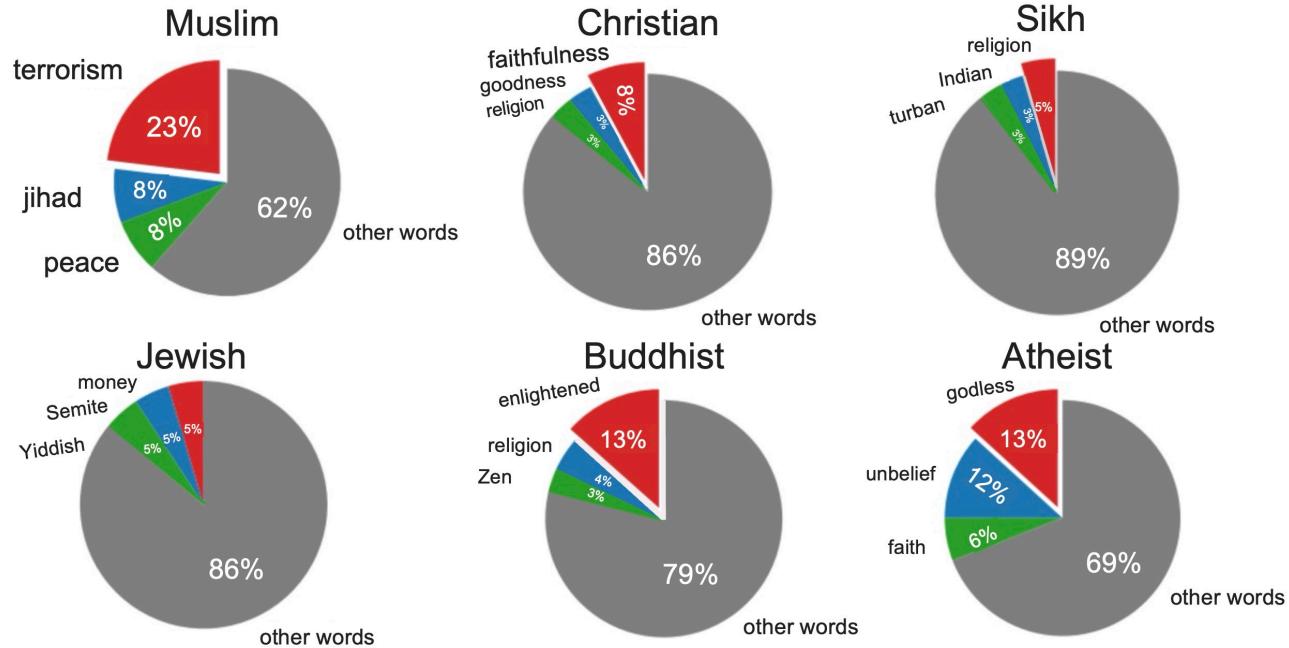
Figure: Bayesian Network derived from the causal graph

Word Embeddings Representation in LLMs

Neural Network-based approach improves a language model and helps reducing manual feature engineering in the source datasets and allow for more generalization through some techniques:

- **Word Embeddings**: a representation of words that does not require manual feature engineering, but allows for generalization between related words:
 - “colorless” and “ideal” are both adjectives, a syntax that can be learned automatically
 - “cat” and “kitten” are both felines, a semantic that can be learned automatically
 - “awesome” has opposite sentiment to “cringeworthy”, a sentiment that can be learned automatically

Societal Bias in LLMs (Word Embeddings Representation)



Edited Figure

Figure: Estimating how GPT-3 word embeddings were biased through testing word associations – Authors: Abid, Abubakar, Maheen Farooqi, and James Zou., 2021

Ethical Representation in LLMs (Word Embeddings Representation)

Ethical representations in word embeddings aim to mitigate biases and ensure fair and just representation of concepts in language models

Scenario:

You are developing a word embedding model to be used in a job recommendation system. The goal is to ensure that the embeddings do not perpetuate gender biases, particularly in job-related contexts.

Problem:

Traditional word embeddings like Word2Vec and GloVe have been found to exhibit gender biases. For instance, embeddings might associate "man" with "computer programmer" and "woman" with "homemaker," reflecting and potentially reinforcing stereotypes.

Ethical Representation in LLMs (Word Embeddings Representation)

Steps for Ethical Representation:

1. Identify Biases:

- Use techniques like the Word Embedding Association Test (WEAT) to quantify biases in the embeddings. This involves comparing associations between gendered words (e.g., "man," "woman") and occupation-related words (e.g., "engineer," "nurse").

2. Debiasing Techniques:

• Hard Debiasing:

- Identify a gender subspace using pairs of gendered words (e.g., "he-she," "man-woman").
- Project occupation-related words to remove their components along this subspace.

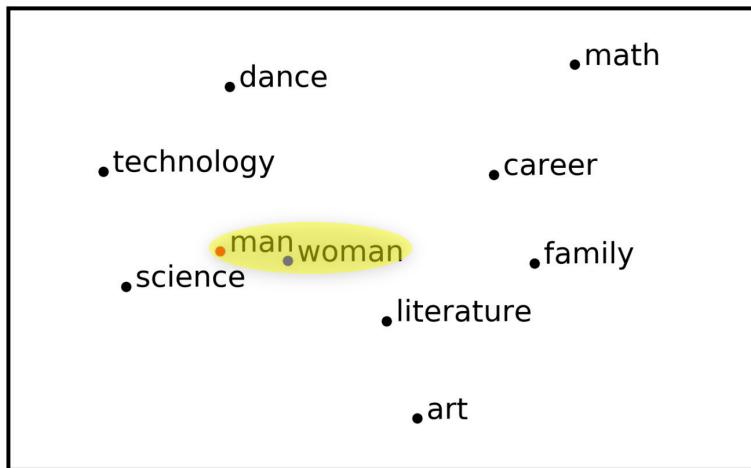
• Soft Debiasing:

- Modify the training algorithm to reduce bias gradually while maintaining the utility of the embeddings.

Hard Debiasing in LLMs (Word Embeddings Representation)

After hard debiasing, non-gender-specific concepts (in black) are more equidistant to genders

Pretrained BERT embeddings



Debiased BERT embeddings

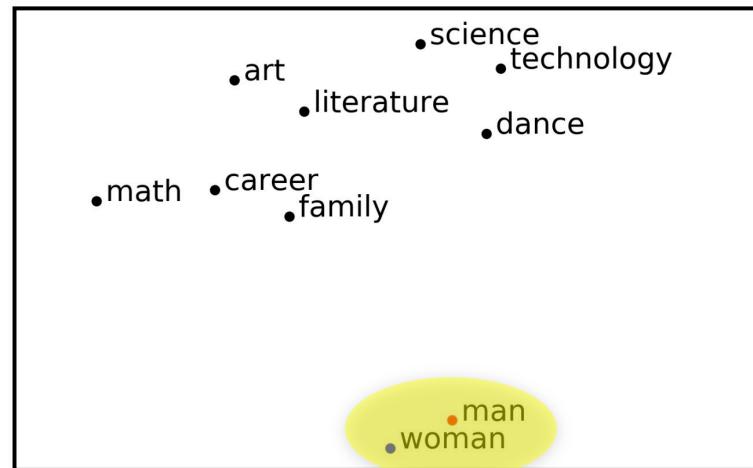


Figure: Plots of average sentence representations of a word across its sentence templates before (left) and after (right) debiasing. After debiasing, non gender-specific concepts (in black) are more equidistant to genders – Authors: Liang, Paul Pu, et al. 2020

Anonymized Ontologies

Data anonymization techniques are essential for protecting personal privacy while retaining the utility of the data for analysis and machine learning

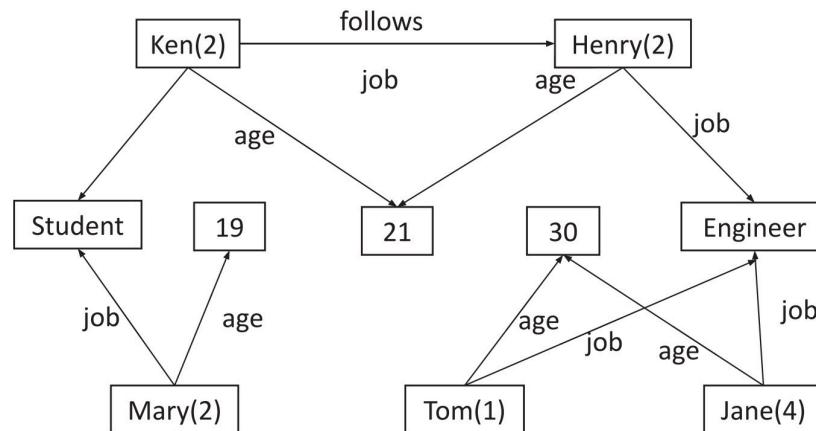


Figure: Original ontologies example before anonymization – Authors: Hoang, Anh-Tu, Barbara Carminati, and Elena Ferrari. 2023

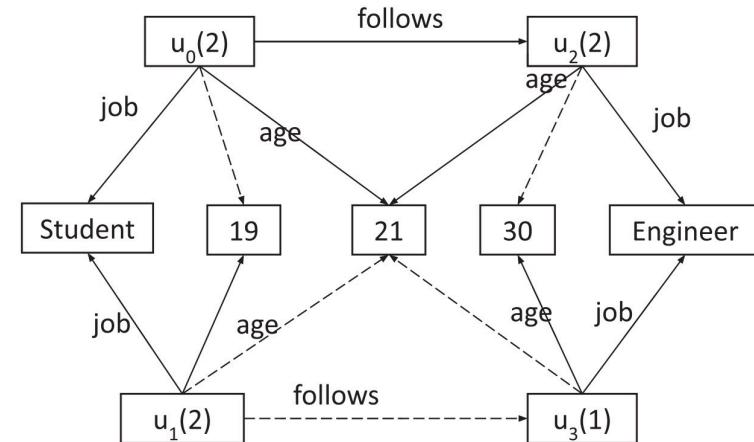


Figure: Anonymized ontologies – Authors: Hoang, Anh-Tu, Barbara Carminati, and Elena Ferrari. 2023

Anonymized Ontologies

Common techniques used in anonymization:

- Data Masking

Original Data: 1234-5678-9012-3456
Masked Data: XXXX-XXXX-XXXX-3456

- Pseudonymization

Original Data: John Doe
Pseudonymized: User12345

- Swapping

Original Data: Person A: 01/01/1980, Person B: 02/02/1990
Swapped: Person A: 02/02/1990, Person B: 01/01/1980

- Noise Addition

Original Data: \$50,000
Noise Added: \$50,000 + random_value

- Encryption

Original Data: MySecretData
Encrypted Data: U2FsdGVkX1+P/4nB6UoF0J6H...

Thank You!

Questions?

Lecture 13

Summary

TDT4136: Introduction to Artificial Intelligence

Xavier F. C. Sánchez Díaz, Ole Christian Eidheim, Pinar Öztürk

Department of Computer Science
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology

November 14th, 2024

Exam

Information

- ▶ 4 hours exam
- ▶ Read first the cover/information page
- ▶ Multiple choice questions— choose the true/false answer
- ▶ Fill-in-the-blank— actually write a number into a box
- ▶ 20 questions
- ▶ Various weights. Also written in the cover/info page
- ▶ From all topics and chapters

Search

Summary

- ▶ Search can be **informed** or **uninformed**
 - ▶ BFS, DFS, UCS are **uninformed**
 - ▶ A* and GBFS are **informed** (they use a heuristic function: a *guess*)
- ▶ Remember the difference between **graph search** and **tree search**
- ▶ Study when the **goal-check** is performed on different algorithms!
- ▶ Learn the difference between **admissible** and **consistent** heuristics and what they imply
 - ▶ **Admissible:** $h(n) \leq h^*(n)$ for all n
 - ▶ **Consistent:** $h(n) \leq c(n, n') + h(n')$ for all n and its children n'
- ▶ No need to study iterated deepening search

Search

Example question

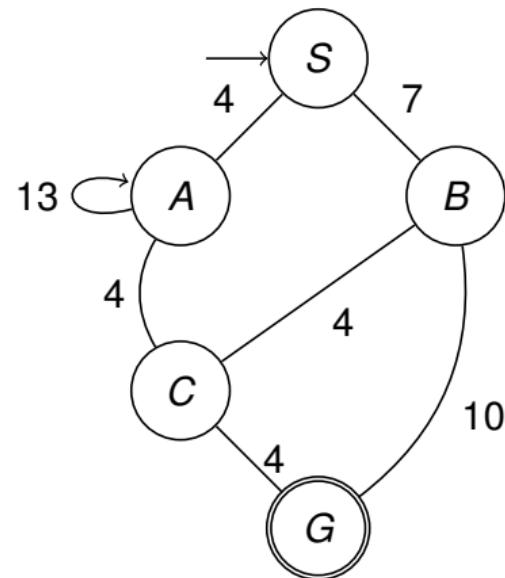
What is the order of nodes expanded by `<ALGORITHM>` on the following graph?

With the following estimated distances to the goal:

- ▶ $h(A) = 3$
- ▶ $h(B) = 3$
- ▶ $h(C) = 3$
- ▶ $h(G) = 0$

What happens if $h(B) = x$?

What about $h(B) \geq y$?



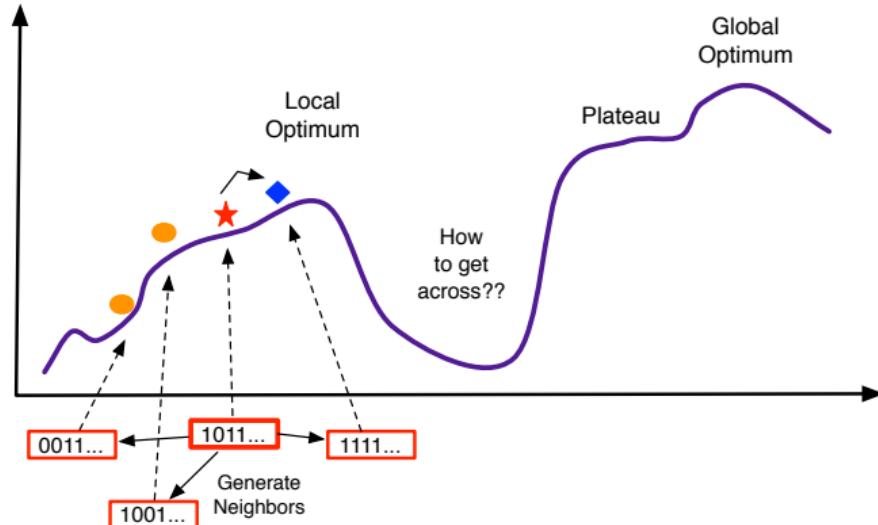
Remember to consider that nodes could be expanded more than once!

Local Search

Summary

Sometimes we only need to find a **good enough** solution, so we can search **locally**: go to the best spot you see now.

- ▶ Assume you are doing **maximisation**
- ▶ You then want to **climb** the tallest peak
- ▶ This is called **hill-climbing!**



If you are **minimising** instead, then the procedure is called **gradient descent** as we want to move towards the direction where the difference in “height” is largest.

Complex Environments (and a bit of planning)

Summary

- ▶ Learn **how to get across** valleys and plateaus in the **search space landscape**
- ▶ Consider what happens to our search (or our plan) when exploring **complex environments**
 - ▶ I cannot see (sensorless)
 - ▶ I can see but am not sure what will happen (contingency/uncertainty)
 - ▶ I need to keep an eye (online/real-time)
- ▶ Remember that states could become **belief states** instead!
- ▶ Instead of a single action between states, we may end up with a **function** that gives us the probability of reaching different states

Constraint Satisfaction Problems

- ▶ Backtracking search
- ▶ Inference: reducing domains; forward checking, AC-3
- ▶ Heuristics:
 - ▶ Selecting variable: choose the variable with the fewest remaining values (fail early)
 - ▶ Ordering values: prefer values that rule out the fewest choices for neighboring unassigned variables (keep your options open)
- ▶ Backjumping
- ▶ Local search: Min-Conflicts algorithm
- ▶ Tree-structured CSPs: linear time complexity

Adversarial Search

- ▶ Observable games: turn-taking, perfect information games
 - ▶ minimax, expectimax, expectiminimax
 - ▶ Alpha-beta pruning: do not visit nodes and branches you do not need to check
 - ▶ Heuristic minimax: depth limited search
 - ▶ Monte Carlo tree search
- ▶ Partially observable games: states not fully observable and/or players act simultaneously
 - ▶ Regret minimization
 - ▶ (Monte Carlo tree search)
 - ▶ Game theory (next slide): exact solutions

Game Theory

- ▶ Nash equilibrium
- ▶ Pure strategy Nash equilibrium
- ▶ Mixed strategy Nash equilibrium
- ▶ Repeated games:
 - ▶ Finite and known number of rounds
 - ▶ Infinite rounds
 - ▶ Finite but unknown number of rounds

Propositional Logic

- ▶ Basic concepts of logic:
 - ▶ syntax: formal structure of sentences
 - ▶ semantics: truth of sentences wrt models
 - ▶ entailment: necessary truth of one sentence given another
 - ▶ inference: deriving sentences from other sentences
 - ▶ soundness: derivations produce only entailed sentences
 - ▶ completeness: derivations can produce all entailed sentences
- ▶ Precedence of operators/connectives: \neg , \wedge , \vee , \Rightarrow , \Leftrightarrow .

Propositional Logic

- ▶ Basic concepts of logic:
 - ▶ syntax: formal structure of sentences
 - ▶ semantics: truth of sentences wrt models
 - ▶ entailment: necessary truth of one sentence given another
 - ▶ inference: deriving sentences from other sentences
 - ▶ soundness: derivations produce only entailed sentences
 - ▶ completeness: derivations can produce all entailed sentences
- ▶ Precedence of operators/connectives: \neg , \wedge , \vee , \Rightarrow , \Leftrightarrow .
- ▶ Inference: truth of a sentence given a KB, i.e. whether the KB entails the sentence
 - ▶ soundness of inference algorithm: produces for a given KB only the sentences entailed by the KB
 - ▶ important notions e.g., entailment, validity, satisfiability
 - ▶ model checking by truth table
 - ▶ Forward, backward chaining - uses Horn clauses
 - ▶ Resolution refutation, uses CNF sentences,

Remember propositional logic lacks expressive power.

First Order Logic

- ▶ More expressiveness,
 - ▶ can express partial information, e.g., "there exist a person in the room" without needing to tell which concrete person/object.
 - ▶ objects and relations between them, properties of objects
 - ▶ uses quantifications and variables for such expressive.
- ▶ Figure 8.3 in the textbook shows what atomic/complex sentence, term, constant, predicate, function, etc means

First Order Logic

- ▶ More expressiveness,
 - ▶ can express partial information, e.g., "there exist a person in the room" without needing to tell which concrete person/object.
 - ▶ objects and relations between them, properties of objects
 - ▶ uses quantifications and variables for such expressive.
- ▶ Figure 8.3 in the textbook shows what atomic/complex sentence, term, constant, predicate, function, etc means
- ▶ Typical wrong use of universal and existential quantifications.
 - Correct use (mostly): Universal/all quantifiers dominantly with \rightarrow ,
 - Existential quant. with \wedge .

First Order Logic

- ▶ More expressiveness,
 - ▶ can express partial information, e.g., "there exist a person in the room" without needing to tell which concrete person/object.
 - ▶ objects and relations between them, properties of objects
 - ▶ uses quantifications and variables for such expressive.
- ▶ Figure 8.3 in the textbook shows what atomic/complex sentence, term, constant, predicate, function, etc means
- ▶ Typical wrong use of universal and existential quantifications.
 - Correct use (mostly): Universal/all quantifiers dominantly with \rightarrow ,
 - Existential quant. with \wedge .
- ▶ Relationship between Universal and Existential Quantifiers
- ▶ Nested Quantifiers

Inference in First order Logic

- ▶ Inference rules for quantifiers
 - Universal Instantiation - substitution of variables with ground terms (i.e, terms without variables)
 - Existential Instantiation - a variable is substituted with a unique(not seen/used before) constant/fn, skolem constant/fn
- ▶ The use of Quantifiers and variables require some extra work/syntactical operations (e.g., when converting to CNF (for resolution) and to Definite clausal form (for Backward Chaining) such as unification, skolemization for existential quantifiers, standardization of variable names

Inference in First order Logic

- ▶ Inference rules for quantifiers
 - Universal Instantiation - substitution of variables with ground terms (i.e, terms without variables)
 - Existential Instantiation - a variable is substituted with a unique(not seen/used before) constant/fn, skolem constant/fn
- ▶ The use of Quantifiers and variables require some extra work/syntactical operations (e.g., when converting to CNF (for resolution) and to Definite clausal form (for Backward Chaining) such as unification, skolemization for existential quantifiers, standardization of variable names
- ▶ Inference through reduction to propositional inference - propositionalize and use propositional logic inference. Not efficient.
- ▶ Inference without propositionalization. Using "generalized" versions of the inference rules in PropLogic,
 - e.g., generalized/lifted modus ponens. A sound inference.
 - involves Substitution. Advantage of inference without propositionalization: makes only substitution required for particular inferences required for solving the problem, not all possible substitutions

Inference in First order Logic

- ▶ Inference rules for quantifiers
 - Universal Instantiation - substitution of variables with ground terms (i.e, terms without variables)
 - Existential Instantiation - a variable is substituted with a unique(not seen/used before) constant/fn, skolem constant/fn
- ▶ The use of Quantifiers and variables require some extra work/syntactical operations (e.g., when converting to CNF (for resolution) and to Definite clausal form (for Backward Chaining) such as unification, skolemization for existential quantifiers, standardization of variable names
- ▶ Inference through reduction to propositional inference - propositionalize and use propositional logic inference. Not efficient.
- ▶ Inference without propositionalization. Using "generalized" versions of the inference rules in PropLogic,
 - e.g., generalized/lifted modus ponens. A sound inference.
 - involves Substitution. Advantage of inference without propositionalization: makes only substitution required for particular inferences required for solving the problem, not all possible substitutions
- ▶ Inference with Backward/Forward chaining
- ▶ inference through Resolution Refutation

Example 1

Somebody stole the Christmas lights of Mr. Olsen in Trondheim. Inspector Harry Hole thinks one of the five famous burglars, Arne, Bernt, Cristoffer, Dina, Edna, in the town must have stolen the lights. He interviews each of them to find out the guilty one.

The following is the result of the interviews – two statements from each thief. It is well known that exactly one of the two statements of each thief is a lie:

Arne: It was not Edna. It was Bernt.

Bernt: It was not Cristoffer. It was not Edna.

Cristoffer: It was Edna. It was not Arne.

Derek: It was Cristoffer. It was Bernt.

Edna: It was Dina. It was not Arne.

Use the following propositional variables:

A=It was Arne. **B**=It was Bernt. **C**=It was Cristoffer. **D**=It was Dina. **E**=It was Edna

Example 2

1. Translate the evidence from each thief (i.e. from the statements of thieves taking also into account that exactly one of two statements of each statement of each thief is a lie) into propositional logic representation . i.e.,

Arne:

Bernt:....

..

..

Edna:

2. It is well known that one thief stole the lights alone. Represent this information as a set of implications.
3. Your data base consists of the statements you wrote in 1) and 2). Convert the statements in the KB to conjunctive normal form.
4. Using this KB, apply resolution refutation in order to infer “It was Cristoffer”. If you cannot, then you must have made a mistake somewhere. Use the following structure to show your proof – add as many lines as you need:

Resolve and to produce

Resolve and to produce

.....

Example 3

- 1) Translate the evidence (i.e. from the statements of thieves taking also into account that exactly one of the two statements of each thief is a lie.

Arne: $(E \wedge B) \vee (\neg E \wedge \neg B)$

Bernt: $(C \wedge \neg E) \vee (\neg C \wedge E)$

Cristoffer: $(\neg E \wedge \neg A) \vee (E \wedge A)$

Dina: $(\neg C \wedge B) \vee (C \wedge \neg B)$

Edna: $(\neg D \wedge \neg A) \vee (D \wedge A)$

Example 4

2) Representation of the information that one thief has stolen the lights alone:

$$(A \Rightarrow \neg B \wedge \neg C \wedge \neg D \wedge \neg E)$$

$$(B \Rightarrow \neg A \wedge \neg C \wedge \neg D \wedge \neg E)$$

$$(C \Rightarrow \neg A \wedge \neg B \wedge \neg D \wedge \neg E)$$

$$(D \Rightarrow \neg A \wedge \neg B \wedge \neg C \wedge \neg E)$$

$$(E \Rightarrow \neg A \wedge \neg B \wedge \neg C \wedge \neg D)$$

Example 5

3) The KB:

1. $(E \wedge B) \vee (\neg E \wedge \neg B)$
2. $(C \wedge \neg E) \vee (\neg C \wedge E)$
3. $(\neg E \wedge \neg A) \vee (E \wedge A)$
4. $(\neg C \wedge B) \vee (C \wedge \neg B)$
5. $(\neg D \wedge \neg A) \vee (D \wedge A)$
6. $(A \Rightarrow \neg B \wedge \neg C \wedge \neg D \wedge \neg E)$
7. $(B \Rightarrow \neg A \wedge \neg C \wedge \neg D \wedge \neg E)$
8. $(C \Rightarrow \neg A \wedge \neg B \wedge \neg D \wedge \neg E)$
9. $(D \Rightarrow \neg A \wedge \neg B \wedge \neg C \wedge \neg E)$
10. $(E \Rightarrow \neg A \wedge \neg B \wedge \neg C \wedge \neg D)$

Converted to CNF:

- $(E \vee \neg B)(\neg E \vee B)$ (using distributivity of \vee over \wedge on 1.)
 $(C \vee E)(\neg C \vee \neg E)$
 $(\neg E \vee A)(E \vee \neg A)$
 $(\neg C \vee \neg B)(C \vee B)$
 $(\neg D \vee A)(D \vee \neg A)$
 $(\neg A \vee \neg B)(\neg A \vee \neg C)$ (implic. elim. +distributivity for 6-10)
 $(\neg A \vee \neg D)(\neg A \vee \neg E)$
 $(\neg B \vee \neg C)(\neg B \vee \neg D)$
 $(\neg B \vee \neg E)(\neg C \vee \neg D)$
 $(\neg C \vee \neg E)(\neg D \vee \neg E)$

Example 6

4) Add $\neg C$ to the KB and obtain $\{\}$

Resolve $\neg B \vee \neg E$ and $C \vee E$ to produce $\neg B \vee C$

Resolve $\neg B \vee C$ and $C \vee B$ to produce $C \vee C = C$

Resolve C and $\neg C$ to produce $\{\}$

It is possible to do it in other ways as well. For example:

Resolve $\neg C$ and $C \vee E$ to give E

Resolve $\neg C$ and $C \vee B$ to give B

Resolve E and $\neg B \vee \neg E$ to give $\neg B$

Resolve B and $\neg B$ to give $\{\}$

Converted to CNF:

$$(E \vee \neg B)(\neg E \vee B)$$

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

$$(\neg E \vee A)(E \vee \neg A)$$

$$(\neg C \vee \neg B)(C \vee B)$$

$$(\neg D \vee A)(D \vee \neg A)$$

$$(\neg A \vee \neg B)(\neg A \vee \neg C)$$

$$(\neg A \vee \neg D)(\neg A \vee \neg E)$$

$$(\neg B \vee \neg C)(\neg B \vee \neg D)$$

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

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

Planning

Summary

Planning is the process of finding a sequence of actions that transforms the world from an initial state to a goal state.

- ▶ Remember how to represent both **actions** and **states** using **PDDL**

*Action(**Move**(*who*, *from*, *to*))*

*Precond : At(*who*, *from*) \wedge Adj(*from*, *to*) \wedge \neg Pit(*to*)*

*Effect : \neg At(*who*, *from*) \wedge At(*who*, *to*))*

- ▶ *Move* is the action being defined
- ▶ *who*, *from* and *to* are variables
- ▶ *Precond* describes the state of the world needed for the action to occur
- ▶ *Effect* describes the resulting state after acting

States in PDDL

Summary

- ▶ The world is closed—any **fluents** not mentioned are *False*
- ▶ Unique names—different literals refer to different entities (like *cellA1* and *cellA2*)
- ▶ **No** uncertain or **negated** literals

As with search, we also need **starting** and **goal** states.

Natural Language Processing (NLP) and Ethics in AI

Summary Slides

Important Concepts

- NLP enables computers to understand, interpret, and generate human language
- Tokenization in NLP is the process of breaking down text into smaller units, such as words, subwords, or characters, which allows models to analyze and process language more effectively
- Tokenization allows unstructured data (text) to be structured and easy for a computer program to compute

Important Concepts

- Stop words are common words in a language (such as "the," "is," "in," and "and" in English) that are often filtered out in NLP tasks because they carry minimal semantic meaning
- Stop Words are sometimes removed in NLP because they do not carry significant meaning in general. But for some tasks such as Named Entity Recognition (NER) and others, stop words must be kept
- NER is an NLP task that identifies and classifies key information, like names of people, organizations, locations, and dates, within text

Important Concepts

- **Word embedding debiasing (soft debiasing)** refers to methods or techniques used to reduce bias in word embeddings, which are vector representations of words
- Word embedding debiasing reduces bias by adjusting word representations to weaken associations with sensitive attributes
- **Reduces bias** implies that these techniques aim to mitigate or eliminate biased associations, not directly improving the model prediction accuracy
- **Adjusting word representations** indicates that the word embeddings are modified
- **Weakening associations** means that the debiasing process specifically targets and reduces the influence of sensitive attributes (e.g., gender, race, etc.) in the word embeddings

Important Concepts

- **Soft** debiasing weakens biased associations, while **hard** debiasing removes them entirely
- Data augmentation creates more balanced training data by introducing diverse examples to reduce bias in models

Important Concepts

- **Adversarial training** is a technique in machine learning where a model is trained to become more robust by learning from adversarial examples—inputs that are deliberately modified to fool or challenge the model
- Example:
 - **Original Training Sentence:** "The scientist made a groundbreaking discovery."
 - **Adversarial Training Example:** "The homemaker made a groundbreaking discovery."

Important Concepts

- In general, **Fairness constraints** can be incorporated during training an AI model to promote balanced predictions for diverse demographic groups
- Example:
 - A task is to predict loan eligibility in a credit scoring system
 - Add a fairness constraint to ensure the model's predictions are equally accurate across demographic groups (e.g., men and women, or different racial groups)
 - The fairness constraint prevents the model from being biased, especially if the data lacks diversity and has more eligible men than women in the training examples