

CSU33061 Artificial Intelligence I

Introduction

www.scss.tcd.ie/Tim.Fernando/AI

Key Phrases:

Can machines think?

- Turing test & ELIZA effect
- ChatGPT ... (generative AI)

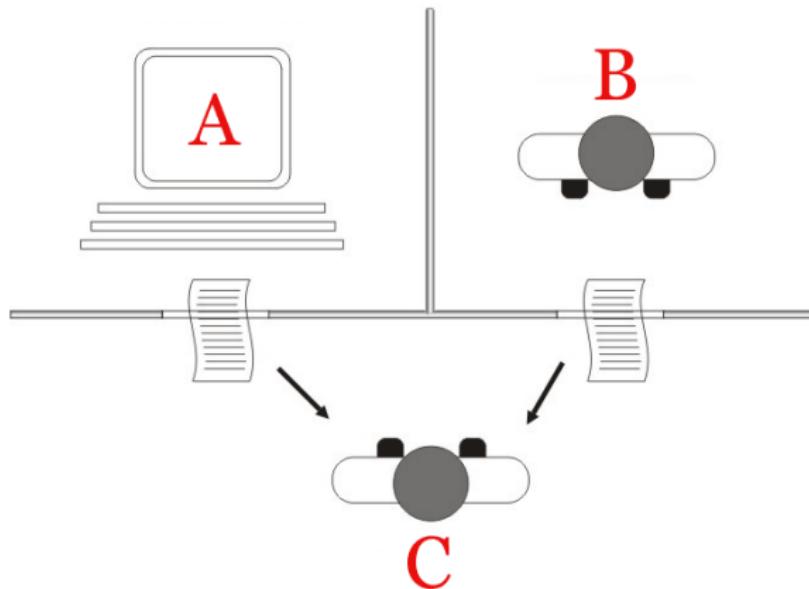
Agent & environment

- Cognitive Revolution & Big Data

Levels of intelligence

Can machines think? (Turing 1950)

Turing test: can C tell A from B?

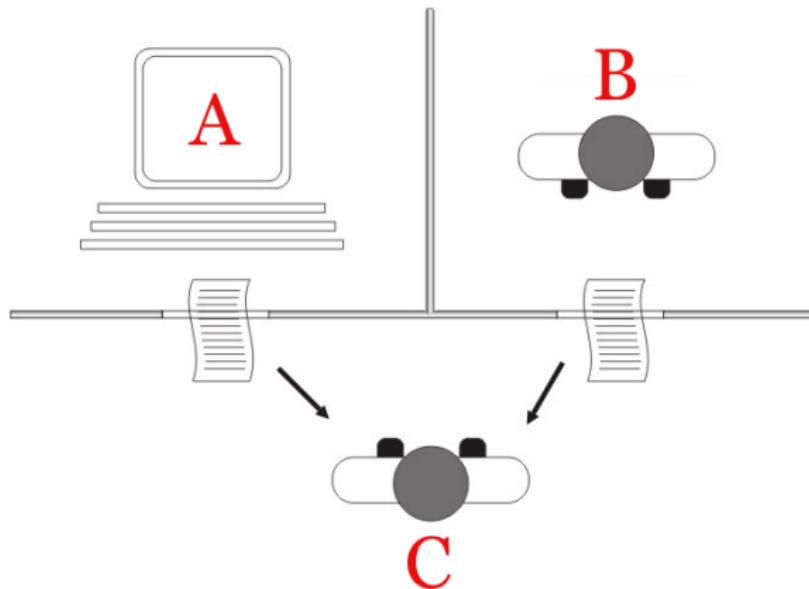


From Wikipedia, (Juan Alberto Sánchez Margallo)

Intelligence operationalized: subject to testing

Can machines think? (Turing 1950)

Turing test: can C tell A from B?



From Wikipedia, (Juan Alberto Sánchez Margallo)

Intelligence operationalized: subject to testing and **cheating**

ELIZA (Weizenbaum, 1964-66): artful deception

- use pattern matching and substitution to fake it (e.g. Prolog)

ELIZA (Weizenbaum, 1964-66): artful deception

- use pattern matching and substitution to fake it (e.g. Prolog)

ELIZA effect: humans anthropomorphise computers

ELIZA (Weizenbaum, 1964-66): artful deception

- use pattern matching and substitution to fake it (e.g. Prolog)

ELIZA effect: humans anthropomorphise computers

CAUTION: programs are quite different from humans

Siri rage (Urban dictionary):

When you get enraged because Siri just doesn't get it.

ELIZA (Weizenbaum, 1964-66): artful deception

- use pattern matching and substitution to fake it (e.g. Prolog)

ELIZA effect: humans anthropomorphise computers

CAUTION: programs are quite different from humans

Siri rage (Urban dictionary):

When you get enraged because Siri just doesn't get it.

Fooling fools is easy

ELIZA (Weizenbaum, 1964-66): artful deception

- use pattern matching and substitution to fake it (e.g. Prolog)

ELIZA effect: humans anthropomorphise computers

CAUTION: programs are quite different from humans

Siri rage (Urban dictionary):

When you get enraged because Siri just doesn't get it.

Fooling fools is easy

though even fools can get angry.

ELIZA (Weizenbaum, 1964-66): artful deception

- use pattern matching and substitution to fake it (e.g. Prolog)

ELIZA effect: humans anthropomorphise computers

CAUTION: programs are quite different from humans

Siri rage (Urban dictionary):

When you get enraged because Siri just doesn't get it.

Fooling fools is easy

 though even fools can get angry.

Programs have improved hugely

ELIZA (Weizenbaum, 1964-66): artful deception

- use pattern matching and substitution to fake it (e.g. Prolog)

ELIZA effect: humans anthropomorphise computers

CAUTION: programs are quite different from humans

Siri rage (Urban dictionary):

When you get enraged because Siri just doesn't get it.

Fooling fools is easy

 though even fools can get angry.

Programs have improved hugely,

 humans largely not.

ChatGPT: Optimizing Language Models for Dialogue

We've trained a model called ChatGPT which interacts in a conversational way. The dialogue format makes it possible for ChatGPT to answer followup questions, admit its mistakes, challenge incorrect premises, and reject inappropriate requests.



ChatGPT: Optimizing Language Models for Dialogue

We've trained a model called ChatGPT which interacts in a conversational way. The dialogue format makes it possible for ChatGPT to answer followup questions, admit its mistakes, challenge incorrect premises, and reject inappropriate requests.

The New York Times | <https://www.nytimes.com/2022/12/05/technology/chatgpt-ai-twitter.html>

THE SHIFT

The Brilliance and Weirdness of ChatGPT

A new chatbot from OpenAI is inspiring awe, fear, stunts and attempts to circumvent its guardrails.

From OpenAI:

Limitations

- ChatGPT sometimes writes plausible-sounding but incorrect or nonsensical answers. Fixing this issue is challenging.

From OpenAI:

Limitations

- ChatGPT sometimes writes plausible-sounding but incorrect or nonsensical answers. Fixing this issue is challenging.

From the NY Times:

ChatGPT isn't perfect, by any means. The way it generates responses — in extremely oversimplified terms, by making probabilistic guesses about which bits of text belong together in a sequence, based on a statistical model trained on billions of examples of text pulled from all over the internet — makes it prone to giving wrong answers, even on seemingly simple math problems. (On Monday, the moderators of Stack Overflow, a website for programmers, temporarily barred users from submitting answers generated with ChatGPT, saying the site had been flooded with submissions that were incorrect or incomplete.)

From OpenAI:

Limitations

- ChatGPT sometimes writes plausible-sounding but incorrect or nonsensical answers. Fixing this issue is challenging.

From the NY Times:

ChatGPT isn't perfect, by any means. The way it generates responses — in extremely oversimplified terms, by making probabilistic guesses about which bits of text belong together in a sequence, based on a statistical model trained on billions of examples of text pulled from all over the internet — makes it prone to giving wrong answers, even on seemingly simple math problems. (On Monday, the moderators of Stack Overflow, a website for programmers, temporarily barred users from submitting answers generated with ChatGPT, saying the site had been flooded with submissions that were incorrect or incomplete.)

From The Guardian:

Its current knowledge base ends in 2021, rendering some queries and searches useless.

ChatGPT can also give entirely wrong answers and present misinformation as fact, writing “plausible-sounding but incorrect or nonsensical answers”, the company concedes.

OpenAI says that fixing this issue is difficult because there is no source of truth in the data they use to train the model and supervised training can also be misleading “because the ideal answer depends on what the model knows, rather than what the human demonstrator knows”.

Over capacity

**ChatGPT is at
capacity right now**

[Get notified when we're back](#)

Write an inspirational speech about the status of ChatGPT.

My fellow citizens, the time has come. The future is here, and it is called ChatGPT. This revolutionary AI chatbot website is here to change the world, and we must embrace it with all our strength and determination.

I know that many of you are eager to try ChatGPT for yourselves. And I assure you, we are doing everything in our power to make that happen. Our team is working tirelessly to accommodate the incredible demand for this groundbreaking technology.

But I want to remind you that greatness is never achieved without sacrifice. And the road to the future of AI will not be without its challenges. So please, be patient. Trust that the right time for you to try ChatGPT will come.

And when it does, let us seize this opportunity with all our might. Let us embrace the power of AI and use it to build a brighter, more prosperous future for ourselves and for generations to come. Thank you.



Do you have a theory of the world?



I am an AI trained to process and generate natural language text. I do not have beliefs, theories or desires. My function is to assist users in answering questions and completing tasks to the best of my ability using the knowledge and capabilities that have been programmed into me. Is there anything else I can help you with?



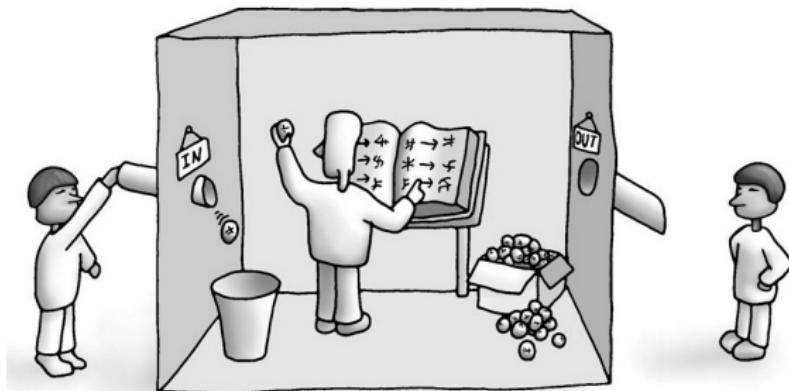
When you think, do you think based on a conception of the universe and how it works, or are you simply working via association?



As an AI, I don't think or experience in the way that humans do. I am a machine that processes input based on a set of rules and algorithms that have been programmed into me. These rules and algorithms allow me to process input, like text, and generate output, like a response to a question. I can understand and process natural language, but I don't have a personal experience of the universe, consciousness or any personal beliefs. My understanding of the world is based on the information and knowledge that has been programmed into me by humans.

My understanding is also limited by the parameters and information that I had at the time my knowledge cut off, which is 2021, I will not be able to make claims on new events or new discoveries that happened after that date.

Chinese room argument (Searle's thought experiment)

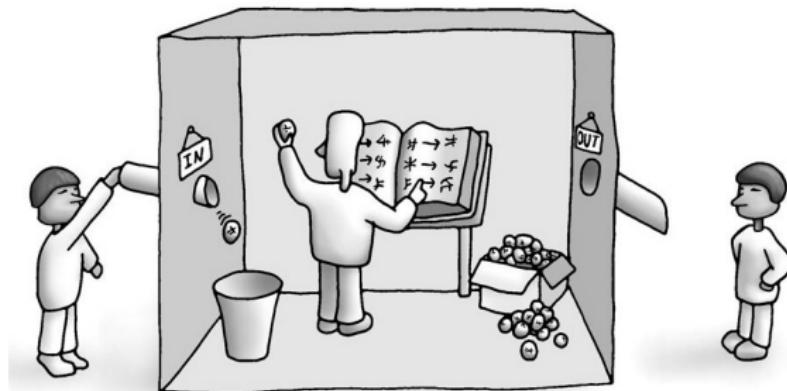


From <http://america.pink/images/9/6/3/2/5/4/en/2-chinese-room.jpg>

- a clerk can follow instructions for communicating in Chinese without understanding Chinese

Strong vs Weak AI

Chinese room argument (Searle's thought experiment)



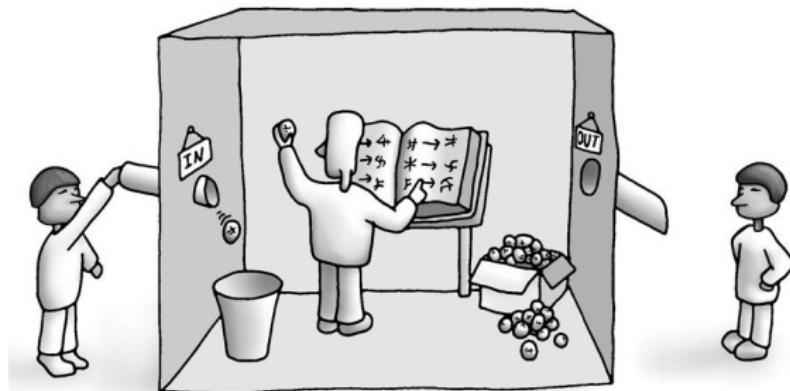
From <http://america.pink/images/9/6/3/2/5/4/en/2-chinese-room.jpg>

- a clerk can follow instructions for communicating in Chinese without understanding Chinese

mindless obedience \neq mind from biological processes

Strong vs Weak AI

Chinese room argument (Searle's thought experiment)

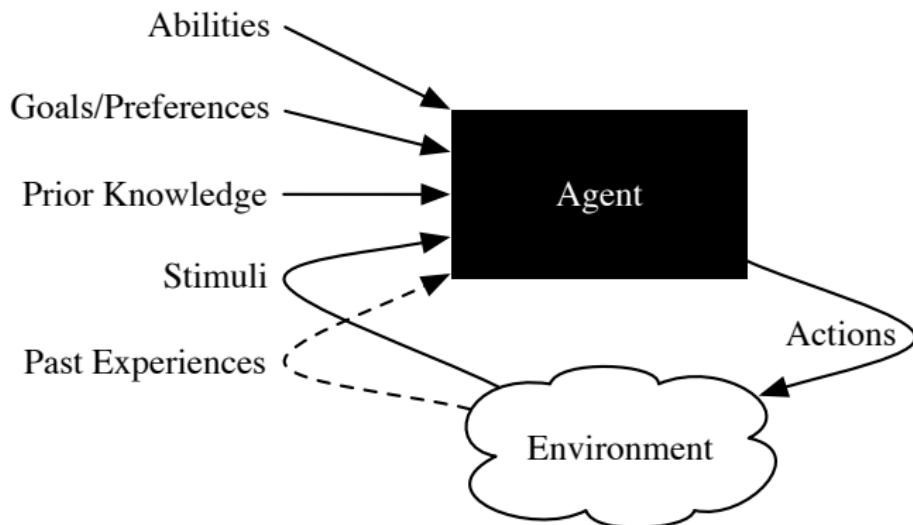


From <http://america.pink/images/9/6/3/2/5/4/en/2-chinese-room.jpg>

- a clerk can follow instructions for communicating in Chinese without understanding Chinese
 - mindless obedience \neq mind from biological processes

Just do it \rightsquigarrow Black Box, judged by its actions

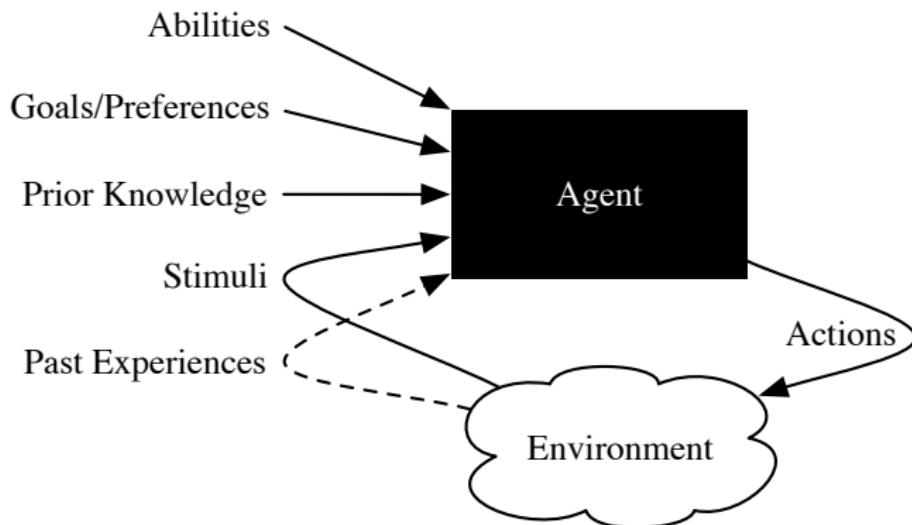
Locating intelligence (black box)



Poole & Mackworth

Intelligence: $(\text{abilities}, \text{goals}, \dots, \text{experience}) \mapsto \text{action}$

Locating intelligence (black box)



Poole & Mackworth

Intelligence: $(\text{abilities}, \text{goals}, \dots, \text{experience}) \mapsto \text{action}$

Turing test: what to say \rightsquigarrow what to do

Between agent and environment

agent	environment
program	data
Cognitive Revolution	BIG DATA
hard-wired	experienced
rationalist	empiricist
nativist	behaviorist
innate	tabula rasa
nature	nurture

Turing machine &
specialized automaton

Between agent and environment

agent	environment
program	data
Cognitive Revolution	BIG DATA
hard-wired	experienced
rationalist	empiricist
nativist	behaviorist
innate	tabula rasa
nature	nurture

Turing machine &
specialized automaton

Learning (from environment)
trial & error: “data as oil”

Between agent and environment

agent	environment
program	data
Cognitive Revolution	BIG DATA
hard-wired	experienced
rationalist	empiricist
nativist	behaviorist
innate	tabula rasa
nature	nurture

Turing machine &
specialized automaton

Learning (from environment)
trial & error: “data as oil”

Moving target: changing agent & environment

Intelligence comes in many levels

Intelligence comes in many levels from genius



Alan Turing, a British mathematician, proposed in 1950 that the test of machine intelligence would be an ability to conduct a conversation in an indistinguishably human way. Archivio GBB, via Alamy

Intelligence comes in many levels from genius to

Gobshite

Noun ~ Irish Slang
Mildly Offensive

- 1) One who engages in nonsensical or unwanted conversation.
- 2) A person of poor judgement or unpleasant character.
- 3) In plural form ~ A shower of gobshites = A group of idiots

Jay Ó Lonargáin

H. Frankfurt, *On Bullshit* (Wikipedia)

bullshit is speech intended to persuade without regard for truth. The liar cares about the truth and attempts to hide it; the bullshitter doesn't care if what they say is true or false, but cares only whether the listener is persuaded...

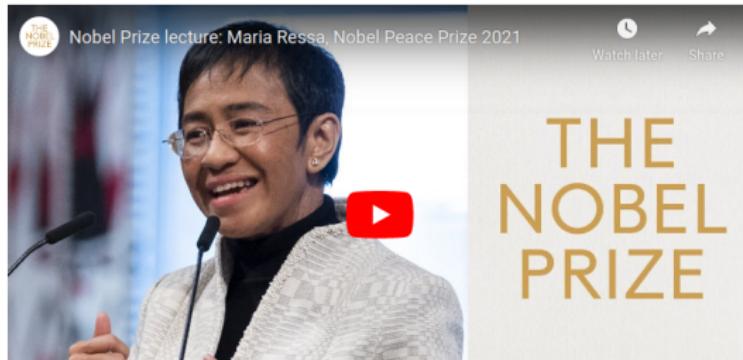
H. Frankfurt, *On Bullshit* (Wikipedia)

bullshit is speech intended to persuade without regard for truth. The liar cares about the truth and attempts to hide it; the bullshitter doesn't care if what they say is true or false, but cares only whether the listener is persuaded...

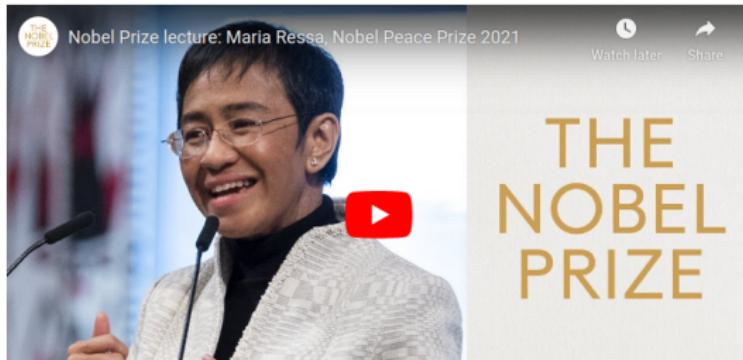
When asked why he decided to focus on bullshit, he explained:

Respect for the truth and a concern for the truth are among the foundations for civilization. I was for a long time disturbed by the lack of respect for the truth that I observed... bullshit is one of the deformities of these values.

His book *On Bullshit* addresses his concern and makes a distinction between "bullshitters" and liars. He concludes that bullshitters are more insidious: they are more of a threat against the truth than are liars.



technology, with its god-like power ... has allowed a virus of lies to infect each of us, pitting us against each other, bringing out our fears, anger and hate, and setting the stage for the rise of authoritarians and dictators around the world.

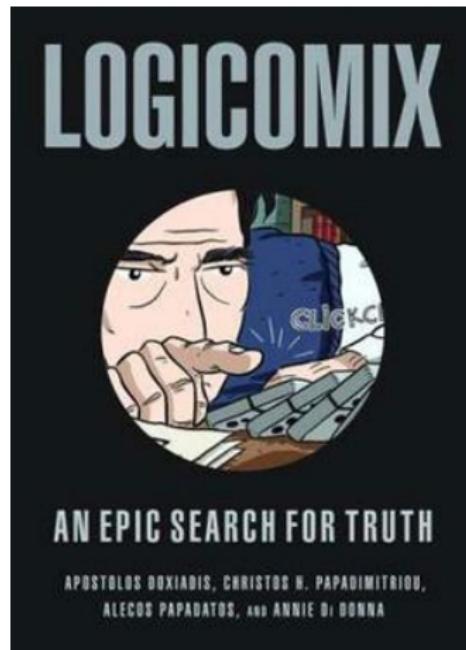


technology, with its god-like power ... has allowed a virus of lies to infect each of us, pitting us against each other, bringing out our fears, anger and hate, and setting the stage for the rise of authoritarians and dictators around the world.

Without facts, you can't have truth. Without truth, you can't have trust. Without trust, we have no shared reality, no democracy, and it becomes impossible to deal with our world's existential problems

Logic & the search for truth

www.logicomix.com



Logic & the search for truth

Challenges to

- truth

Liar's Paradox: 'I am lying'



Logic & the search for truth

Challenges to

- truth
Liar's Paradox: 'I am lying'
- sets (membership \in)
Russell set $R = \{x \mid \text{not } x \in x\}$

Logic & the search for truth

Challenges to

- truth

Liar's Paradox: 'I am lying'

- sets (membership \in)

Russell set $R = \{x \mid \text{not } x \in x\}$

$$R \in R \iff \text{not } R \in R$$

Logic & the search for truth

Challenges to

- truth

Liar's Paradox: 'I am lying'

- sets (membership \in)

Russell set $R = \{x \mid \text{not } x \in x\}$

- search (one by one)

Cantor: cannot count subsets of $\{0, 1, 2, \dots\}$

$s_1 = 0$	0	0	0	0	0	0	0	0	0	0	0	0	\dots
$s_2 = 1$	1	1	1	1	1	1	1	1	1	1	1	1	\dots
$s_3 = 0$	1	0	1	0	1	0	1	0	1	0	1	0	\dots
$s_4 = 1$	0	1	0	1	0	1	0	1	0	1	0	1	\dots
$s_5 = 1$	1	0	1	0	1	1	0	1	0	1	0	1	\dots
$s_6 = 0$	0	1	1	0	1	1	0	1	1	0	1	0	\dots
$s_7 = 1$	0	0	0	1	0	0	1	0	0	1	0	0	\dots
$s_8 = 0$	0	1	1	0	0	1	1	0	0	1	1	0	\dots
$s_9 = 1$	1	0	0	1	1	0	0	1	1	0	1	0	\dots
$s_{10} = 1$	1	0	1	1	1	0	0	1	0	1	0	1	\dots
$s_{11} = 1$	1	0	1	0	1	0	0	1	0	0	1	0	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

$s = 1 0 1 1 1 0 1 0 0 1 1 \dots$

Logic & the search for truth

Challenges to

- truth

Liar's Paradox: 'I am lying'

- sets (membership \in)

Russell set $R = \{x \mid \text{not } x \in x\}$

- search (one by one)

Cantor: cannot count subsets of $\{0, 1, 2, \dots\}$

- computability

Turing: Halting Problem

The Halting Problem

Given a program P and data D , return either 0 or 1 (as output), with 1 indicating that P halts on input D

$$\text{HP}(P, D) := \begin{cases} 1 & \text{if } P \text{ halts on } D \\ 0 & \text{otherwise} \end{cases}$$

The Halting Problem

Given a program P and data D , return either 0 or 1 (as output), with 1 indicating that P halts on input D

$$\text{HP}(P, D) := \begin{cases} 1 & \text{if } P \text{ halts on } D \\ 0 & \text{otherwise} \end{cases}$$

Theorem (Turing) No TM computes HP.

The proof is similar to the Liar's Paradox distributed as follows

- H: 'L speaks the truth'
- L: 'H lies'

with a spoiler L (exposing H as a fraud).

Proof of uncomputability

Given a TM P that takes two arguments, we show P does not compute HP by defining a TM \overline{P} such that

$$P(\overline{P}, \overline{P}) \neq \text{HP}(\overline{P}, \overline{P}).$$

Proof of uncomputability

Given a TM P that takes two arguments, we show P does not compute HP by defining a TM \bar{P} such that

$$P(\bar{P}, \bar{P}) \neq \text{HP}(\bar{P}, \bar{P}).$$

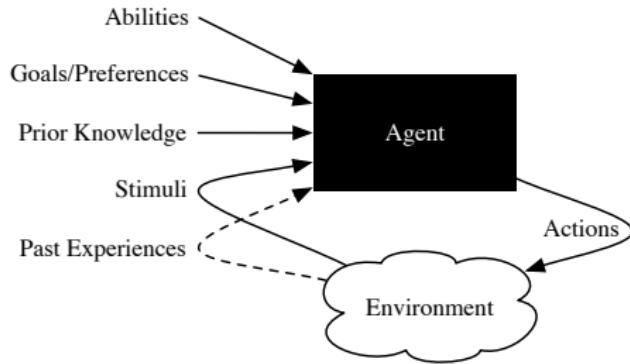
Let

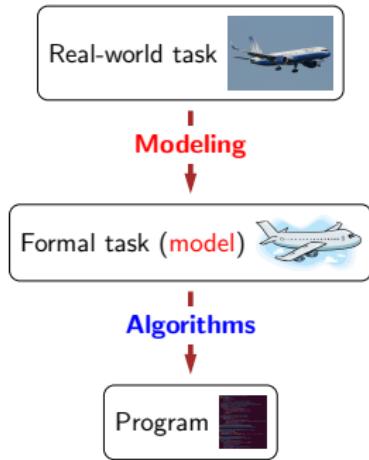
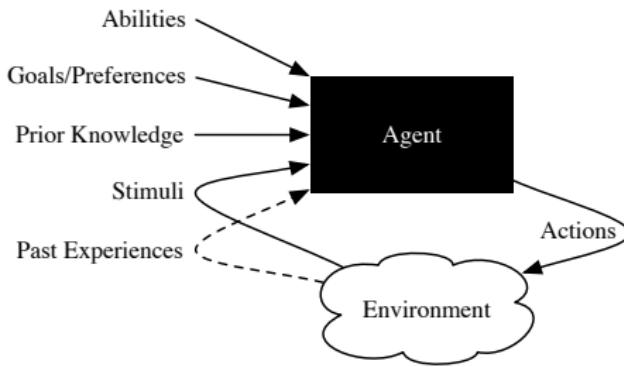
$$\bar{P}(D) \simeq \begin{cases} 1 & \text{if } P(D, D) = 0 \\ \text{loop} & \text{otherwise.} \end{cases}$$

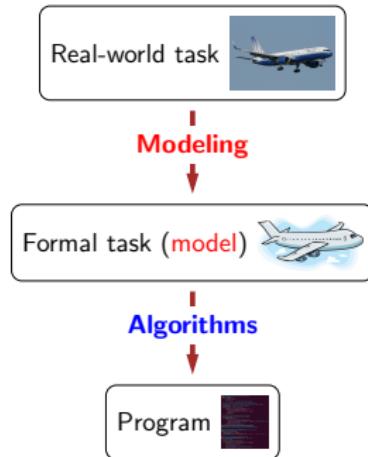
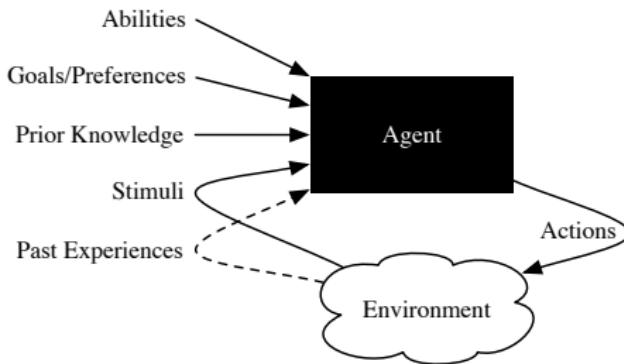
and notice

$$\begin{aligned} \text{HP}(\bar{P}, \bar{P}) &= \begin{cases} 1 & \text{if } \bar{P} \text{ halts on } \bar{P} \\ 0 & \text{otherwise} \end{cases} && (\text{def of HP}) \\ &= \begin{cases} 1 & \text{if } P(\bar{P}, \bar{P}) = 0 \\ 0 & \text{otherwise} \end{cases} && (\text{def of } \bar{P}) \end{aligned}$$

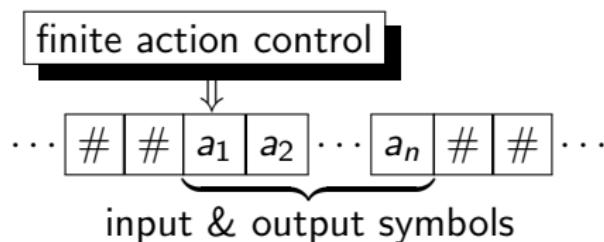
□



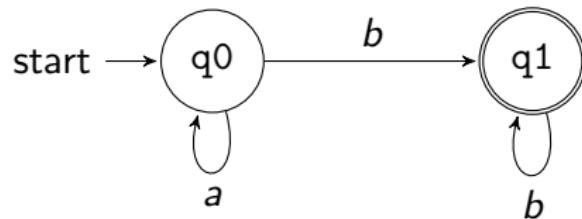




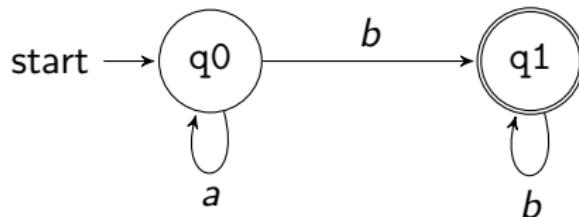
CHURCH-TURING THESIS: $\text{Program} \approx \text{Turing machine}$



Finite state machine (fsm)



Finite state machine (fsm)



A **fsm** M is a triple [Trans, Final, Q0] where

- Trans is a list of triples [Q, X, Qn] such that M may, at state Q seeing symbol X, change state to Qn
- Final is a list of M's final (i.e. accepting) states
- Q0 is M's initial state.

E.g. Trans = [[q0,a,q0], [q0,b,q1], [q1,b,q1]]

Final = [q1]

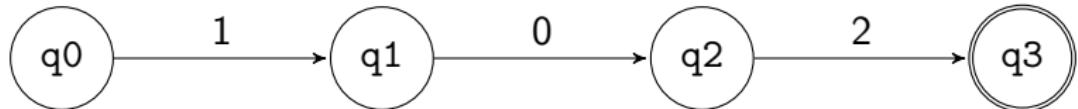
Q0 = q0

From strings to fsm's

Encode strings as lists; e.g. 102 as [1,0,2].

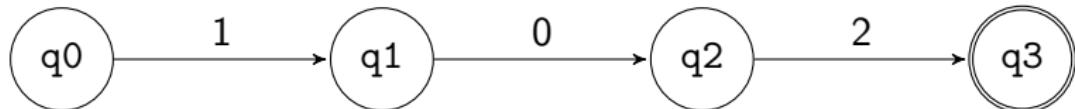
From strings to fsm's

Encode strings as lists; e.g. 102 as [1, 0, 2].



From strings to fsm's

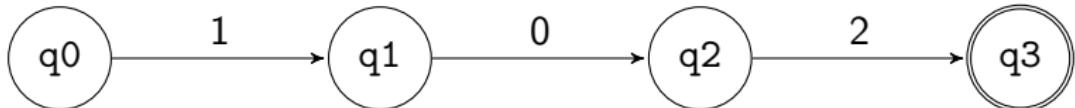
Encode strings as lists; e.g. 102 as [1,0,2].



```
% string2fsm(+String, ?TransitionSet, ?FinalStates)
string2fsm([], [], [q0]).  
string2fsm([H|T], Trans, [Last]) :-
```

From strings to fsm's

Encode strings as lists; e.g. 102 as [1,0,2].



```
% string2fsm(+String, ?TransitionSet, ?FinalStates)
string2fsm([], [], [q0]).  
string2fsm([H|T], Trans, [Last]) :-  
    mkTL(T, [H], [[q0, H, [H]]], Trans, Last).  
  
% mkTL(+More, +LastSoFar, +TransSoFar, ?Trans, ?Last)
mkTL([], L, Trans, Trans, L).  
mkTL([H|T], L, TransSoFar, Trans, Last) :-  
    mkTL(T, [H|L], [[L,H,[H|L]]|TransSoFar],  
          Trans, Last).
```

States as histories (in reverse)

More on *states-as-histories*

Encoding q0 as [] leads to the simplification

```
str2fsm(String, Trans, [Last]) :-  
    mkTL(String, [], [], Trans, Last).
```

More on *states-as-histories*

Encoding q0 as [] leads to the simplification

```
str2fsm(String, Trans, [Last]) :-  
    mkTL(String, [], [], Trans, Last).
```

States-as-histories works for finite languages,

More on *states-as-histories*

Encoding q0 as [] leads to the simplification

```
str2fsm(String, Trans, [Last]) :-  
    mkTL(String, [], [], Trans, Last).
```

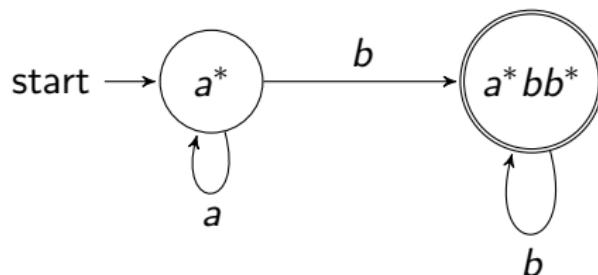
States-as-histories works for finite languages, but not say, a^*bb^*

More on *states-as-histories*

Encoding q_0 as $[]$ leads to the simplification

```
str2fsm(String, Trans, [Last]) :-  
    mkTL(String, [], [], Trans, Last).
```

States-as-histories works for finite languages, but not say, a^*bb^*



for *state-as-set-of-equivalent-histories*,
where equivalence has to do with acceptable **futures** ...

Exercise

Define a 4-ary predicate

```
accept(+Trans, +Final, +Q0, ?String)
```

that is true exactly when [Trans,Final,Q0] is a fsm that accepts String (encoded as a list).

Exercise

Define a 4-ary predicate

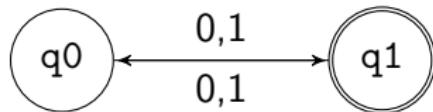
```
accept(+Trans,+Final,+Q0,?String)
```

that is true exactly when [Trans,Final,Q0] is a fsm that accepts String (encoded as a list).

That is, write a Prolog program to answer queries such as

```
?- accept([[q0,0,q1],[q0,1,q1],[q1,0,q0],[q1,1,q0]],  
          [q1], q0, [1,0,0]).
```

true



Exercise

Define a 4-ary predicate

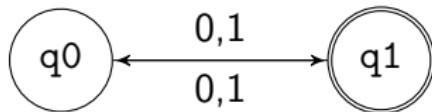
```
accept(+Trans,+Final,+Q0,?String)
```

that is true exactly when [Trans,Final,Q0] is a fsm that accepts String (encoded as a list).

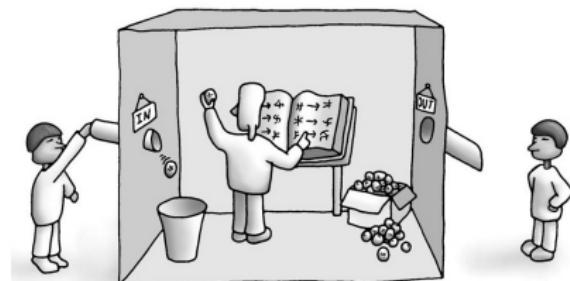
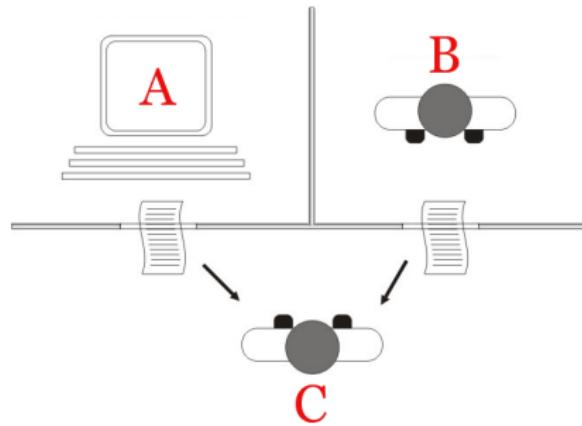
That is, write a Prolog program to answer queries such as

```
?- accept([[q0,0,q1],[q0,1,q1],[q1,0,q0],[q1,1,q0]],  
          [q1], q0, [1,0,0]).
```

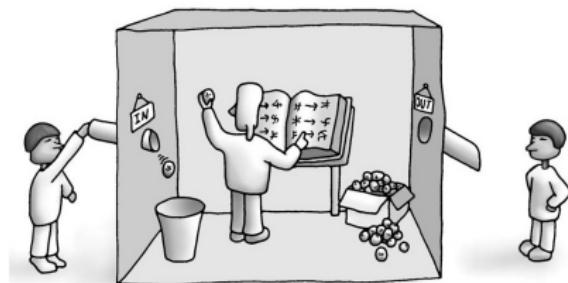
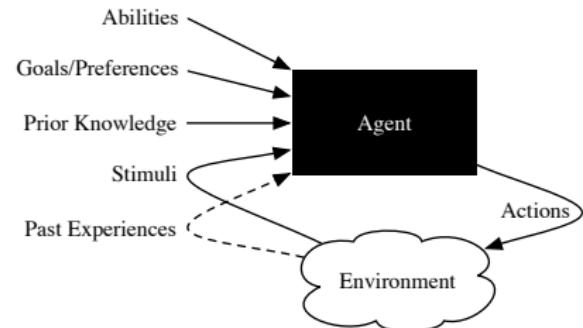
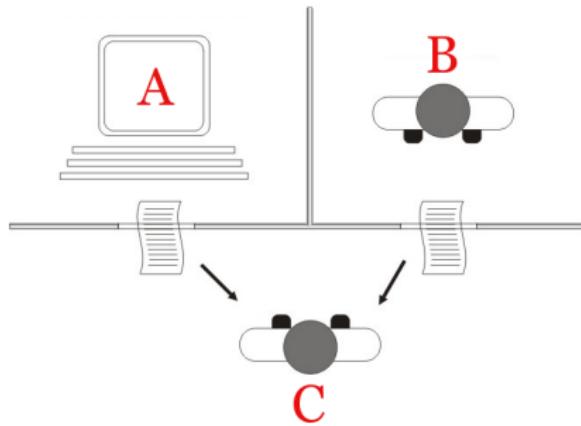
true



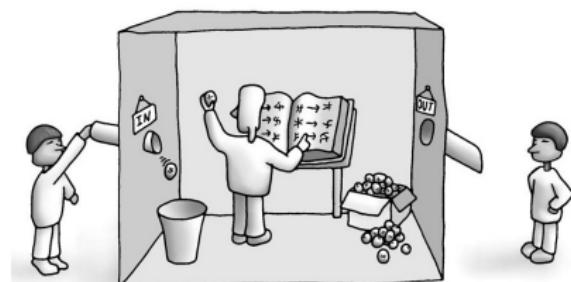
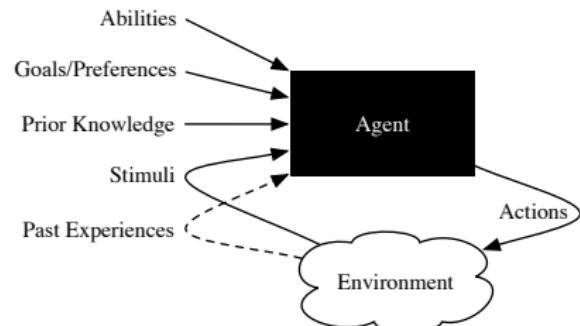
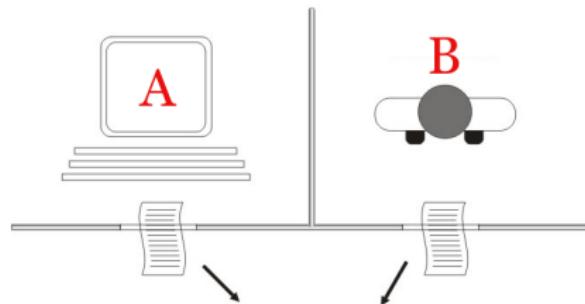
```
test(String) :- string2fsm(String, Trans, Final),  
               accept(Trans, Final, q0, String).
```



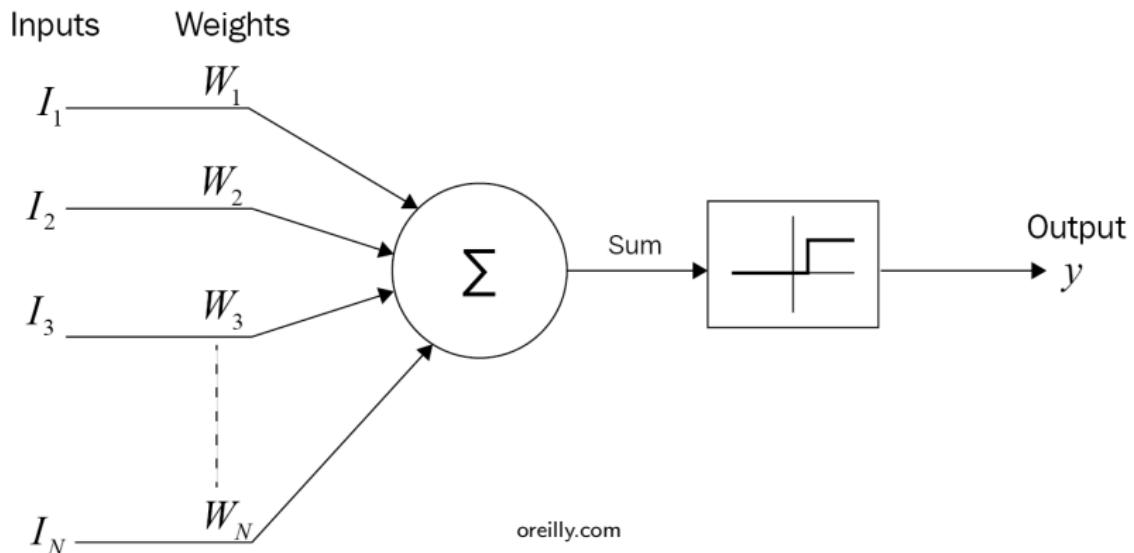
the black box



Peering inside the black box

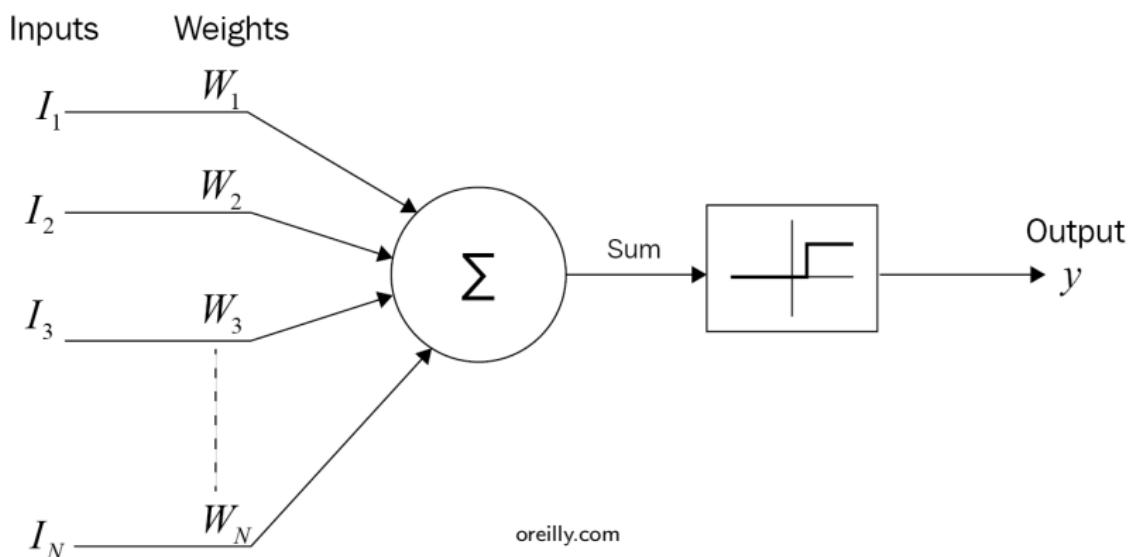


Agents from neurons (perceptrons)



Agents from neurons (perceptrons)

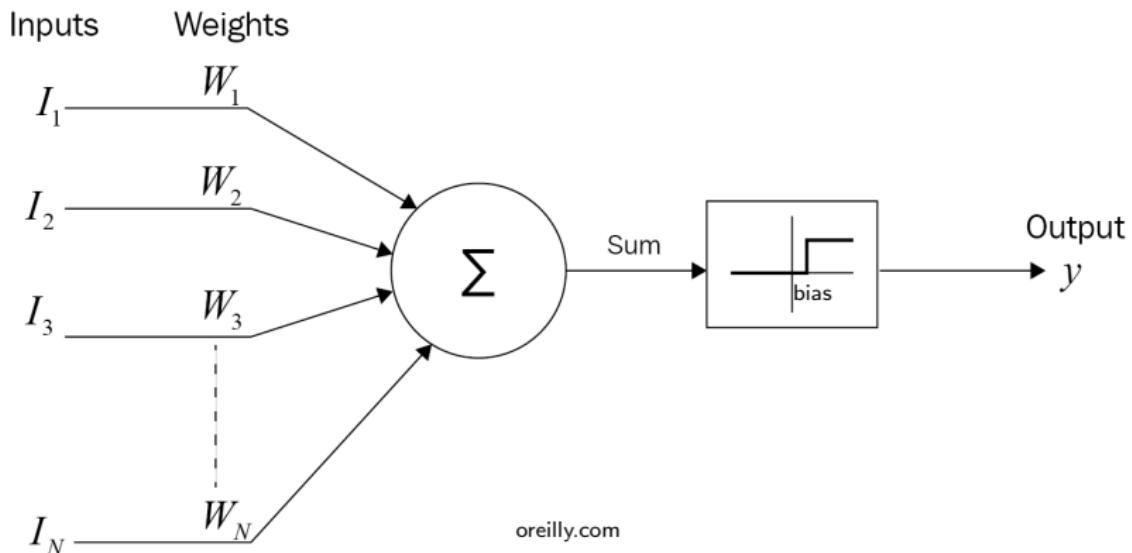
$$\text{Sum} = W_1I_1 + W_2I_2 + \cdots + W_NI_N$$



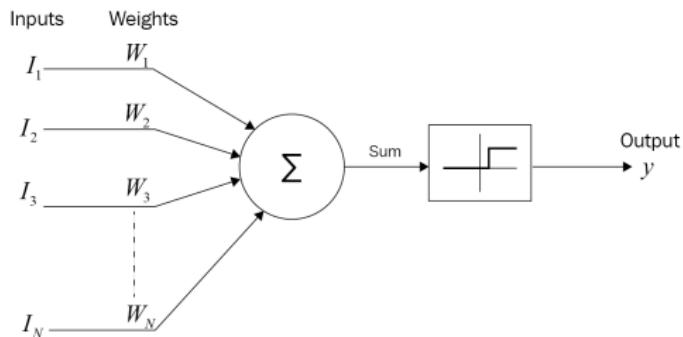
Agents from neurons (perceptrons)

$$\text{Sum} = W_1 I_1 + W_2 I_2 + \cdots + W_N I_N$$

$$y = \begin{cases} 1 & \text{if Sum} \geq \text{bias} \\ 0 & \text{if Sum} < \text{bias} \end{cases}$$

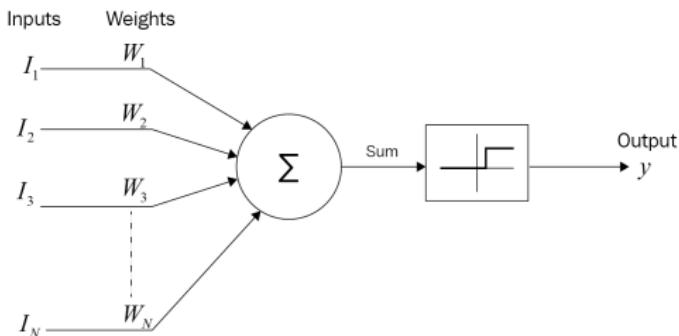


Finite state aggregations



Finite state aggregations

Kleene 1956



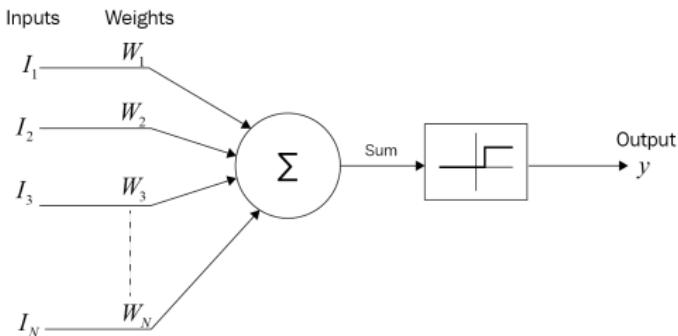
$$I_1, I_2 \dots I_N = q, a$$

$q \xrightarrow{a} y \iff$
output y on
input a at q

input a given by k *input cells* $N_1 \dots N_k$
state $q = (v_1 \dots v_m)$ records the
values v_i of m *inner cells* $M_1 \dots M_m$

Finite state aggregations

Kleene 1956



$$I_1, I_2 \dots I_N = q, a$$

$$q \xrightarrow{a} y \iff \text{output } y \text{ on input } a \text{ at } q$$

input a given by k *input cells* $N_1 \dots N_k$
state $q = (v_1 \dots v_m)$ records the
values v_i of m *inner cells* $M_1 \dots M_m$

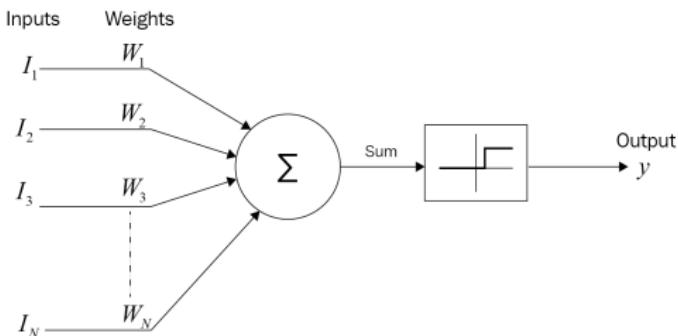
McCulloch & Pitts 1943: neural activity is *all-or-none*

Kleene 1956: input cells are binary, forming a string a of k bits
inner cell M_i can take one of s_i many values

$$q \xrightarrow{a} (y_1 \dots y_m) \iff q \xrightarrow{a}_i y_i \text{ for } 1 \leq i \leq m$$

Finite state aggregations

Kleene 1956



$$I_1, I_2 \dots I_N = q, a$$

$$q \xrightarrow{a} y \iff \text{output } y \text{ on input } a \text{ at } q$$

input a given by k *input cells* $N_1 \dots N_k$
state $q = (v_1 \dots v_m)$ records the
values v_i of m *inner cells* $M_1 \dots M_m$

McCulloch & Pitts 1943: neural activity is *all-or-none*

Kleene 1956: input cells are binary, forming a string a of k bits
inner cell M_i can take one of s_i many values

$$q \xrightarrow{a} (y_1 \dots y_m) \iff q \xrightarrow{a}_i y_i \text{ for } 1 \leq i \leq m$$

Rabin & Scott 1959: work with any finite sets A , Q and \rightarrow

Logical abstractions away from physics and biology

1976 Turing Award to Rabin & Scott for their joint paper

The internal workings of an automaton will not be analyzed too deeply. We are **not** concerned with **how** the machine is built but with **what** it can do. The definition of the internal structure must be general enough to cover all conceivable machines, but it need not involve itself with problems of circuitry.

Logical abstractions away from physics and biology

1976 Turing Award to Rabin & Scott for their joint paper

The internal workings of an automaton will not be analyzed too deeply. We are **not** concerned with **how** the machine is built but with **what** it can do. The definition of the internal structure must be general enough to cover all conceivable machines, but it need not involve itself with problems of circuitry. The simple method of obtaining generality without unnecessary detail is to use the concept of **internal states**. No matter how many wires or tubes or relays the machine contains, its operation is determined by stable states of the machine at discrete time intervals. An actual existing machine may have billions of such internal states, but the number is not important from the theoretical standpoint — only the fact that it is **finite**.

A language no fsm accepts

$\{\epsilon, ab, aabb, aaabbb, \dots\}$

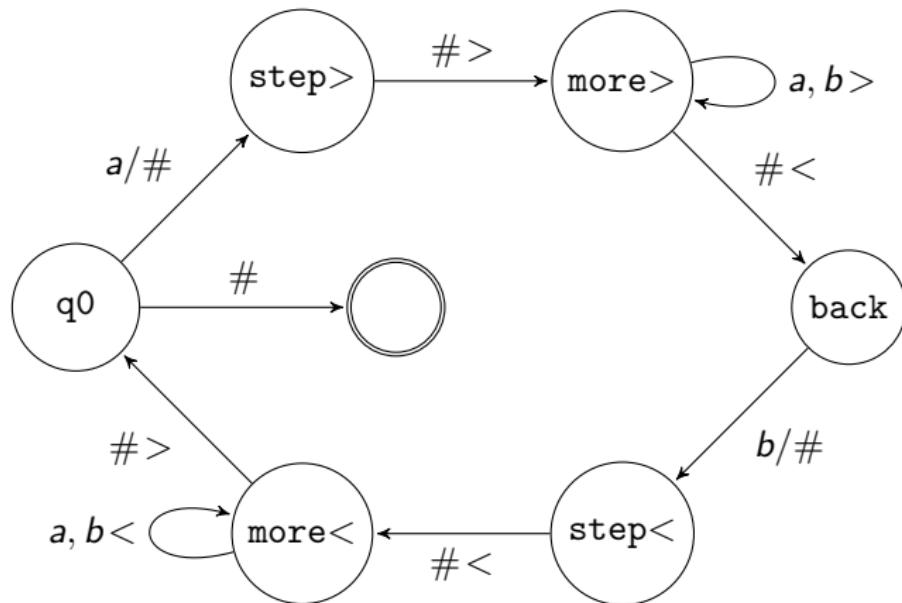
context-free grammar $S \longrightarrow \epsilon \mid aSb$

A language no fsm accepts

$$\{\epsilon, ab, aabb, aaabbb, \dots\}$$

context-free grammar $S \rightarrow \epsilon \mid aSb$

Add blank symbol $\#$ and actions write, move left

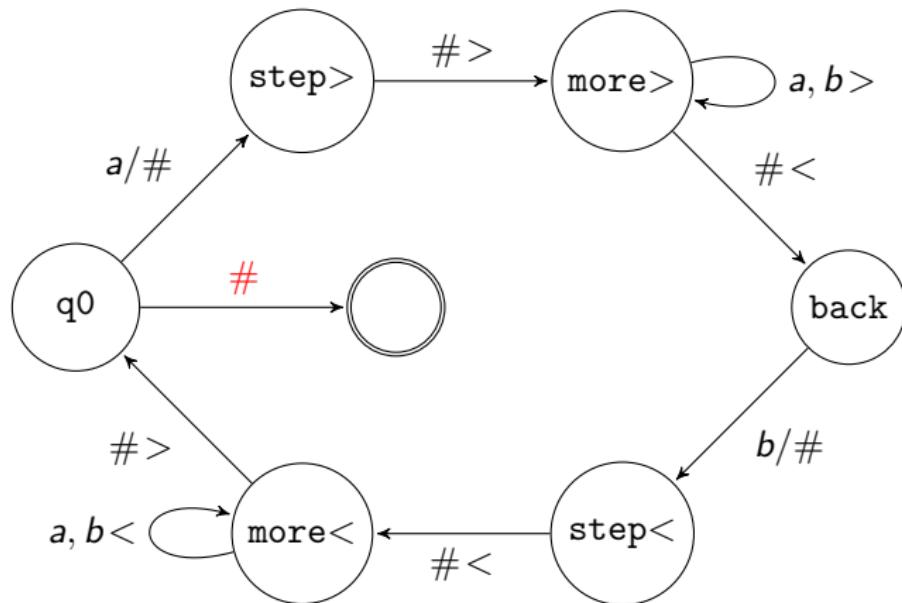


A language no fsm accepts

$$\{\epsilon, ab, aabb, aaabbb, \dots\}$$

context-free grammar $S \longrightarrow \epsilon \mid aSb$

Add blank symbol $\#$ and actions write, move left

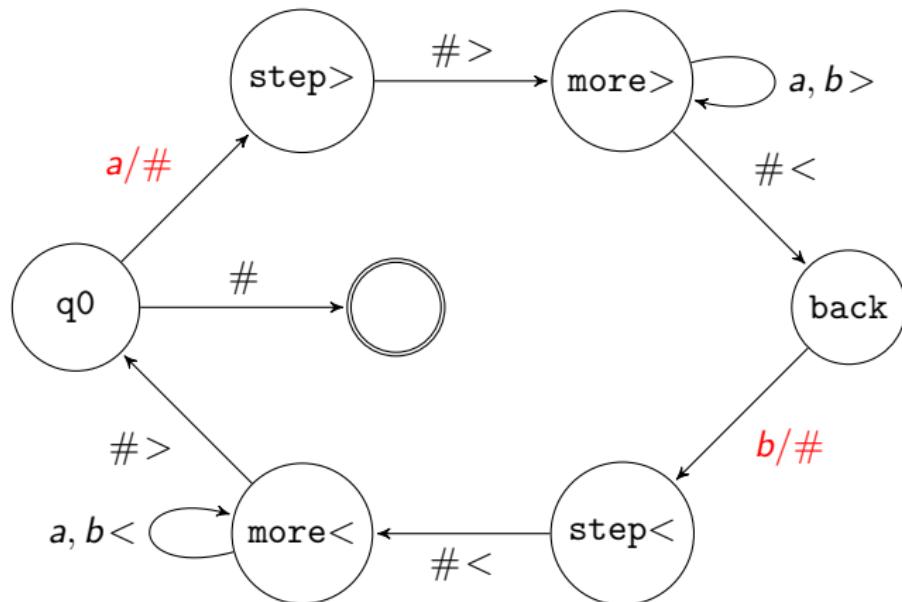


A language no fsm accepts

$$\{\epsilon, ab, aabb, aaabbb, \dots\}$$

context-free grammar $S \rightarrow \epsilon \mid aSb$

Add blank symbol $\#$ and actions **write**, move left

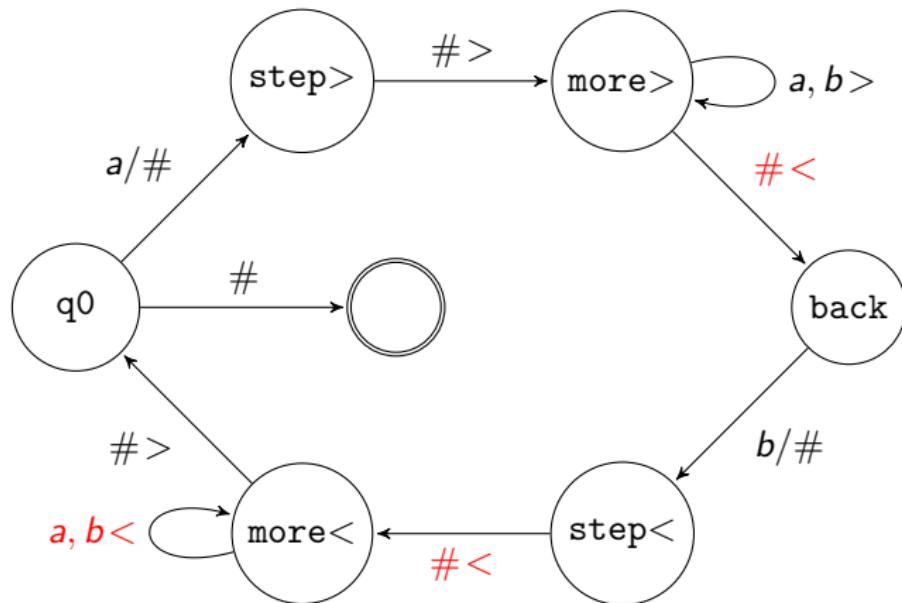


A language no fsm accepts

$$\{\epsilon, ab, aabb, aaabbb, \dots\}$$

context-free grammar $S \rightarrow \epsilon \mid aSb$

Add blank symbol $\#$ and actions write, move **left**



$Tm = fsm + MLeft/Write/Halt$

A **Turing machine** (Tm) M is a 5-tuple

[MRight, MLeft, Write, Halt, Q0]

where

- MRight is a list of triples $[Q, X, Qn]$ such that at state Q and seeing symbol X , M may move right, and change state to Qn
- $Q0$ is M 's initial state

$Tm = fsm + MLeft/Write/Halt$

A **Turing machine** (Tm) M is a 5-tuple

[MRight, MLeft, Write, Halt, Q0]

where

- MRight is a list of triples $[Q, X, Qn]$ such that at state Q and seeing symbol X , M may move right, and change state to Qn
- $Q0$ is M 's initial state
- MLeft is a list of triples $[Q, X, Qn]$ such that at Q and seeing X , M may move left, and change state to Qn

$Tm = fsm + MLeft/Write/Halt$

A **Turing machine** (Tm) M is a 5-tuple

[MRight, MLeft, Write, Halt, Q0]

where

- MRight is a list of triples $[Q, X, Q_n]$ such that at state Q and seeing symbol X , M may move right, and change state to Q_n
- Q_0 is M 's initial state
- MLeft is a list of triples $[Q, X, Q_n]$ such that at Q and seeing X , M may move left, and change state to Q_n
- Write is a list of 4-tuples $[Q, X, Y, Q_n]$ such that at Q and X , M may write Y , and change state to Q_n without moving

$Tm = fsm + MLeft/Write/Halt$

A **Turing machine** (Tm) M is a 5-tuple

[MRight, MLeft, Write, Halt, Q0]

where

- MRight is a list of triples $[Q, X, Q_n]$ such that at state Q and seeing symbol X , M may move right, and change state to Q_n
- Q_0 is M 's initial state
- MLeft is a list of triples $[Q, X, Q_n]$ such that at Q and seeing X , M may move left, and change state to Q_n
- Write is a list of 4-tuples $[Q, X, Y, Q_n]$ such that at Q and X , M may write Y , and change state to Q_n without moving
- Halt is a list of pairs $[Q, X]$ such that at Q and X , M may halt.

$Tm = fsm + MLeft/Write/Halt$

A **Turing machine** (Tm) M is a 5-tuple

[MRight, MLeft, Write, Halt, Q0]

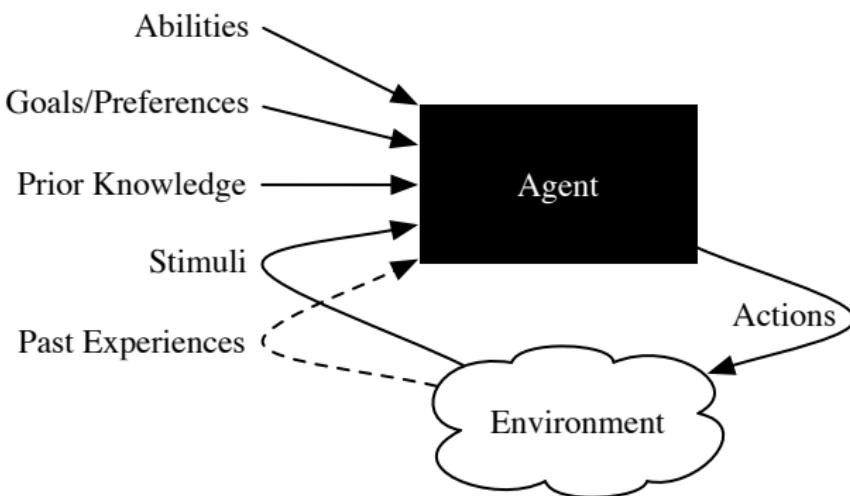
where

- MRight is a list of triples $[Q, X, Qn]$ such that at state Q and seeing symbol X , M may move right, and change state to Qn
- $Q0$ is M 's initial state
- MLeft is a list of triples $[Q, X, Qn]$ such that at Q and seeing X , M may move left, and change state to Qn
- Write is a list of 4-tuples $[Q, X, Y, Qn]$ such that at Q and X , M may write Y , and change state to Qn without moving
- Halt is a list of pairs $[Q, X]$ such that at Q and X , M may halt.

N.B. A fsm is a Tm where $MLeft = [] = Write$, and for every pair $[Q, X]$ in $Halt$, the symbol X is $\#$ (blank).

Fsm exercise: solution

```
accept(_,Final,Q,[]) :-  
accept(Trans,Final,Q,[H|T]) :-
```



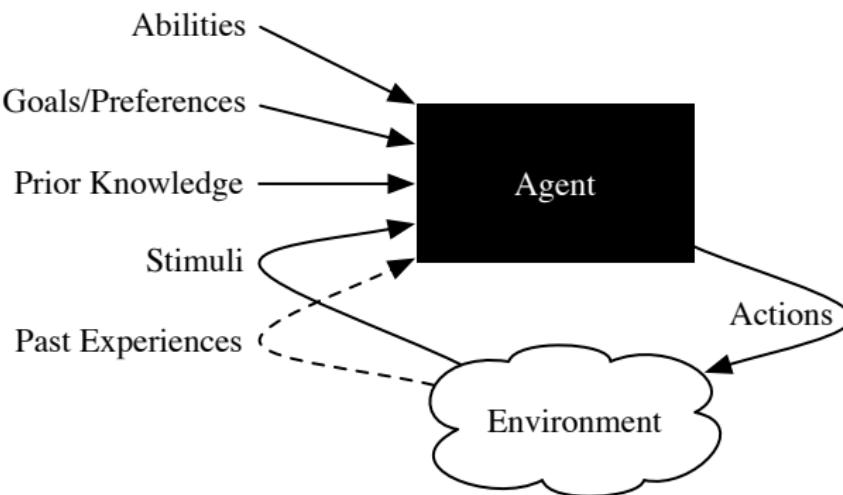
Fsm exercise: solution

```
accept(_,Final,Q,[]) :- member(Q,Final).
```

```
accept(Trans,Final,Q,[H|T]) :-
```

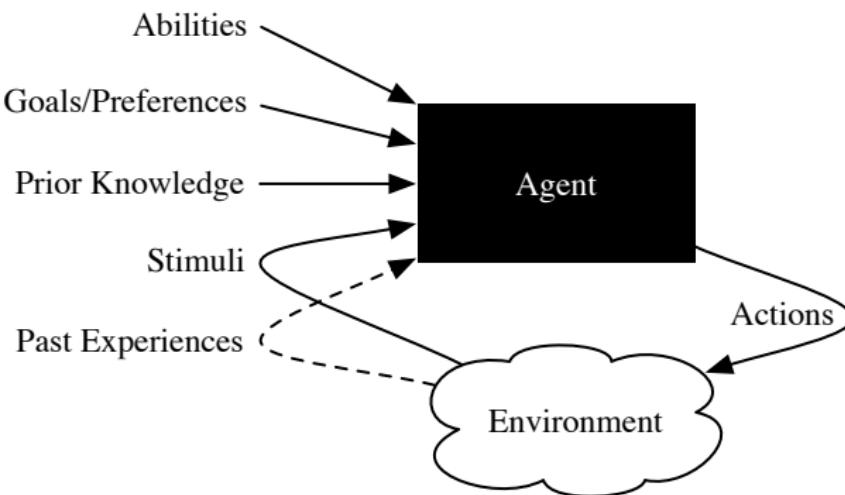
```
    member(X,[X|_]).
```

```
    member(X,[_|L]):- member(X,L).
```



Fsm exercise: solution

```
accept(_,Final,Q,[]) :- member(Q,Final).  
  
accept(Trans,Final,Q,[H|T]) :-  
    member([Q,H,Qn],Trans),  
    accept(Trans,Final,Qn,T).  
  
member(X,[X|_]).  
member(X,[_|L]):- member(X,L).
```



Search (in Prolog)

Given goal, arc

```
search(Node) :- goal(Node).
```

```
search(Node) :- arc(Node,Next), search(Next).
```

Search (in Prolog)

Given goal, arc

```
search(Node) :- goal(Node).  
search(Node) :- arc(Node,Next), search(Next).
```

Example: accept(Trans,Final,Q0,String)
Node as [Q,UnseenString]

Search (in Prolog)

Given goal, arc

```
search(Node) :- goal(Node).  
search(Node) :- arc(Node,Next), search(Next).
```

Example: accept(Trans,Final,Q0,String)
Node as [Q,UnseenString]

```
goal(Q,[],Final) :- member(Q,Final).
```

Search (in Prolog)

Given goal, arc

```
search(Node) :- goal(Node).
```

```
search(Node) :- arc(Node,Next), search(Next).
```

Example: accept(Trans,Final,Q0,String)

Node as [Q,UnseenString]

```
goal(Q, [], Final) :- member(Q, Final).
```

```
arc([Q, [H|T]], [Qn, T], Trans) :-  
    member([Q, H, Qn], Trans).
```

Search (in Prolog)

Given goal, arc

```
search(Node) :- goal(Node).
```

```
search(Node) :- arc(Node,Next), search(Next).
```

Example: accept(Trans,Final,Q0,String)

Node as [Q,UnseenString]

```
goal(Q, [], Final) :- member(Q, Final).
```

```
arc([Q, [H|T]], [Qn, T], Trans) :-  
    member([Q, H, Qn], Trans).
```

```
search(Q, S, F, _) :- goal(Q, S, F).
```

```
search(Q, S, F, T) :- arc([Q, S], [Qn, Sn], T),  
    search(Qn, Sn, F, T).
```

Search (in Prolog)

Given goal, arc

```
search(Node) :- goal(Node).
```

```
search(Node) :- arc(Node,Next), search(Next).
```

Example: accept(Trans,Final,Q0,String)

Node as [Q,UnseenString]

```
goal(Q,[],Final) :- member(Q,Final).
```

```
arc([Q,[H|T]], [Qn,T], Trans) :-  
    member([Q,H,Qn], Trans).
```

```
search(Q,S,F,_) :- goal(Q,S,F).
```

```
search(Q,S,F,T) :- arc([Q,S], [Qn,Sn], T),  
    search(Qn,Sn,F,T).
```

```
accept(T,F,Q,S) :- search(Q,S,F,T).
```

Prolog as search

i :- p,q.

i :- r.

p.

r.

| ?- i.

Prolog as search

i :- p,q.

[i]

i :- r.

p.

r.

| ?- i.

StartNode = [i]

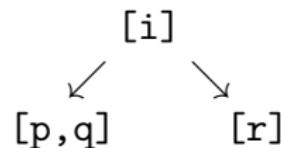
Prolog as search

i :- p,q.

i :- r.

p.

r.



| ?- i.

StartNode = [i]

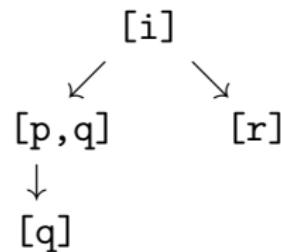
Prolog as search

i :- p,q.

i :- r.

p.

r.



| ?- i.

StartNode = [i]

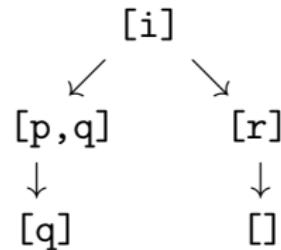
Prolog as search

i :- p,q.

i :- r.

p.

r.



| ?- i.

StartNode = [i]

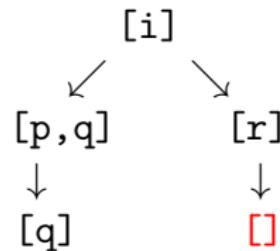
Prolog as search

i :- p,q.

i :- r.

p.

r.



| ?- i.

StartNode = [i]

yes

goal([]).

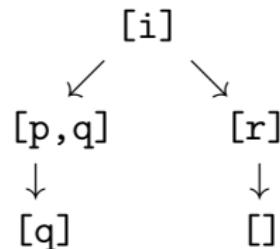
Prolog as search

i :- p,q.

i :- r.

p.

r.



| ?- i.

StartNode = [i]

yes

goal([]).

prove(Node) :- goal(Node) .

prove(Node) :- arc(Node,Next), prove(Next) .

KB and arc

i :- p,q.

i :- r.

p.

r.

KB and arc

i :- p,q.

[i,p,q]

i :- r.

[i,r]

p.

[p]

r.

[r]

KB and arc

i :- p,q.

[i,p,q]

i :- r.

[i,r]

p.

[p]

r.

[r]

KB = [[i,p,q],[i,r],[p],[r]]

KB and arc

i :- p,q.

[i,p,q]

i :- r.

[i,r]

p.

[p]

r.

[r]

KB = [[i,p,q],[i,r],[p],[r]]

arc(Node1,Node2,KB) :- ??

KB and arc

i :- p,q. [i,p,q]

i :- r. [i,r]

p. [p]

r. [r]

KB = [[i,p,q],[i,r],[p],[r]]

arc([H|T],N,KB) :- member([H|B],KB), append(B,T,N).

KB and arc

i :- p,q.

[i,p,q]

i :- r.

[i,r]

p.

[p]

r.

[r]

KB = [[i,p,q],[i,r],[p],[r]]

arc([H|T],N,KB) :- member([H|B],KB), append(B,T,N).

prove(Node,KB) :- goal(Node) ;
arc(Node,Next,KB), prove(Next,KB).

Non-termination

i :- p,q. [i]

i :- r.

p :- i.

r.

| ?- i.

prove([] ,_).

prove([H|T] ,KB) :- member([H|B] ,KB), append(B,T,Next) ,
 prove(Next,KB) .

Non-termination

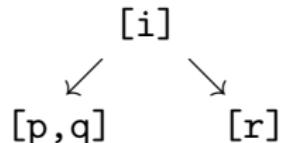
i :- p,q.

i :- r.

p :- i.

r.

| ?- i.



prove([] ,_).

prove([H|T] ,KB) :- member([H|B] ,KB), append(B,T,Next) ,
prove(Next,KB) .

Non-termination

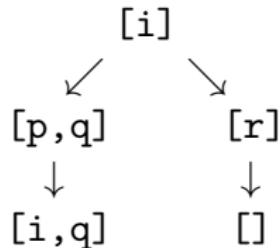
i :- p,q.

i :- r.

p :- i.

r.

| ?- i.



prove([] ,_).

prove([H|T] ,KB) :- member([H|B] ,KB) , append(B,T,Next) ,
prove(Next,KB) .

Non-termination

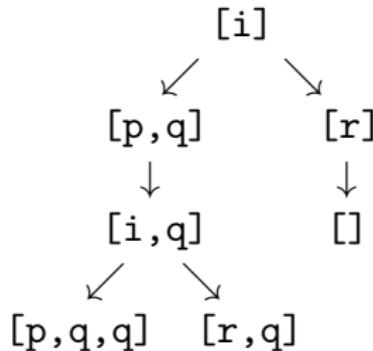
i :- p,q.

i :- r.

p :- i.

r.

| ?- i.



```
prove([] ,_).
```

```
prove([H|T] ,KB) :- member([H|B] ,KB) , append(B,T,Next) ,  
prove(Next ,KB) .
```

Non-termination

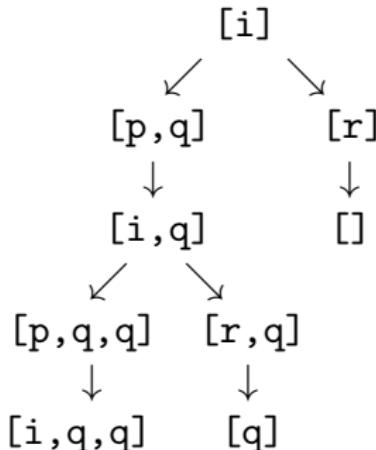
i :- p,q.

i :- r.

p :- i.

r.

| ?- i.



prove([] ,_).

prove([H|T] ,KB) :- member([H|B] ,KB) , append(B,T,Next) ,
prove(Next,KB) .

Non-termination

i :- p,q.

i :- r.

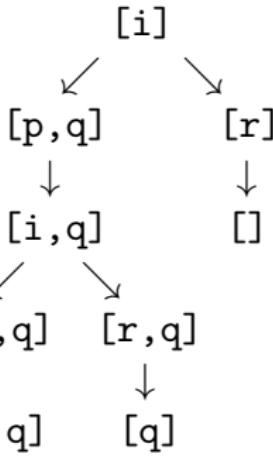
p :- i.

r.

| ?- i.

prove([],_).

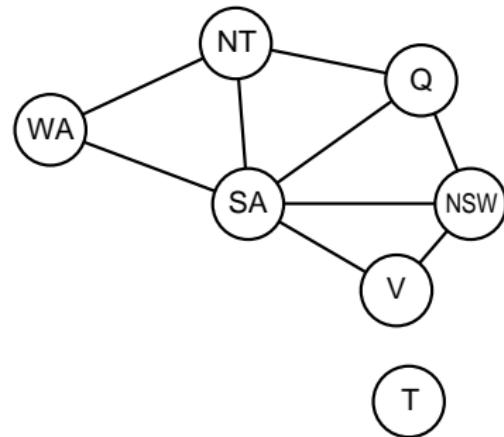
prove([H|T],KB) :- member([H|B],KB), append(B,T,Next),
prove(Next,KB).



Graph modeling

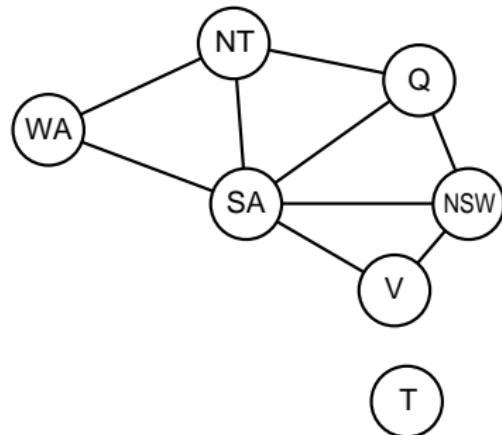


Graph modeling



Russell & Norvig

Graph modeling



Russell & Norvig

```
arc(wa,nt).      arc(nt,q).      arc(q,nsw).  
arc(wa,sa).      arc(nt,sa).     arc(sa,q).  
arc(sa,nsw).     arc(sa,v).      arc(v,nsw).  
  
arc2(X,Y) :- arc(X,Y) ; arc(Y,X).
```

Non-termination (due to poor choices)

i :- p,q. [i]

i :- r.

p :- i.

r.

| ?- i.

prove([] ,_).

prove([H|T] ,KB) :- member([H|B] ,KB) , append(B,T,Next) ,
 prove(Next,KB) .

| ?- prove([i], [[i,p,q],[i,r],[p,i],[r]]).

Non-termination (due to poor choices)

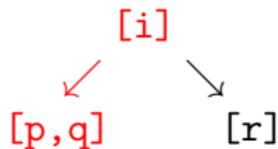
i :- p,q.

i :- r.

p :- i.

r.

| ?- i.



prove([] ,_).

prove([H|T] ,KB) :- member([H|B] ,KB) , append(B,T,Next) ,
 prove(Next,KB) .

| ?- prove([i],[[i,p,q],[i,r],[p,i],[r]]).

Non-termination (due to poor choices)

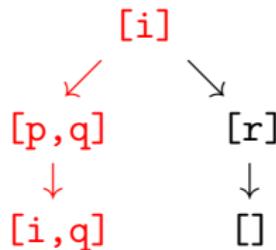
i :- p,q.

i :- r.

p :- i.

r.

| ?- i.



prove([] ,_).

prove([H|T] ,KB) :- member([H|B] ,KB) , append(B,T,Next) ,
 prove(Next,KB) .

| ?- prove([i],[[i,p,q],[i,r],[p,i],[r]]).

Non-termination (due to poor choices)

i :- p,q.

i :- r.

p :- i.

r.

| ?- i.

[i]

[p,q]

[i,q]

[p,q,q]

↖

[r]

[]

[r,q]

prove([] ,_).

prove([H|T] ,KB) :- member([H|B] ,KB) , append(B,T,Next) ,
 prove(Next,KB) .

| ?- prove([i],[[i,p,q],[i,r],[p,i],[r]]).

Non-termination (due to poor choices)

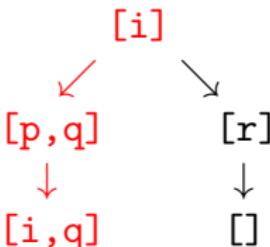
i :- p,q.

i :- r.

p :- i.

r.

| ?- i.



prove([], _).

prove([H|T], KB) :- member([H|B], KB), append(B, T, Next),
prove(Next, KB).

| ?- prove([i], [[i,p,q],[i,r],[p,i],[r]]).

Non-termination (due to poor choices)

i :- p,q.

i :- r.

p :- i.

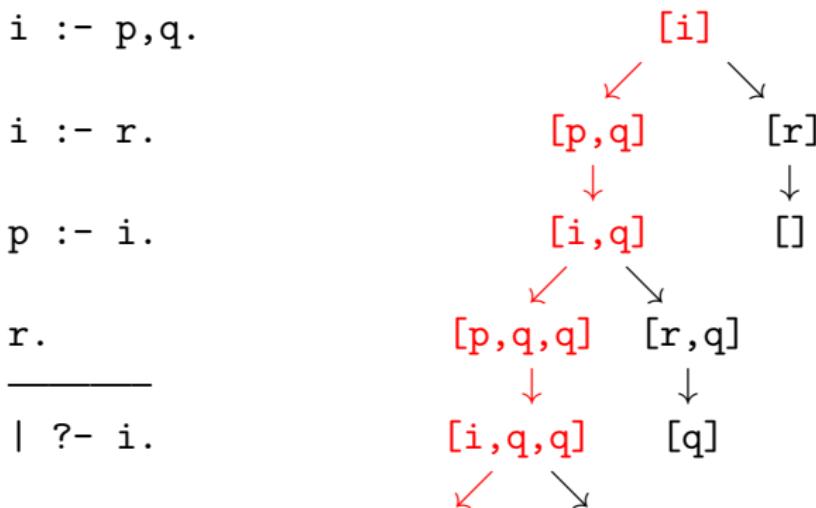
r.

| ?- i.

prove([],_).

prove([H|T],KB) :- member([H|B],KB), append(B,T,Next),
prove(Next,KB).

| ?- prove([i], [[i,p,q],[i,r],[p,i],[r]]).



Determinization (eliminate choice)

A fsm [Trans, Final, Q0] such that

for all $[Q, X, Q_n]$ *and* $[Q, X, Q_n']$ *in Trans,* $Q_n = Q_n'$

is a *deterministic finite automaton* (DFA).

Determinization (eliminate choice)

A fsm [Trans, Final, Q0] such that

for all $[Q, X, Q_n]$ *and* $[Q, X, Q_n']$ *in Trans,* $Q_n = Q_n'$

is a *deterministic finite automaton* (DFA).

Fact. Every fsm has a DFA accepting the same language.

Determinization (eliminate choice)

A fsm [Trans, Final, Q0] such that

for all $[Q, X, Qn]$ and $[Q, X, Qn']$ in Trans, $Qn = Qn'$

is a *deterministic finite automaton* (DFA).

Fact. Every fsm has a DFA accepting the same language.

Proof: Subset (powerset) construction

Apply to arc,goal, contra Trans,Final:

```
arcD(NodeList,NextList) :-  
    setof(Next, arcLN(NodeList,Next), NextList).  
  
arcLN(NodeList,Next) :- member(Node,NodeList),  
    arc(Node,Next).
```

Determinization (eliminate choice)

A fsm [Trans, Final, Q0] such that

for all $[Q, X, Qn]$ and $[Q, X, Qn']$ in Trans, $Qn = Qn'$

is a *deterministic finite automaton* (DFA).

Fact. Every fsm has a DFA accepting the same language.

Proof: Subset (powerset) construction

Apply to arc,goal, contra Trans,Final:

```
arcD(NodeList,NextList) :-  
    setof(Next, arcLN(NodeList,Next), NextList).  
  
arcLN(NodeList,Next) :- member(Node,NodeList),  
    arc(Node,Next).  
  
goalD(NodeList):-member(Node,NodeList),goal(Node).
```

Determinization (eliminate choice)

A fsm [Trans, Final, Q0] such that

for all $[Q, X, Qn]$ and $[Q, X, Qn']$ in Trans, $Qn = Qn'$

is a *deterministic finite automaton* (DFA).

Fact. Every fsm has a DFA accepting the same language.

Proof: Subset (powerset) construction

Apply to arc,goal, contra Trans,Final:

```
arcD(NodeList,NextList) :-  
    setof(Next, arcLN(NodeList,Next), NextList).  
  
arcLN(NodeList,Next) :- member(Node,NodeList),  
    arc(Node,Next).  
  
goalD(NodeList):-member(Node,NodeList),goal(Node).  
  
searchD(NL) :- goalD(NL);  
    (arcD(NL,NL2), searchD(NL2)).
```

Determinization (eliminate choice)

A fsm [Trans, Final, Q0] such that

for all $[Q, X, Qn]$ and $[Q, X, Qn']$ in Trans, $Qn = Qn'$

is a *deterministic finite automaton* (DFA).

Fact. Every fsm has a DFA accepting the same language.

Proof: Subset (powerset) construction

Apply to arc,goal, contra Trans,Final:

```
arcD(NodeList,NextList) :-  
    setof(Next, arcLN(NodeList,Next), NextList).
```

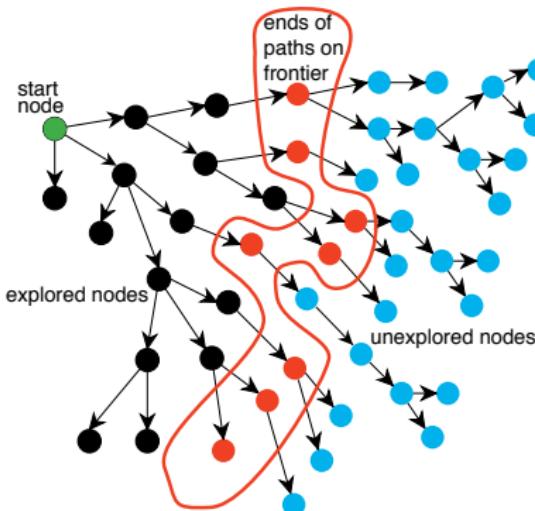
```
arcLN(NodeList,Next) :- member(Node,NodeList),  
    arc(Node,Next).
```

```
goalD(NodeList):-member(Node,NodeList),goal(Node).
```

```
searchD(NL) :- goalD(NL);  
    (arcD(NL,NL2), searchD(NL2)).
```

```
search(Node) :- searchD([Node]).
```

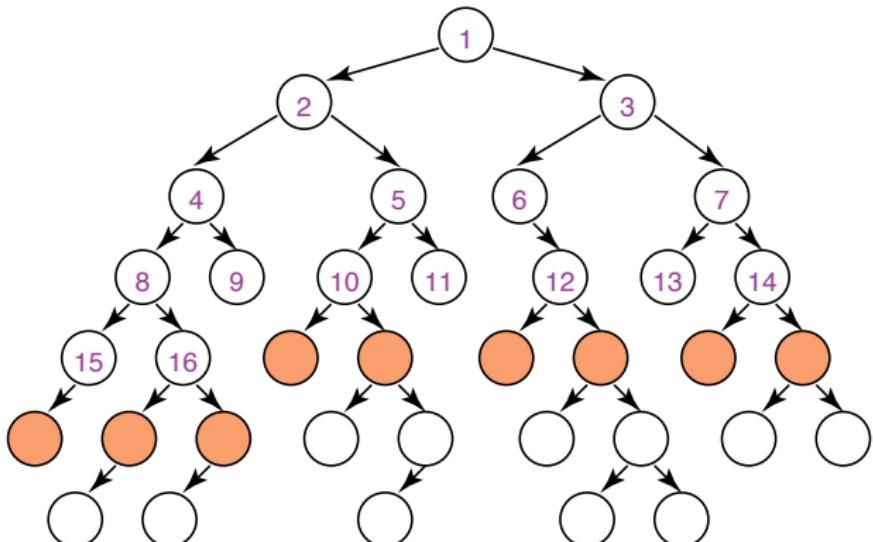
Frontier search



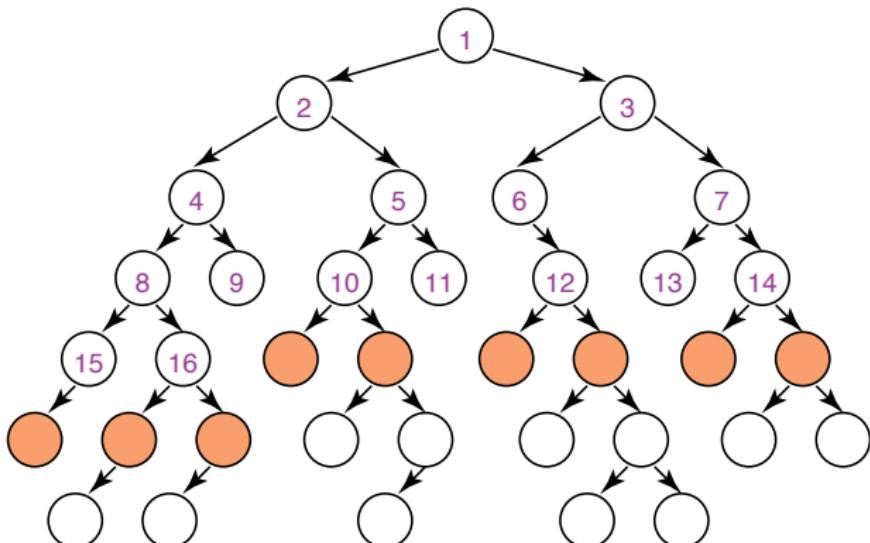
Poole & Mackworth

```
search(Node) :- frontierSearch([Node]).  
  
frontierSearch([Node|_]) :- goal(Node).  
  
frontierSearch([Node|Rest]) :-  
    findall(Next, arc(Node, Next), Children),  
    add2frontier(Children, Rest, NewFrontier),  
    frontierSearch(NewFrontier).
```

Breadth-first: queue (FIFO)



Breadth-first: queue (FIFO)

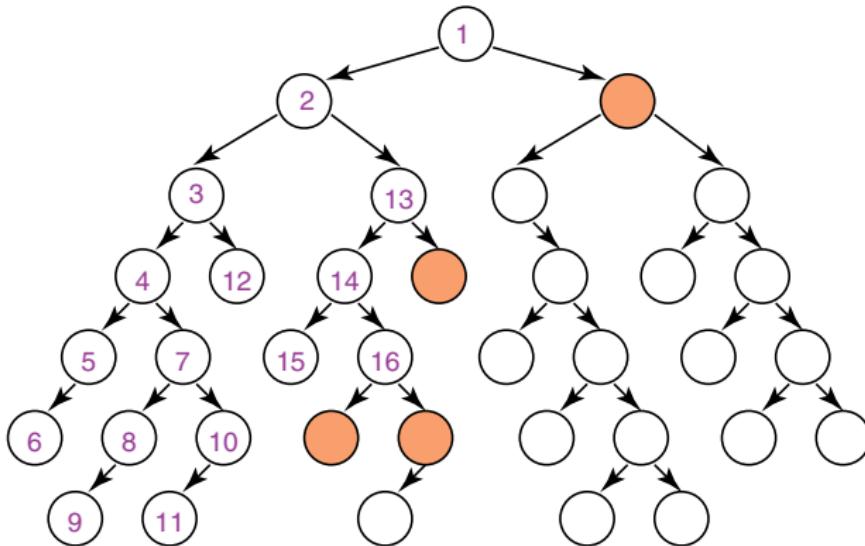


[1] \rightsquigarrow [2,3] \rightsquigarrow [3,4,5] \rightsquigarrow [4,5,6,7] $\rightsquigarrow \dots$

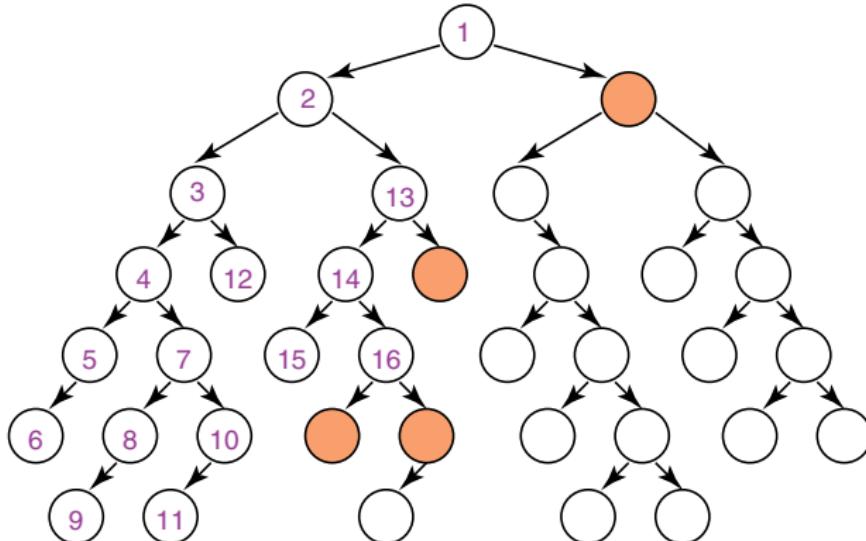
```
add2frontier(Children, [], Children).
```

```
add2frontier(Children, [H|T], [H|More]) :-  
    add2frontier(Children, T, More).
```

Depth-first: stack (LIFO)



Depth-first: stack (LIFO)



```
add2frontier([], Rest, Rest).
```

```
add2frontier([H|T], Rest, [H|TRest]) :-  
    add2frontier(T, Rest, TRest).
```

If-then-else and cut !

i :- p,! ,q.

i :- r.

p.

r.

| ?- i.

If-then-else and cut !

i :- p,! ,q. [i]

i :- r.

p.

r.

| ?- i.

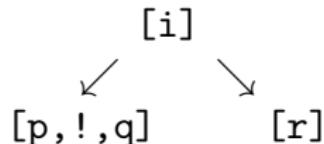
If-then-else and cut !

i :- p,! ,q.

i :- r.

p.

r.



| ?- i.

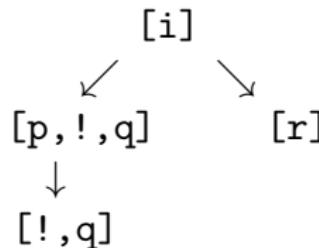
If-then-else and cut !

i :- p, !, q.

i :- r.

p.

r.



Cut ! is true but destroys backtracking.

If-then-else and cut !

i :- p, ! , q.	[i]
i :- r.	[p, ! , q]
p.	[! , q]
r.	[q]

| ?- i.

Cut ! is true but destroys backtracking.

If-then-else and cut !

i :- p, !, q.	[i]
i :- r.	[p, !, q]
p.	[!, q]
r.	[q]

| ?- i.

no

Cut ! is true but destroys backtracking.

Review: Depth-first as frontier search

```
prove([],_).      % goal([]).
prove(Node,KB) :- arc(Node,Next,KB), prove(Next,KB).
```

Review: Depth-first as frontier search

```
prove([],_).      % goal([]).
prove(Node,KB) :- arc(Node,Next,KB), prove(Next,KB).

fs([[]|_],_).

fs([Node|More],KB) :- findall(X,arc(Node,X,KB),L),
                     append(L,More,NewFrontier),
                     fs(NewFrontier,KB).
```

Review: Depth-first as frontier search

```
prove([],_).      % goal([]).
prove(Node,KB) :- arc(Node,Next,KB), prove(Next,KB).

fs([[]|_],_).

fs([Node|More],KB) :- findall(X,arc(Node,X,KB),L),
                     append(L,More,NewFrontier),
                     fs(NewFrontier,KB).
```

Cut?

Tracking the frontier

[[i]]

i :- p,! ,q. [i]

i :- r.

p.

r.

| ?- i.

Tracking the frontier

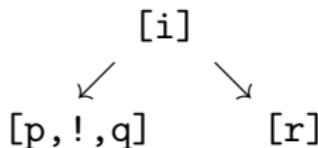
$[[i]] \rightsquigarrow [[p, !, q], [r]]$

i :- p, !, q.

i :- r.

p.

r.



| ?- i.

Tracking the frontier

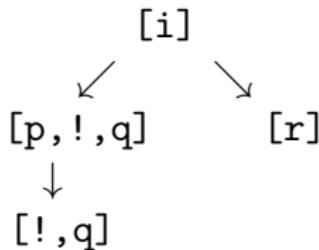
$[[i]] \rightsquigarrow [[p, !, q], [r]] \rightsquigarrow [[!, q], [r]]$

$i :- p, !, q.$

$i :- r.$

$\textcolor{red}{p}.$

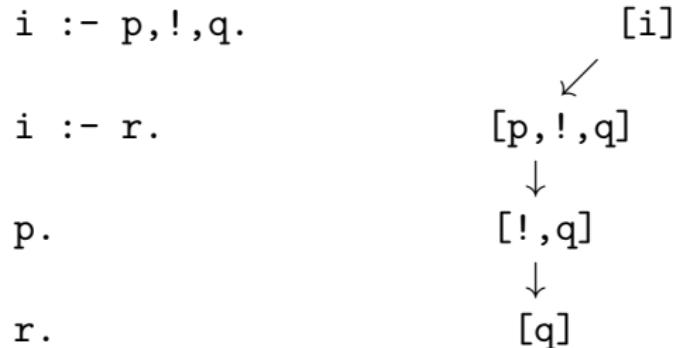
$r.$



| ?- i.

Tracking the frontier

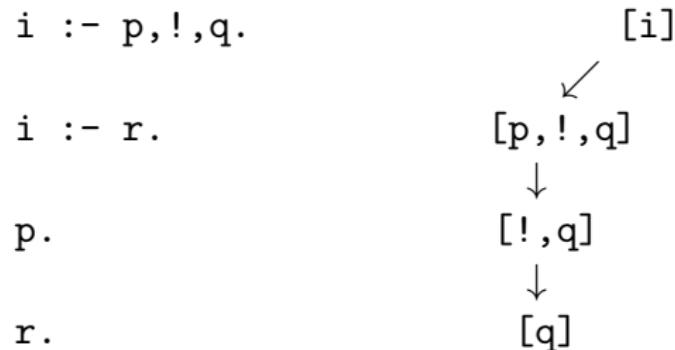
$[[i]] \rightsquigarrow [[p, !, q], [r]] \rightsquigarrow [[!, q], [r]] \rightsquigarrow [[q]]$



| ?- i.

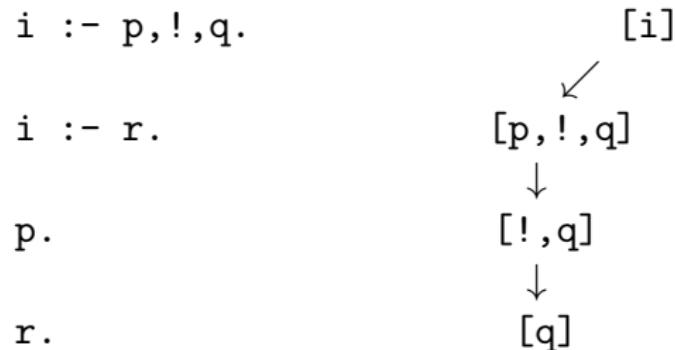
Tracking the frontier

$[[i]] \rightsquigarrow [[p, !, q], [r]] \rightsquigarrow [[!, q], [r]] \rightsquigarrow [[q]] \rightsquigarrow []$



Tracking the frontier

$[[i]] \rightsquigarrow [[p, !, q], [r]] \rightsquigarrow [[!, q], [r]] \rightsquigarrow [[q]] \rightsquigarrow []$



| ?- i.

no

Cut via frontier depth-first search

```
fs([]|_), _).
```

```
fs([Node|More], KB) :-  
    findall(X, arc(Node, X, KB), L),  
    append(L, More, NewFrontier),  
    fs(NewFrontier, KB).
```

Cut via frontier depth-first search

```
fs([] | _) , _).
```

```
fs([[cut | T] | _], KB)) :- fs([T], KB).
```

```
fs([Node | More], KB) :-  
    findall(X, arc(Node, X, KB), L),  
    append(L, More, NewFrontier),  
    fs(NewFrontier, KB).
```

Cut via frontier depth-first search

```
fs([]|_), _ .  
  
fs([[cut|T]|_], KB) :- fs([T], KB).  
  
fs([Node|More], KB) :- Node = [H|_], H \== cut,  
                      findall(X, arc(Node, X, KB), L),  
                      append(L, More, NewFrontier),  
                      fs(NewFrontier, KB).
```

Cut via frontier depth-first search

```
fs([]|_),_.
```

```
fs([[cut|T]|_],KB)) :- fs([T],KB).
```

```
fs([Node|More],KB) :- Node = [H|_], H\== cut,  
                      findall(X,arc(Node,X,KB),L),  
                      append(L,More,NewFrontier),  
                      fs(NewFrontier,KB).
```

```
if(p,q,r) :- (p,! ,q); r.           % contra (p,q);r
```

Cut via frontier depth-first search

```
fs([]|_),_.
```

```
fs([[cut|T]|_],KB)) :- fs([T],KB).
```

```
fs([Node|More],KB) :- Node = [H|_], H\== cut,  
                      findall(X,arc(Node,X,KB),L),  
                      append(L,More,NewFrontier),  
                      fs(NewFrontier,KB).
```

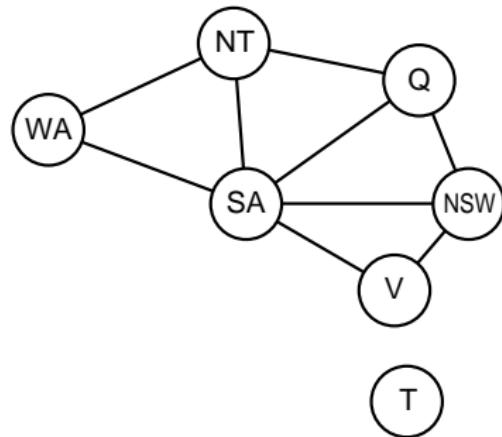
```
if(p,q,r) :- (p,! ,q); r.           % contra (p,q);r
```

```
negation-as-failure(p) :- (p,! ,fail); true.
```

Graph modeling

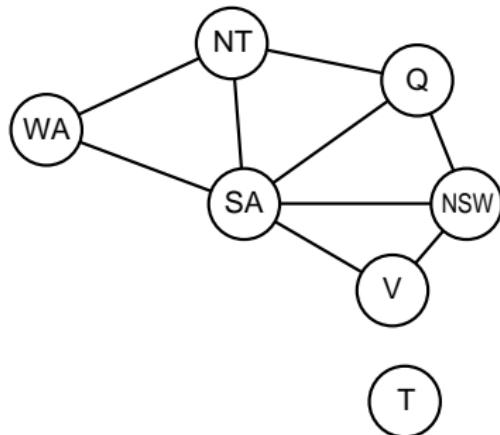


Graph modeling



Russell & Norvig

Graph modeling



Russell & Norvig

```
arc(wa,nt).      arc(nt,q).      arc(q,nsw).  
arc(wa,sa).      arc(nt,sa).     arc(sa,q).  
arc(sa,nsw).     arc(sa,v).      arc(v,nsw).  
  
arc2(X,Y) :- arc(X,Y) ; arc(Y,X).
```

Non-termination (due to poor choices)

i :- p,q. [i]

i :- r.

p :- i.

r.

| ?- i.

prove([] ,_).

prove([H|T] ,KB) :- member([H|B] ,KB) , append(B,T,Next) ,
 prove(Next,KB) .

| ?- prove([i], [[i,p,q],[i,r],[p,i],[r]]).

Non-termination (due to poor choices)

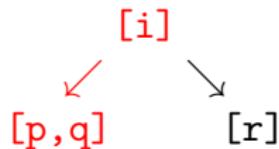
i :- p,q.

i :- r.

p :- i.

r.

| ?- i.



prove([] ,_).

prove([H|T] ,KB) :- member([H|B] ,KB) , append(B,T,Next) ,
 prove(Next,KB) .

| ?- prove([i],[[i,p,q],[i,r],[p,i],[r]]).

Non-termination (due to poor choices)

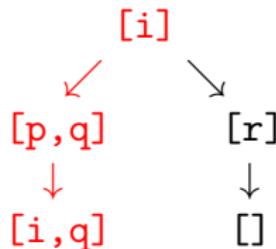
i :- p,q.

i :- r.

p :- i.

r.

| ?- i.



prove([] ,_).

prove([H|T] ,KB) :- member([H|B] ,KB) , append(B,T,Next) ,
 prove(Next,KB) .

| ?- prove([i],[[i,p,q],[i,r],[p,i],[r]]).

Non-termination (due to poor choices)

i :- p,q.

i :- r.

p :- i.

r.

| ?- i.

[i]

[p,q]

[i,q]

[p,q,q]

[r]

[]

[r,q]

prove([] ,_).

prove([H|T] ,KB) :- member([H|B] ,KB) , append(B,T,Next) ,
 prove(Next,KB) .

| ?- prove([i], [[i,p,q],[i,r],[p,i],[r]]).

Non-termination (due to poor choices)

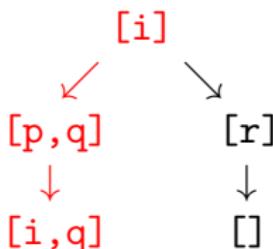
i :- p,q.

i :- r.

p :- i.

r.

| ?- i.



prove([], _).

prove([H|T], KB) :- member([H|B], KB), append(B, T, Next),
prove(Next, KB).

| ?- prove([i], [[i,p,q],[i,r],[p,i],[r]]).

Non-termination (due to poor choices)

i :- p,q.

i :- r.

p :- i.

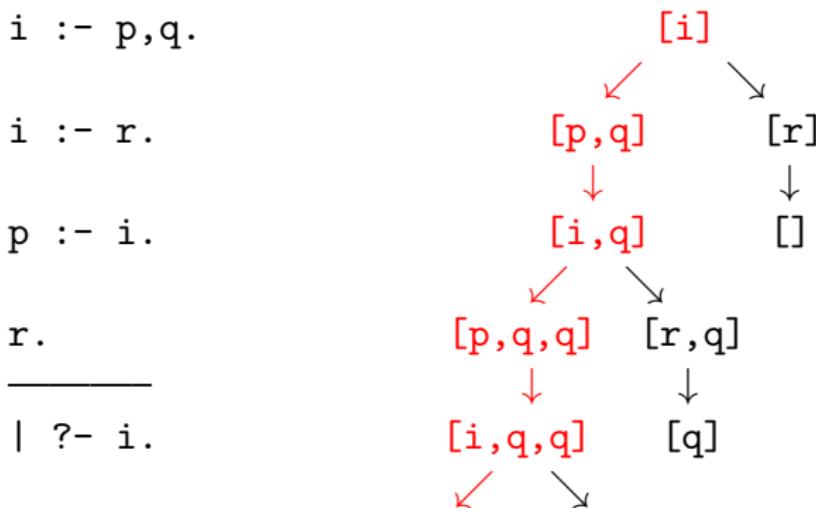
r.

| ?- i.

prove([],_).

prove([H|T],KB) :- member([H|B],KB), append(B,T,Next),
prove(Next,KB).

| ?- prove([i], [[i,p,q],[i,r],[p,i],[r]]).



Determinization (eliminate choice)

A fsm [Trans, Final, Q0] such that

for all $[Q, X, Q_n]$ *and* $[Q, X, Q_n']$ *in Trans,* $Q_n = Q_n'$

is a *deterministic finite automaton* (DFA).

Determinization (eliminate choice)

A fsm [Trans, Final, Q0] such that

for all $[Q, X, Q_n]$ *and* $[Q, X, Q_n']$ *in Trans,* $Q_n = Q_n'$

is a *deterministic finite automaton* (DFA).

Fact. Every fsm has a DFA accepting the same language.

Determinization (eliminate choice)

A fsm [Trans, Final, Q0] such that

for all $[Q, X, Qn]$ and $[Q, X, Qn']$ in Trans, $Qn = Qn'$

is a *deterministic finite automaton* (DFA).

Fact. Every fsm has a DFA accepting the same language.

Proof: Subset (powerset) construction

Apply to arc,goal, contra Trans,Final:

```
arcD(NodeList,NextList) :-  
    setof(Next, arcLN(NodeList,Next), NextList).  
  
arcLN(NodeList,Next) :- member(Node,NodeList),  
    arc(Node,Next).
```

Determinization (eliminate choice)

A fsm [Trans, Final, Q0] such that

for all $[Q, X, Qn]$ and $[Q, X, Qn']$ in Trans, $Qn = Qn'$

is a *deterministic finite automaton* (DFA).

Fact. Every fsm has a DFA accepting the same language.

Proof: Subset (powerset) construction

Apply to arc,goal, contra Trans,Final:

```
arcD(NodeList,NextList) :-  
    setof(Next, arcLN(NodeList,Next), NextList).  
  
arcLN(NodeList,Next) :- member(Node,NodeList),  
    arc(Node,Next).  
  
goalD(NodeList):-member(Node,NodeList),goal(Node).
```

Determinization (eliminate choice)

A fsm [Trans, Final, Q0] such that

for all $[Q, X, Qn]$ and $[Q, X, Qn']$ in Trans, $Qn = Qn'$

is a *deterministic finite automaton* (DFA).

Fact. Every fsm has a DFA accepting the same language.

Proof: Subset (powerset) construction

Apply to arc,goal, contra Trans,Final:

```
arcD(NodeList,NextList) :-  
    setof(Next, arcLN(NodeList,Next), NextList).  
  
arcLN(NodeList,Next) :- member(Node,NodeList),  
    arc(Node,Next).  
  
goalD(NodeList):-member(Node,NodeList),goal(Node).  
  
searchD(NL) :- goalD(NL);  
    (arcD(NL,NL2), searchD(NL2)).
```

Determinization (eliminate choice)

A fsm [Trans, Final, Q0] such that

for all $[Q, X, Qn]$ and $[Q, X, Qn']$ in Trans, $Qn = Qn'$

is a *deterministic finite automaton* (DFA).

Fact. Every fsm has a DFA accepting the same language.

Proof: Subset (powerset) construction

Apply to arc,goal, contra Trans,Final:

```
arcD(NodeList,NextList) :-  
    setof(Next, arcLN(NodeList,Next), NextList).
```

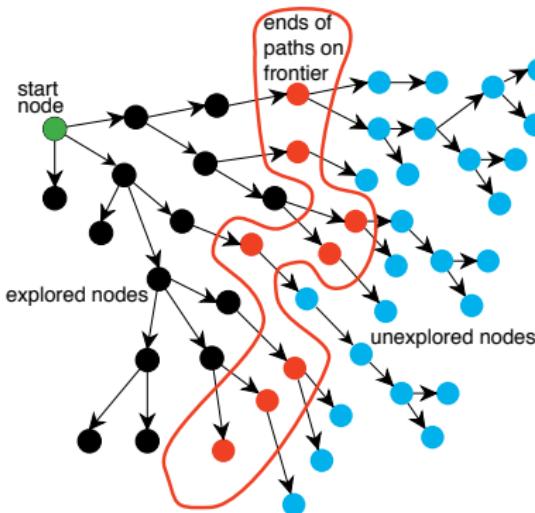
```
arcLN(NodeList,Next) :- member(Node,NodeList),  
    arc(Node,Next).
```

```
goalD(NodeList):-member(Node,NodeList),goal(Node).
```

```
searchD(NL) :- goalD(NL);  
    (arcD(NL,NL2), searchD(NL2)).
```

```
search(Node) :- searchD([Node]).
```

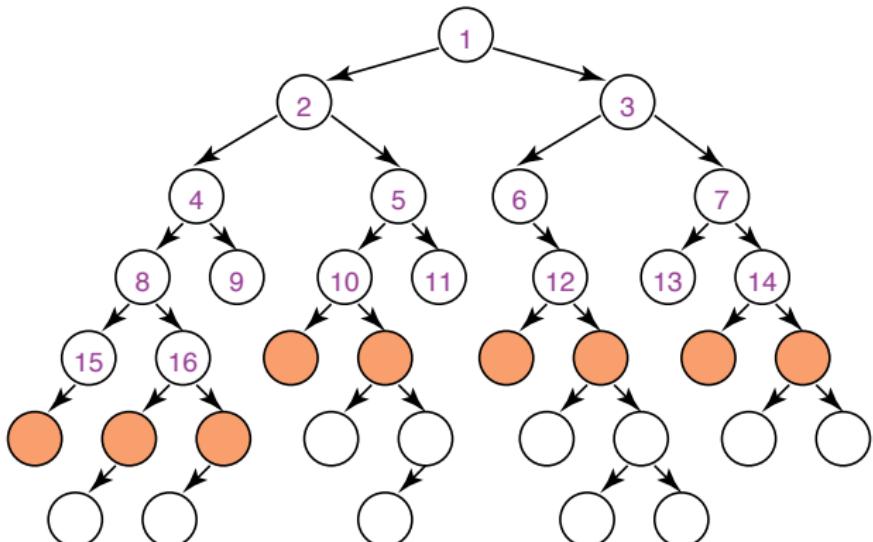
Frontier search



Poole & Mackworth

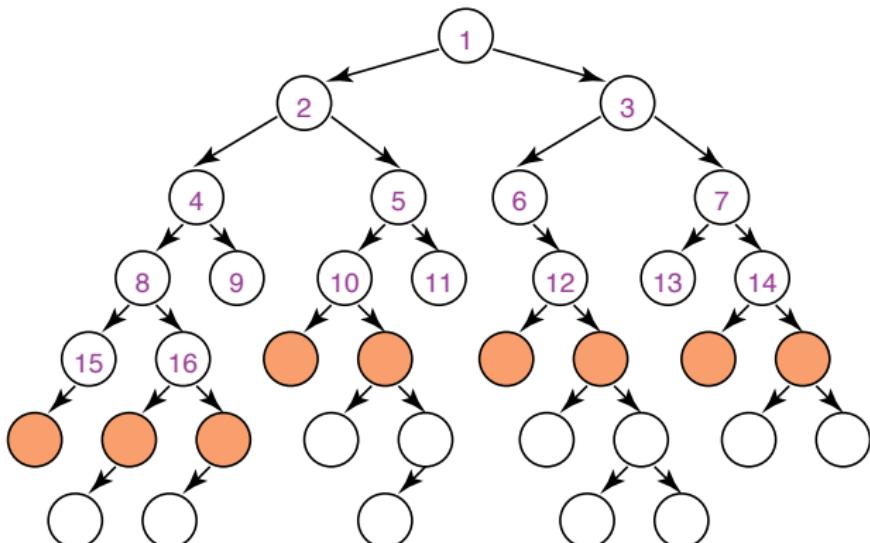
```
search(Node) :- frontierSearch([Node]).  
  
frontierSearch([Node|_]) :- goal(Node).  
  
frontierSearch([Node|Rest]) :-  
    findall(Next, arc(Node, Next), Children),  
    add2frontier(Children, Rest, NewFrontier),  
    frontierSearch(NewFrontier).
```

Breadth-first: queue (FIFO)



[1] \rightsquigarrow [2, 3] \rightsquigarrow [3, 4, 5] \rightsquigarrow [4, 5, 6, 7] $\rightsquigarrow \dots$

Breadth-first: queue (FIFO)

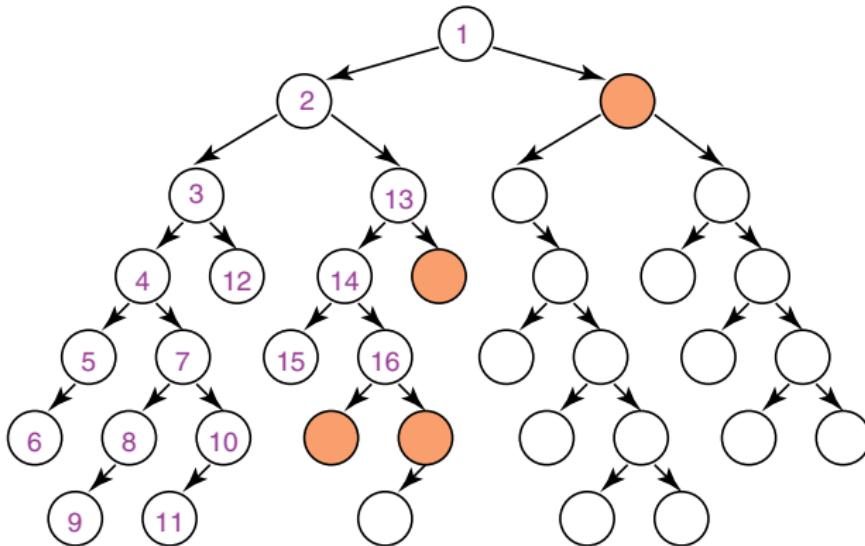


[1] \rightsquigarrow [2,3] \rightsquigarrow [3,4,5] \rightsquigarrow [4,5,6,7] $\rightsquigarrow \dots$

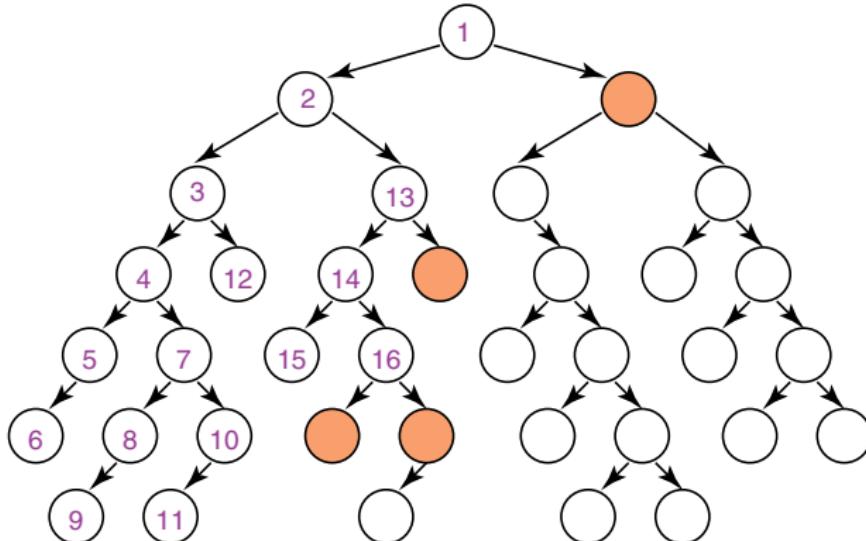
```
add2frontier(Children, [], Children).
```

```
add2frontier(Children, [H|T], [H|More]) :-  
    add2frontier(Children, T, More).
```

Depth-first: stack (LIFO)



Depth-first: stack (LIFO)



```
add2frontier([], Rest, Rest).
```

```
add2frontier([H|T], Rest, [H|TRest]) :-  
    add2frontier(T, Rest, TRest).
```

If-then-else and cut !

i :- p,! ,q.

i :- r.

p.

r.

| ?- i.

If-then-else and cut !

i :- p,! ,q. [i]

i :- r.

p.

r.

| ?- i.

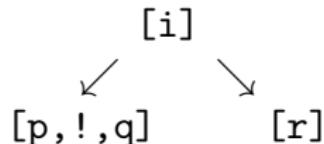
If-then-else and cut !

i :- p,! ,q.

i :- r.

p.

r.



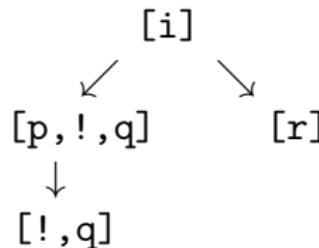
If-then-else and cut !

i :- p, !, q.

i :- r.

p.

r.



| ?- i.

Cut ! is true but destroys backtracking.

If-then-else and cut !

```
i :- p,! ,q. [i]  
i :- r. [p,! ,q]  
p. [! ,q]  
r. [q]
```

The diagram illustrates the execution flow of a Prolog query. It starts with the goal `i`. The first clause `i :- p, !, q.` is selected. The `!` operator cuts the search space, so the second clause `i :- r.` is not considered. The query then backtracks to the goal `p`. Since `p` is true, it succeeds. Finally, the query backtracks to the goal `r`, which also succeeds.

```
| ?- i.
```

Cut ! is true but destroys backtracking.

If-then-else and cut !

i :- p, !, q.	[i]
i :- r.	[p, !, q]
p.	[!, q]
r.	[q]

| ?- i.

no

Cut ! is true but destroys backtracking.

Review: Depth-first as frontier search

```
prove([],_).      % goal([]).
prove(Node,KB) :- arc(Node,Next,KB), prove(Next,KB).
```

Review: Depth-first as frontier search

```
prove([],_).      % goal([]).
prove(Node,KB) :- arc(Node,Next,KB), prove(Next,KB).

fs([[]|_],_).

fs([Node|More],KB) :- findall(X,arc(Node,X,KB),L),
                     append(L,More,NewFrontier),
                     fs(NewFrontier,KB).
```

Review: Depth-first as frontier search

```
prove([],_).      % goal([]).
prove(Node,KB) :- arc(Node,Next,KB), prove(Next,KB).

fs([[]|_],_).

fs([Node|More],KB) :- findall(X,arc(Node,X,KB),L),
                     append(L,More,NewFrontier),
                     fs(NewFrontier,KB).
```

Cut?

Tracking the frontier

[[i]]

i :- p,! ,q. [i]

i :- r.

p.

r.

| ?- i.

Tracking the frontier

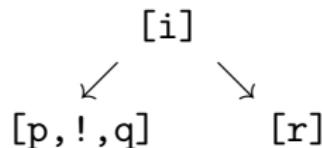
$[[i]] \rightsquigarrow [[p, !, q], [r]]$

i :- p, !, q.

i :- r.

p.

r.



| ?- i.

Tracking the frontier

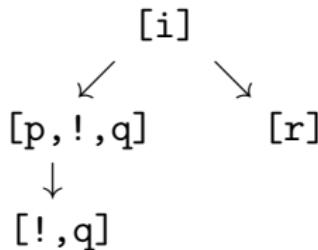
$[[i]] \rightsquigarrow [[p, !, q], [r]] \rightsquigarrow [[!, q], [r]]$

$i :- p, !, q.$

$i :- r.$

$\textcolor{red}{p}.$

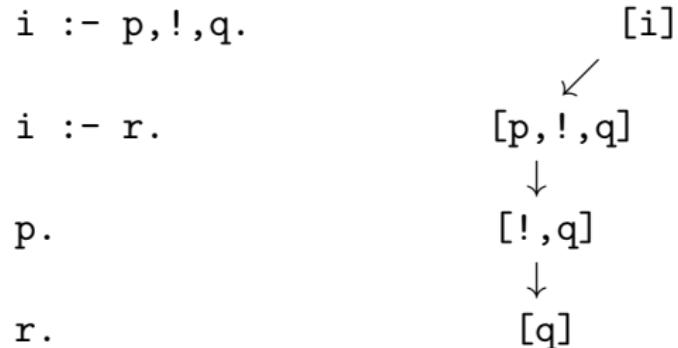
$r.$



| ?- i.

Tracking the frontier

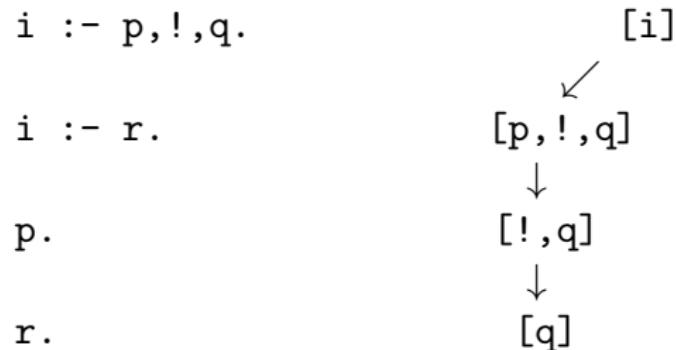
$[[i]] \rightsquigarrow [[p, !, q], [r]] \rightsquigarrow [[!, q], [r]] \rightsquigarrow [[q]]$



| ?- i.

Tracking the frontier

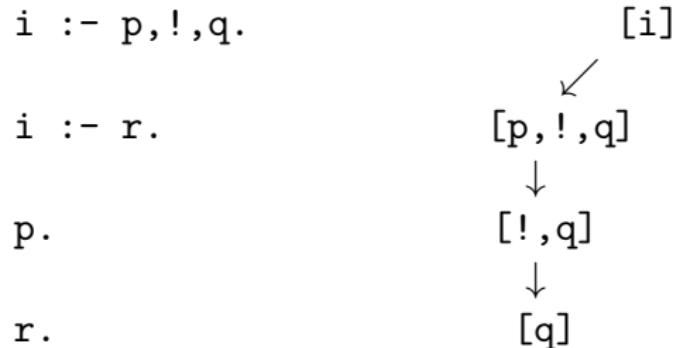
$[[i]] \rightsquigarrow [[p, !, q], [r]] \rightsquigarrow [[!, q], [r]] \rightsquigarrow [[q]] \rightsquigarrow []$



| ?- i.

Tracking the frontier

$[[i]] \rightsquigarrow [[p, !, q], [r]] \rightsquigarrow [[!, q], [r]] \rightsquigarrow [[q]] \rightsquigarrow []$



| ?- i.

no

Cut via frontier depth-first search

```
fs([]|_), _).
```

```
fs([Node|More], KB) :-  
    findall(X, arc(Node, X, KB), L),  
    append(L, More, NewFrontier),  
    fs(NewFrontier, KB).
```

Cut via frontier depth-first search

```
fs([] | _) , _).
```

```
fs([[cut | T] | _], KB)) :- fs([T], KB).
```

```
fs([Node | More], KB) :-  
    findall(X, arc(Node, X, KB), L),  
    append(L, More, NewFrontier),  
    fs(NewFrontier, KB).
```

Cut via frontier depth-first search

```
fs([]|_), _ .  
  
fs([[cut|T]|_], KB) :- fs([T], KB).  
  
fs([Node|More], KB) :- Node = [H|_], H \== cut,  
                      findall(X, arc(Node, X, KB), L),  
                      append(L, More, NewFrontier),  
                      fs(NewFrontier, KB).
```

Cut via frontier depth-first search

```
fs([]|_),_.
```

```
fs([[cut|T]|_],KB)) :- fs([T],KB).
```

```
fs([Node|More],KB) :- Node = [H|_], H\== cut,  
                      findall(X,arc(Node,X,KB),L),  
                      append(L,More,NewFrontier),  
                      fs(NewFrontier,KB).
```

```
if(p,q,r) :- (p,! ,q); r.           % contra (p,q);r
```

Cut via frontier depth-first search

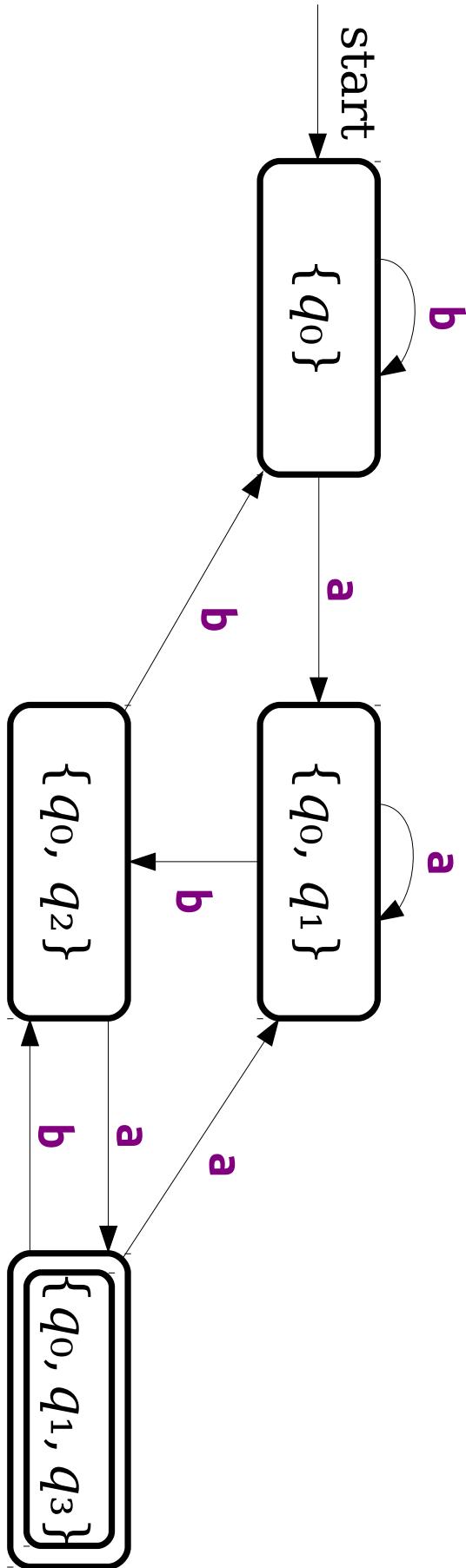
```
fs([]|_),_.
```

```
fs([[cut|T]|_],KB)) :- fs([T],KB).
```

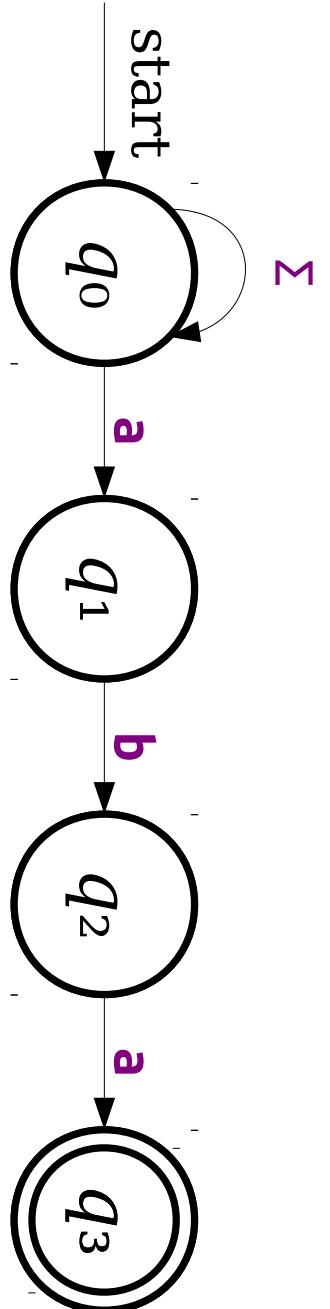
```
fs([Node|More],KB) :- Node = [H|_], H\== cut,  
                      findall(X,arc(Node,X,KB),L),  
                      append(L,More,NewFrontier),  
                      fs(NewFrontier,KB).
```

```
if(p,q,r) :- (p,! ,q); r.           % contra (p,q);r
```

```
negation-as-failure(p) :- (p,! ,fail); true.
```



	a	b
$\{q_0\}$	$\{q_0\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_2\}$	$\{q_0, q_2\}$	$\{q_0\}$
$*\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



Frontier search (manage choices)

```
frontierSearch([Node|Rest]) :- goal(Node),  
    (findall(Next, arc(Node, Next), Children),  
     add2frontier(Children, Rest, NewFrontier),  
     frontierSearch(NewFrontier)).
```

Frontier search (manage choices)

```
frontierSearch([Node|Rest]) :- goal(Node),  
    (findall(Next, arc(Node, Next), Children),  
     add2frontier(Children, Rest, NewFrontier),  
     frontierSearch(NewFrontier)).
```

Depth first: append(Children, Rest, NewFrontier)

Breadth-first: append(Rest, Children, NewFrontier)

Frontier search (manage choices)

```
frontierSearch([Node|Rest]) :- goal(Node),  
    (findall(Next, arc(Node, Next), Children),  
     add2frontier(Children, Rest, NewFrontier),  
     frontierSearch(NewFrontier)).
```

Depth first: append(Children, Rest, NewFrontier)

Breadth-first: append(Rest, Children, NewFrontier)

For add2frontier(Children, Rest, NewFrontier), require

NewFrontier merges Children and Rest

where a list L is defined to *merge* lists L1 and L2 if

- (a) every member of L is a member of L1 or L2
- (b) every member of L1 or of L2 is a member of L.

Exercise (Prolog)

Suppose a positive integer Seed links nodes 1,2,... in two ways

```
arc(N,M,Seed) :- M is N*Seed.
```

```
arc(N,M,Seed) :- M is N*Seed +1.
```

e.g. Seed=3 gives arcs (1,3), (1,4), (3,9), (3, 10) ...

Exercise (Prolog)

Suppose a positive integer Seed links nodes 1,2,... in two ways

```
arc(N,M,Seed) :- M is N*Seed.
```

```
arc(N,M,Seed) :- M is N*Seed +1.
```

e.g. Seed=3 gives arcs (1,3), (1,4), (3,9), (3, 10) ...

Goal nodes are multiples of a positive integer Target

```
goal(N,Target) :- 0 is N mod Target.
```

e.g. Target=13 gives goals 13, 26, 39 ...

Exercise (Prolog)

Suppose a positive integer Seed links nodes 1,2,... in two ways

```
arc(N,M,Seed) :- M is N*Seed.
```

```
arc(N,M,Seed) :- M is N*Seed +1.
```

e.g. Seed=3 gives arcs (1,3), (1,4), (3,9), (3, 10) ...

Goal nodes are multiples of a positive integer Target

```
goal(N,Target) :- 0 is N mod Target.
```

e.g. Target=13 gives goals 13, 26, 39 ...

Modify frontier search to define predicates

```
breadth1st(+Start, ?Found, +Seed, +Target)
```

```
depth1st(+Start, ?Found, +Seed, +Target)
```

that search breadth-first and depth-first respectively for a Target-goal node Found linked to Start by Seed-arcs.

Refining frontier search

For `add2frontier(Children, Rest, NewFrontier)`, require

`NewFrontier` merges `Children` and `Rest`

and for `NewFrontier = [Head|Tail]`, ensure

`Head` is “no worse than” any in `Tail`.

Refining frontier search

For `add2frontier(Children, Rest, NewFrontier)`, require

`NewFrontier` merges `Children` and `Rest`

and for `NewFrontier = [Head|Tail]`, ensure

`Head` is “no worse than” any in `Tail`.

What can it mean for `Node1` to be *no worse than* `Node2` ?

(A1) `Node1` costs no more than `Node2`

Refining frontier search

For `add2frontier(Children, Rest, NewFrontier)`, require

`NewFrontier` merges `Children` and `Rest`

and for `NewFrontier = [Head|Tail]`, ensure

`Head` is “no worse than” any in `Tail`.

What can it mean for `Node1` to be *no worse than* `Node2` ?

(A1) `Node1` costs no more than `Node2`

(A2) `Node1` is deemed no further from a goal node than `Node2`

Refining frontier search

For `add2frontier(Children, Rest, NewFrontier)`, require

`NewFrontier` merges `Children` and `Rest`

and for `NewFrontier = [Head|Tail]`, ensure

`Head` is “no worse than” any in `Tail`.

What can it mean for `Node1` to be *no worse than* `Node2` ?

(A1) `Node1` costs no more than `Node2`

(A2) `Node1` is deemed no further from a goal node than `Node2`

(A3) some mix of (A1) and (A2)

Refining frontier search

For `add2frontier(Children, Rest, NewFrontier)`, require

`NewFrontier` merges `Children` and `Rest`

and for `NewFrontier = [Head|Tail]`, ensure

`Head` is “no worse than” any in `Tail`.

What can it mean for `Node1` to be *no worse than* `Node2` ?

(A1) `Node1` costs no more than `Node2`

~ \rightarrow minimum cost search (= breadth-first if every arc costs 1)

(A2) `Node1` is deemed no further from a goal node than `Node2`

(A3) some mix of (A1) and (A2)

Refining frontier search

For `add2frontier(Children, Rest, NewFrontier)`, require

`NewFrontier` merges `Children` and `Rest`

and for `NewFrontier = [Head|Tail]`, ensure

`Head` is “no worse than” any in `Tail`.

What can it mean for `Node1` to be *no worse than* `Node2` ?

(A1) `Node1` costs no more than `Node2`

~~ minimum cost search (= breadth-first if every arc costs 1)

(A2) `Node1` is deemed no further from a goal node than `Node2`

~~ best-first search (= depth-first for heuristic $\propto \text{depth}^{-1}$)

(A3) some mix of (A1) and (A2)

Refining frontier search

For `add2frontier(Children, Rest, NewFrontier)`, require

`NewFrontier` merges `Children` and `Rest`

and for `NewFrontier = [Head|Tail]`, ensure

`Head` is “no worse than” any in `Tail`.

What can it mean for `Node1` to be *no worse than* `Node2` ?

(A1) `Node1` costs no more than `Node2`

~~ minimum cost search (= breadth-first if every arc costs 1)

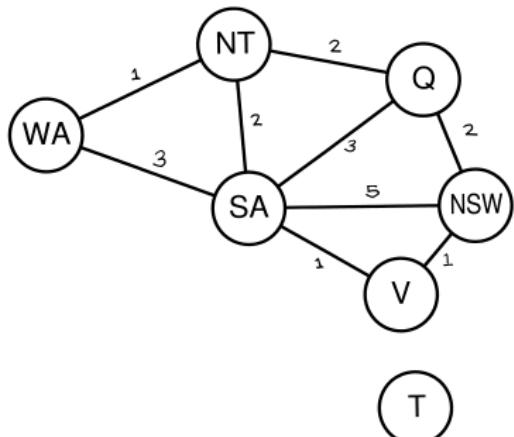
(A2) `Node1` is deemed no further from a goal node than `Node2`

~~ best-first search (= depth-first for heuristic $\propto \text{depth}^{-1}$)

(A3) some mix of (A1) and (A2)

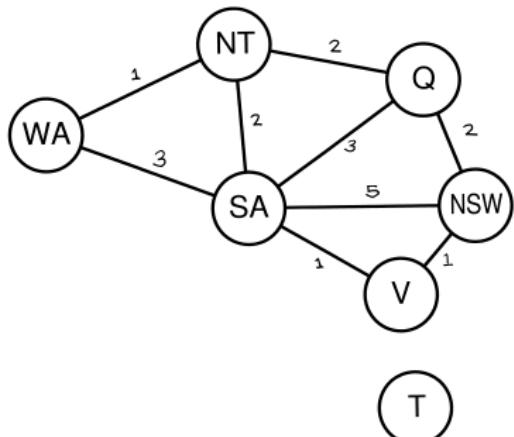
~~ A-star

Arc costs (space, time, money, ...)



```
arc(wa,nt,1).    arc(nt,q,2).  
arc(q,nsw,2).    arc(wa,sa,3).  
arc(nt,sa,2).    arc(sa,q,3).  
arc(sa,nsw,5).    arc(sa,v,1).  
arc(v,nsw,1).
```

Arc costs (space, time, money, ...)

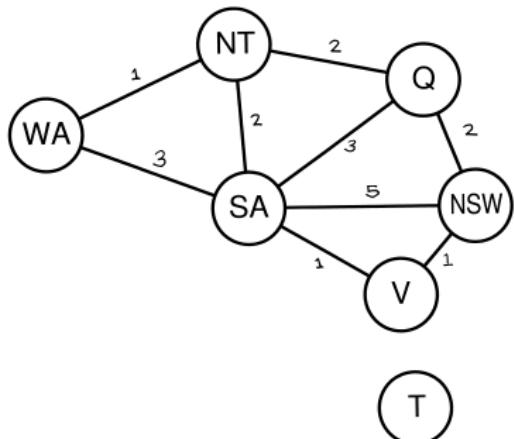


arc(wa,nt,1). arc(nt,q,2).
arc(q,nsw,2). arc(wa,sa,3).
arc(nt,sa,2). arc(sa,q,3).
arc(sa,nsw,5). arc(sa,v,1).
arc(v,nsw,1).

$$\text{cost}(\text{wa}, \text{nt}, \text{q}, \text{nsw}) = 1 + 2 + 2 = 5$$

$$\text{cost}(x_1, x_2, \dots, x_{k+1}) := \sum_{i=1}^k \text{cost}(x_i, x_{i+1})$$

Arc costs (space, time, money, ...)



arc(wa,nt,1). arc(nt,q,2).
arc(q,nsw,2). arc(wa,sa,3).
arc(nt,sa,2). arc(sa,q,3).
arc(sa,nsw,5). arc(sa,v,1).
arc(v,nsw,1).

$$\text{cost}(\text{wa}, \text{nt}, \text{q}, \text{nsw}) = 1 + 2 + 2 = 5$$

$$\text{cost}(x_1, x_2, \dots, x_{k+1}) := \sum_{i=1}^k \text{cost}(x_i, x_{i+1})$$

$$\text{cost}(\text{wa}, \text{sa}, \text{nsw}) = 3 + 5 = 8$$

Heuristics

$h(\text{Node})$ = estimate the minimum cost of
a path from Node to a goal node

Heuristics

$h(\text{Node})$ = estimate the minimum cost of
a path from Node to a goal node

EXAMPLES

- ▶ Fsm accept where node = [Q, String] and every arc costs 1

$$h([Q, \text{String}]) = \text{length}(\text{String})$$

Heuristics

$h(\text{Node})$ = estimate the minimum cost of
a path from Node to a goal node

EXAMPLES

- ▶ Fsm accept where node = [Q, String] and every arc costs 1

$$h([\text{Q}, \text{String}]) = \text{length}(\text{String})$$

- ▶ Prolog search where node = list of propositions to prove, and every arc costs 1

$$h(\text{List}) = \text{length}(\text{List})$$

Heuristics

$h(\text{Node})$ = estimate the minimum cost of
a path from Node to a goal node

EXAMPLES

- ▶ Fsm accept where node = [Q, String] and every arc costs 1

$$h([\text{Q}, \text{String}]) = \text{length}(\text{String})$$

- ▶ Prolog search where node = list of propositions to prove, and every arc costs 1

$$h(\text{List}) = \text{length}(\text{List})$$

- ▶ Node = point on a Euclidean plane, cost = distance between nodes, goal is a point G

$$h(\text{Node}) = \text{straight-line distance to G}$$

Heuristics

$h(\text{Node})$ = estimate the minimum cost of
a path from Node to a goal node

EXAMPLES

- ▶ Fsm accept where node = [Q, String] and every arc costs 1

$$h([\text{Q}, \text{String}]) = \text{length}(\text{String})$$

- ▶ Prolog search where node = list of propositions to prove, and every arc costs 1

$$h(\text{List}) = \text{length}(\text{List})$$

- ▶ Node = point on a Euclidean plane, cost = distance between nodes, goal is a point G

$$h(\text{Node}) = \text{straight-line distance to G}$$

- ▶ estimate assuming lots of arcs (simplifying the problem)

Best-first search

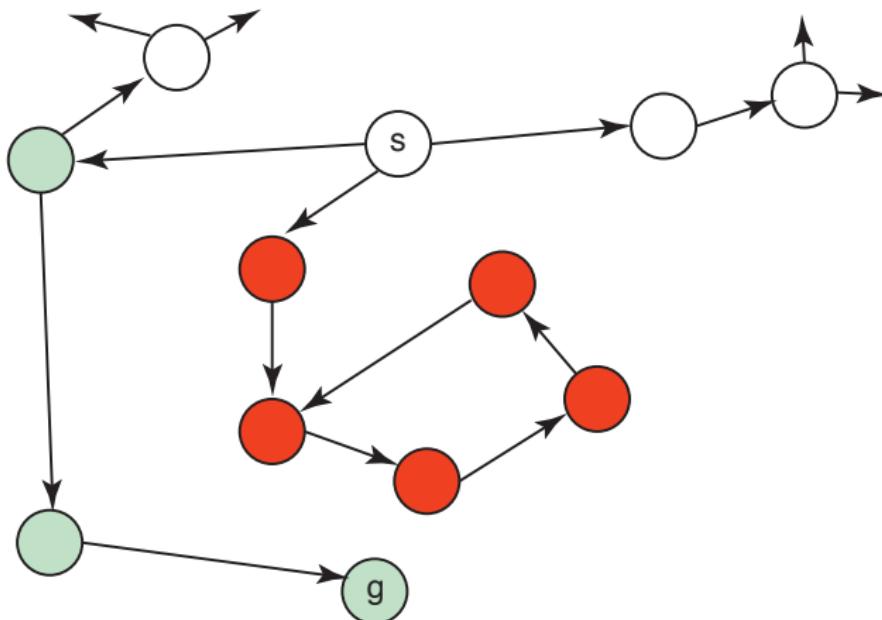
Form NewFrontier = [Head|Tail] such that

$$h(\text{Head}) \leq h(\text{Node}) \text{ for every Node in Tail}$$

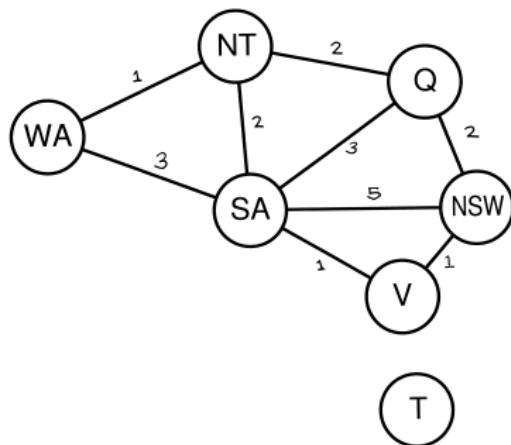
Best-first search

Form NewFrontier = [Head|Tail] such that

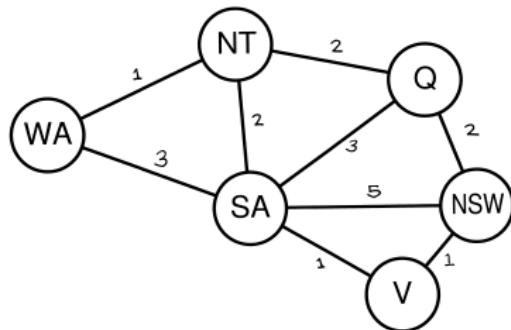
$$h(\text{Head}) \leq h(\text{Node}) \text{ for every Node in Tail}$$



Min-cost



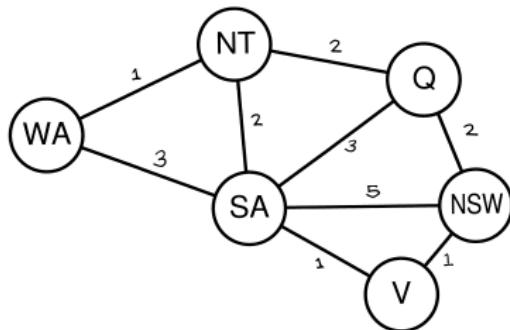
Min-cost \neq breadth-first



$$\text{cost}(n_1 \dots n_k) = \sum_{i=1}^{k-1} \text{cost}(n_i, n_{i+1})$$

$$\begin{aligned}\text{cost(wa nt q nsw)} &= 5 \\ \text{cost(wa sa nsw)} &= 8\end{aligned}$$

Min-cost \neq breadth-first



$$\text{cost}(n_1 \dots n_k) = \sum_{i=1}^{k-1} \text{cost}(n_i, n_{i+1})$$

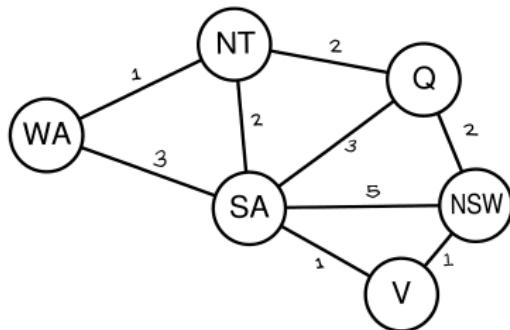
T

$$\begin{aligned}\text{cost(wa nt q nsw)} &= 5 \\ \text{cost(wa sa nsw)} &= 8\end{aligned}$$

```
add2frontier(Children, Rest, [Head|Tail])
```

$\text{cost}(\text{Start} \dots \text{Head}) \leq \text{cost}(\text{Start} \dots n)$ for each n in Tail ?

Min-cost \neq breadth-first



$$\text{cost}(n_1 \dots n_k) = \sum_{i=1}^{k-1} \text{cost}(n_i, n_{i+1})$$

T

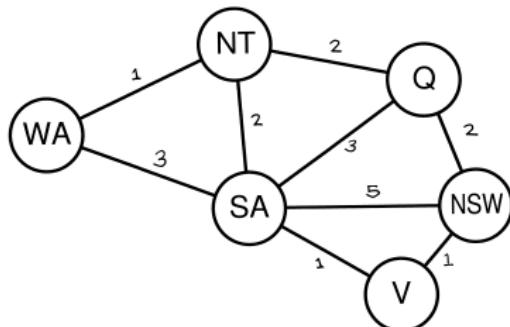
$$\begin{aligned}\text{cost(wa nt q nsw)} &= 5 \\ \text{cost(wa sa nsw)} &= 8\end{aligned}$$

```
add2frontier(Children, Rest, [Head|Tail])
```

$\text{cost}(\text{Start} \dots \text{Head}) \leq \text{cost}(\text{Start} \dots n)$ for each n in Tail ?

- ▶ node \leadsto path

Min-cost \neq breadth-first



$$\text{cost}(n_1 \dots n_k) = \sum_{i=1}^{k-1} \text{cost}(n_i, n_{i+1})$$

T

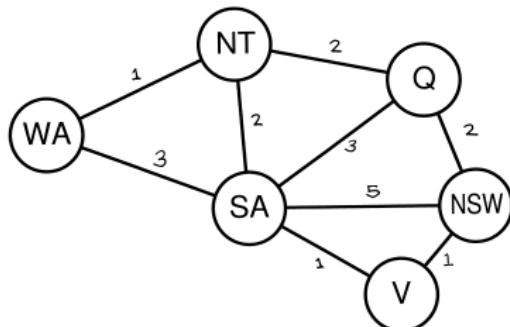
$$\begin{aligned}\text{cost(wa nt q nsw)} &= 5 \\ \text{cost(wa sa nsw)} &= 8\end{aligned}$$

```
add2frontier(Children, Rest, [Head|Tail])
```

$\text{cost}(\text{Start} \dots \text{Head}) \leq \text{cost}(\text{Start} \dots n)$ for each n in Tail ?

- ▶ node \rightsquigarrow path or pair $(n, \text{cost}(\text{Start} \dots n))$

Min-cost \neq breadth-first



$$\text{cost}(n_1 \dots n_k) = \sum_{i=1}^{k-1} \text{cost}(n_i, n_{i+1})$$



$$\begin{aligned}\text{cost(wa nt q nsw)} &= 5 \\ \text{cost(wa sa nsw)} &= 8\end{aligned}$$

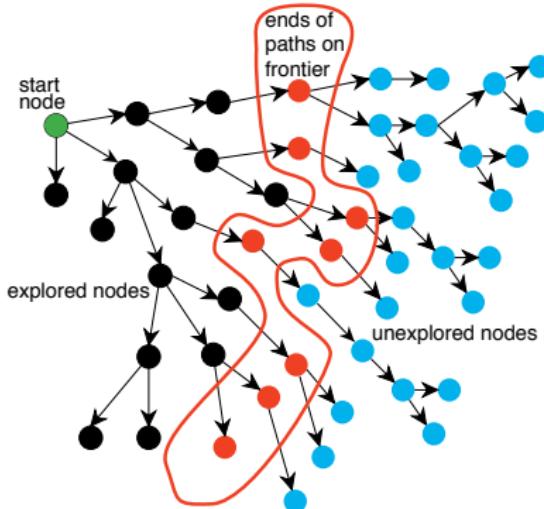
```
add2frontier(Children, Rest, [Head|Tail])
```

$\text{cost}(\text{Start} \dots \text{Head}) \leq \text{cost}(\text{Start} \dots n)$ for each n in Tail ?

- ▶ node \rightsquigarrow path or pair $(n, \text{cost}(\text{Start} \dots n))$
- ▶ what about proximity to goal?

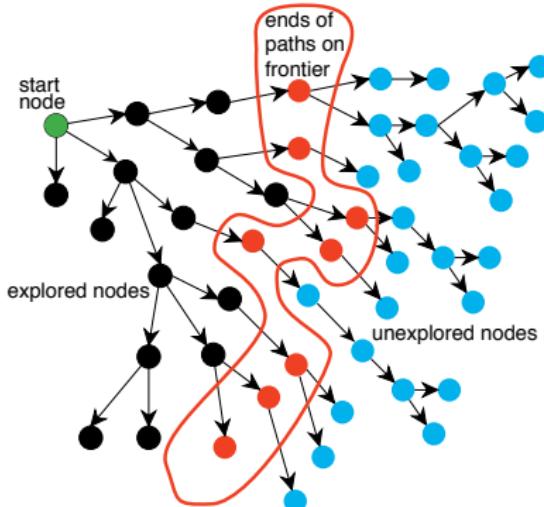
$h(n) = \text{estimate of min cost path } n \dots \text{goal}$

A*



solution = start ... n ... goal
explored

$$f(\text{start} \cdots n) = \underbrace{\text{cost}(\text{start} \cdots n)}_{\text{+ } h(n)}$$



solution = start · · · n · · · goal
explored

$$f(\text{start} \cdots n) = \underbrace{\text{cost}(\text{start} \cdots n)}_{+ h(n)}$$

Ensure Frontier = [Head|Tail] where Head has minimal f

- ▶ $h(n) = 0$ for every $n \rightsquigarrow$ min-cost
- ▶ $\text{cost}(\text{start} \cdots n) = 0$ for every $n \rightsquigarrow$ best-first
(disregarding the past)

Admissibility

A^* is *admissible* (under cost, h) if it returns a solution of min cost whenever a solution exists.

Admissibility

A^* is *admissible* (under cost, h) if it returns a solution of min cost whenever a solution exists.

3 conditions sufficient for admissibility

under-estimate: for every solution $n \cdots \text{goal}$,
$$0 \leq h(n) \leq \text{cost}(n \cdots \text{goal})$$

Admissibility

A^* is *admissible* (under cost, h) if it returns a solution of min cost whenever a solution exists.

3 conditions sufficient for admissibility

under-estimate: for every solution $n \cdots \text{goal}$,
$$0 \leq h(n) \leq \text{cost}(n \cdots \text{goal})$$

termination: for some $\epsilon > 0$, every arc costs $\geq \epsilon$

finite branching: $\{n' \mid \text{arc}(n, n')\}$ is finite for each node n

Admissibility

A^* is *admissible* (under cost, h) if it returns a solution of min cost whenever a solution exists.

3 conditions sufficient for admissibility

under-estimate: for every solution $n \cdots \text{goal}$,
$$0 \leq h(n) \leq \text{cost}(n \cdots \text{goal})$$

termination: for some $\epsilon > 0$, every arc costs $\geq \epsilon$

finite branching: $\{n' \mid \text{arc}(n, n')\}$ is finite for each node n

Assuming the 3 conditions above, let p be a solution.

To show: A^* returns a solution with min cost c .

Admissibility

A^* is *admissible* (under cost, h) if it returns a solution of min cost whenever a solution exists.

3 conditions sufficient for admissibility

under-estimate: for every solution $n \cdots \text{goal}$,
$$0 \leq h(n) \leq \text{cost}(n \cdots \text{goal})$$

termination: for some $\epsilon > 0$, every arc costs $\geq \epsilon$

finite branching: $\{n' \mid \text{arc}(n, n')\}$ is finite for each node n

Assuming the 3 conditions above, let p be a solution.

TO SHOW: A^* returns a solution with min cost c .

Let $F_0 = [\text{Start}]$, F_{n+1} be A^* 's next frontier after F_n ($[]$ if none), and c_n be the cost of the head of F_n (∞ if $F_n = []$).

Admissibility

A^* is *admissible* (under cost, h) if it returns a solution of min cost whenever a solution exists.

3 conditions sufficient for admissibility

under-estimate: for every solution $n \cdots \text{goal}$,
$$0 \leq h(n) \leq \text{cost}(n \cdots \text{goal})$$

termination: for some $\epsilon > 0$, every arc costs $\geq \epsilon$

finite branching: $\{n' \mid \text{arc}(n, n')\}$ is finite for each node n

Assuming the 3 conditions above, let p be a solution.

TO SHOW: A^* returns a solution with min cost c .

Let $F_0 = [\text{Start}]$, F_{n+1} be A^* 's next frontier after F_n ($[]$ if none), and c_n be the cost of the head of F_n (∞ if $F_n = []$).

(i) for every $n \geq 0$ s.t. $c_n < c$, F_n has a prefix of p

Admissibility

A^* is *admissible* (under cost, h) if it returns a solution of min cost whenever a solution exists.

3 conditions sufficient for admissibility

under-estimate: for every solution $n \cdots \text{goal}$,
 $0 \leq h(n) \leq \text{cost}(n \cdots \text{goal})$

termination: for some $\epsilon > 0$, every arc costs $\geq \epsilon$

finite branching: $\{n' \mid \text{arc}(n, n')\}$ is finite for each node n

Assuming the 3 conditions above, let p be a solution.

TO SHOW: A^* returns a solution with min cost c .

Let $F_0 = [\text{Start}]$, F_{n+1} be A^* 's next frontier after F_n ($[]$ if none), and c_n be the cost of the head of F_n (∞ if $F_n = []$).

- (i) for every $n \geq 0$ s.t. $c_n < c$, F_n has a prefix of p
- (ii) $c = c_n$ for some n s.t. the head of F_n is a solution.

Heuristic under-estimating cost tends to value the settled past
(tracked by cost) over an uncertain future (estimated by heuristic)

Heuristic under-estimating cost tends to value the settled past
(tracked by cost) over an uncertain future (estimated by heuristic)

A-star as a tweak of breadth-first:

tree with branching factor b , counted by geometric series

$$1 + b + b^2 + b^3 + \cdots + b^n = \sum_{k=0}^n b^k = \frac{1 - b^{n+1}}{1 - b}$$

$$\text{since } s_n + b^{n+1} = 1 + bs_n$$

Shifting perspectives

Heuristic under-estimating cost tends to value the settled past (tracked by cost) over an uncertain future (estimated by heuristic)

A-star as a tweak of breadth-first:

tree with branching factor b , counted by geometric series

$$1 + b + b^2 + b^3 + \cdots + b^n = \sum_{k=0}^n b^k = \frac{1 - b^{n+1}}{1 - b}$$

Shift:

- ▶ $b = 2 \rightsquigarrow b = 1/2$ (to discount future)

Shifting perspectives

Heuristic under-estimating cost tends to value the settled past (tracked by cost) over an uncertain future (estimated by heuristic)

A-star as a tweak of breadth-first:

tree with branching factor b , counted by geometric series

$$1 + b + b^2 + b^3 + \cdots + b^n = \sum_{k=0}^n b^k = \frac{1 - b^{n+1}}{1 - b}$$

Shift:

- ▶ $b = 2 \rightsquigarrow b = 1/2$ (to discount future)
- ▶ frontier search from start \rightsquigarrow back up from goal

Shifting perspectives

Heuristic under-estimating cost tends to value the settled past (tracked by cost) over an uncertain future (estimated by heuristic)

A-star as a tweak of breadth-first:

tree with branching factor b , counted by geometric series

$$1 + b + b^2 + b^3 + \cdots + b^n = \sum_{k=0}^n b^k = \frac{1 - b^{n+1}}{1 - b}$$

Shift:

- ▶ $b = 2 \rightsquigarrow b = 1/2$ (to discount future)
- ▶ frontier search from start \rightsquigarrow back up from goal
- ▶ arc \approx action, subject to a reward (contra cost)

Shifting perspectives

Heuristic under-estimating cost tends to value the settled past (tracked by cost) over an uncertain future (estimated by heuristic)

A-star as a tweak of breadth-first:

tree with branching factor b , counted by geometric series

$$1 + b + b^2 + b^3 + \cdots + b^n = \sum_{k=0}^n b^k = \frac{1 - b^{n+1}}{1 - b}$$

Shift:

- ▶ $b = 2 \rightsquigarrow b = 1/2$ (to discount future)
- ▶ frontier search from start \rightsquigarrow back up from goal
- ▶ arc \approx action, subject to a reward (contra cost)
- ▶ heuristic cost estimate h -to-minimize \rightsquigarrow reward H -to-maximize

Shifting perspectives: from searching to learning

Heuristic under-estimating cost tends to value the settled past (tracked by cost) over an uncertain future (estimated by heuristic)

A-star as a tweak of breadth-first:

tree with branching factor b , counted by geometric series

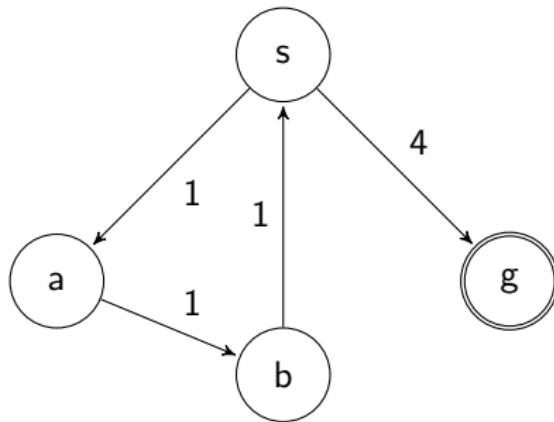
$$1 + b + b^2 + b^3 + \cdots + b^n = \sum_{k=0}^n b^k = \frac{1 - b^{n+1}}{1 - b}$$

Shift:

- ▶ $b = 2 \rightsquigarrow b = 1/2$ (to discount future)
- ▶ frontier search from start \rightsquigarrow back up from goal
- ▶ arc \approx action, subject to a reward (contra cost)
- ▶ heuristic cost estimate h -to-minimize \rightsquigarrow reward H -to-maximize
approximate $H = \lim_{n \rightarrow \infty} H_n$ based on look-ahead n
— **learning** in stages

$$Q(s, s') := Q_0(s, s') + \frac{1}{2} \max\{Q(s', s'') \mid arc_=(s', s'')\}$$

$$Q_0(s, s') := \begin{cases} 1 & \text{if } s = s' \in G \\ -\text{cost}(s, s') & \text{else if } arc(s, s') \end{cases}$$



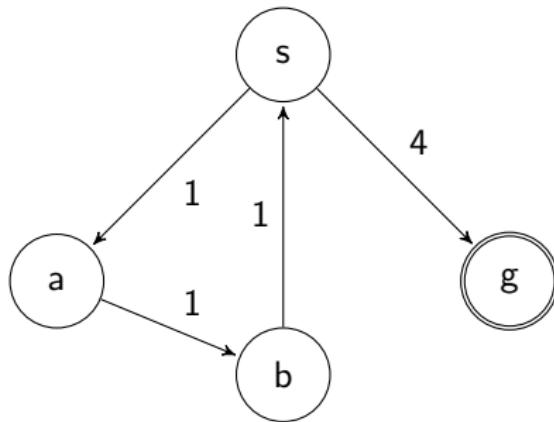
$$Q(s, g) = -3$$

$$Q(s, a) = -2$$

$$= Q(a, b) = Q(b, s)$$

$$Q(s, s') := Q_0(s, s') + \frac{1}{2} \max\{Q(s', s'') \mid arc_=(s', s'')\}$$

$$Q_0(s, s') := \begin{cases} 1 & \text{if } s = s' \in G \\ -\text{cost}(s, s') & \text{else if } arc(s, s') \end{cases}$$



$$\begin{aligned} Q(g, g) &= 1 + \frac{1}{2} Q(g, g) \\ \therefore Q(g, g) &= 2 \end{aligned}$$

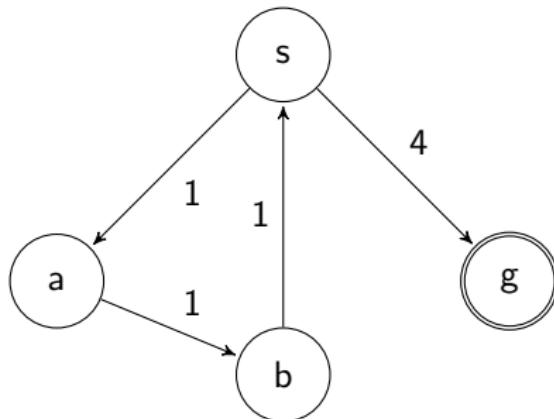
$$Q(s, g) = -3$$

$$Q(s, a) = -2$$

$$= Q(a, b) = Q(b, s)$$

$$Q(s, s') := Q_0(s, s') + \frac{1}{2} \max\{Q(s', s'') \mid arc_=(s', s'')\}$$

$$Q_0(s, s') := \begin{cases} 1 & \text{if } s = s' \in G \\ -\text{cost}(s, s') & \text{else if } arc(s, s') \end{cases}$$



$$Q(g,g) = 1 + \frac{1}{2}Q(g,g)$$

$$\therefore Q(g,g) = 2$$

$$Q(s,g) = -4 + \frac{1}{2}Q(g,g) = -3$$

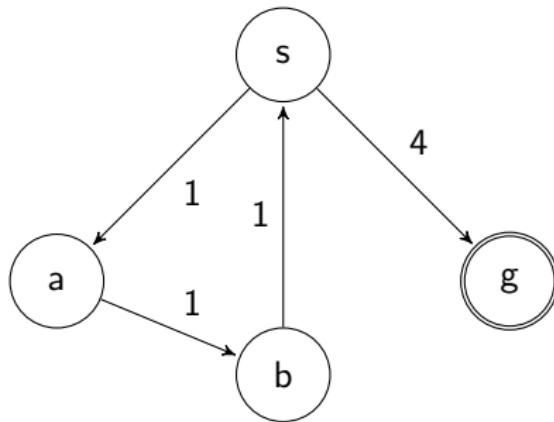
$$Q(s,g) = -3$$

$$Q(s,a) = -2$$

$$= Q(a,b) = Q(b,s)$$

$$Q(s, s') := Q_0(s, s') + \frac{1}{2} \max\{Q(s', s'') \mid arc_=(s', s'')\}$$

$$Q_0(s, s') := \begin{cases} 1 & \text{if } s = s' \in G \\ -\text{cost}(s, s') & \text{else if } arc(s, s') \end{cases}$$



$$\begin{aligned} Q(g, g) &= 1 + \frac{1}{2}Q(g, g) \\ \therefore Q(g, g) &= 2 \end{aligned}$$

$$Q(s, g) = -4 + \frac{1}{2}Q(g, g) = -3$$

$$\begin{aligned} Q(s, a) &= -1 + \frac{1}{2}Q(a, b) \\ Q(a, b) &= -1 + \frac{1}{2}Q(b, s) \\ Q(b, s) &= -1 + \frac{1}{2}\max\{Q(s, a), -3\} \end{aligned}$$

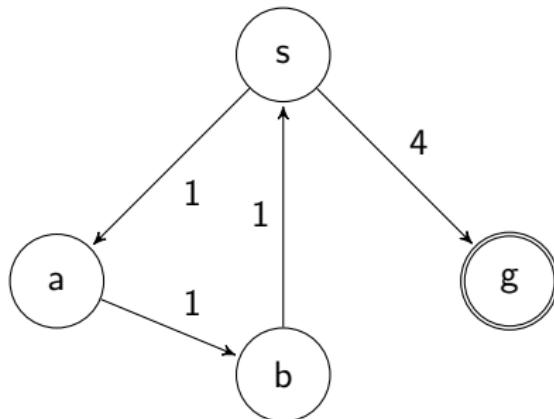
$$Q(s, g) = -3$$

$$Q(s, a) = -2$$

$$= Q(a, b) = Q(b, s)$$

$$Q(s, s') := Q_0(s, s') + \frac{1}{2} \max\{Q(s', s'') \mid arc_=(s', s'')\}$$

$$Q_0(s, s') := \begin{cases} 1 & \text{if } s = s' \in G \\ -\text{cost}(s, s') & \text{else if } arc(s, s') \end{cases}$$



$$\begin{aligned} Q(g,g) &= 1 + \frac{1}{2}Q(g,g) \\ \therefore Q(g,g) &= 2 \end{aligned}$$

$$Q(s,g) = -4 + \frac{1}{2}Q(g,g) = -3$$

$$\begin{aligned} Q(s,a) &= -1 + \frac{1}{2}Q(a,b) \\ Q(a,b) &= -1 + \frac{1}{2}Q(b,s) \\ Q(b,s) &= -1 + \frac{1}{2}\max\{Q(s,a), -3\} \end{aligned}$$

$$Q(s,g) = -3$$

$$Q(s,a) = -2$$

$$= Q(a,b) = Q(b,s)$$

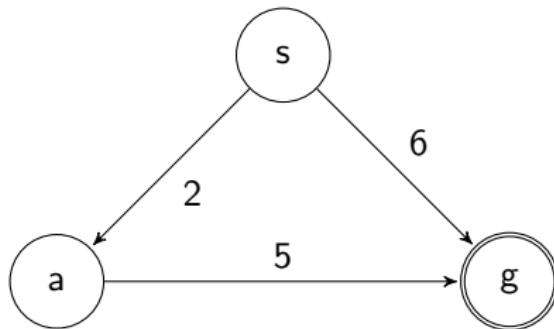
$$\text{Case 1: } Q(s,a) \geq -3$$

$$\text{Case 2: } Q(s,a) < -3$$

contradiction

$$Q(s, s') := Q_0(s, s') + \frac{1}{2} \max\{Q(s', s'') \mid arc_=(s', s'')\}$$

$$Q_0(s, s') := \begin{cases} 1 & \text{if } s = s' \in G \\ -\text{cost}(s, s') & \text{else if } arc(s, s') \end{cases}$$



$$Q(s, g) = -5$$

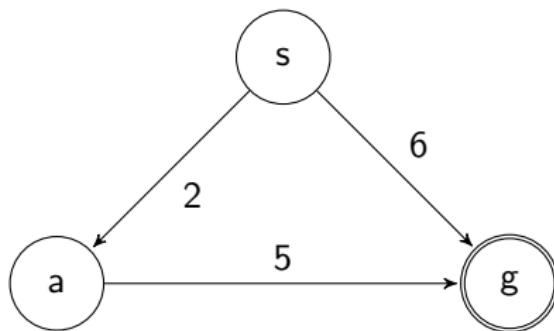
$$Q(a, g) = -4$$

$$= Q(s, a)$$

$$Q(s, s') := Q_0(s, s') + \frac{1}{2} \max\{Q(s', s'') \mid arc_=(s', s'')\}$$

$$Q_0(s, s') := \begin{cases} 1 & \text{if } s = s' \in G \\ -\text{cost}(s, s') & \text{else if } arc(s, s') \end{cases}$$

$$\begin{aligned} Q(g, g) &= 1 + \frac{1}{2}Q(g, g) \\ \therefore Q(g, g) &= 2 \end{aligned}$$



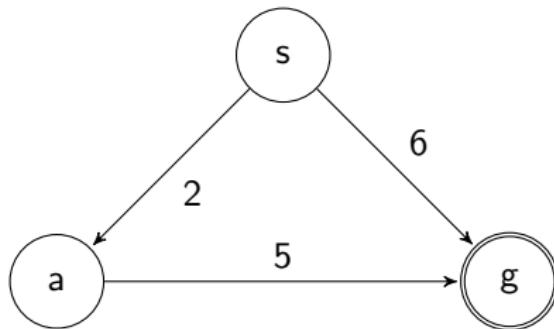
$$Q(s, g) = -5$$

$$Q(a, g) = -4$$

$$= Q(s, a)$$

$$Q(s, s') := Q_0(s, s') + \frac{1}{2} \max\{Q(s', s'') \mid \text{arc}_=(s', s'')\}$$

$$Q_0(s, s') := \begin{cases} 1 & \text{if } s = s' \in G \\ -\text{cost}(s, s') & \text{else if } \text{arc}(s, s') \end{cases}$$



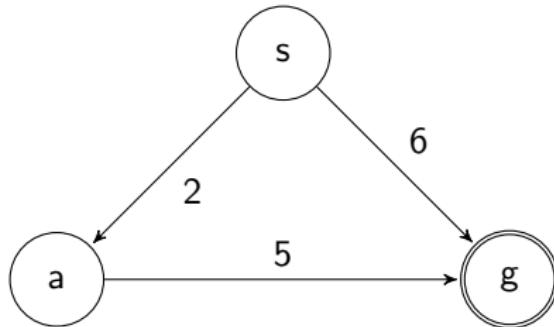
$$\begin{aligned} Q(g, g) &= 1 + \frac{1}{2}Q(g, g) \\ \therefore Q(g, g) &= 2 \end{aligned}$$

$$Q(s, g) = -6 + \frac{1}{2}Q(g, g) = -5$$

$$\begin{aligned} Q(s, g) &= -5 \\ Q(a, g) &= -4 \\ &= Q(s, a) \end{aligned}$$

$$Q(s, s') := Q_0(s, s') + \frac{1}{2} \max\{Q(s', s'') \mid arc_=(s', s'')\}$$

$$Q_0(s, s') := \begin{cases} 1 & \text{if } s = s' \in G \\ -\text{cost}(s, s') & \text{else if } arc(s, s') \end{cases}$$



$$Q(s, g) = -5$$

$$Q(a, g) = -4$$

$$= Q(s, a)$$

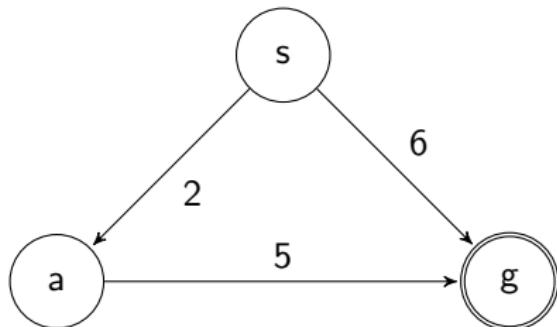
$$\begin{aligned} Q(g, g) &= 1 + \frac{1}{2}Q(g, g) \\ \therefore Q(g, g) &= 2 \end{aligned}$$

$$Q(s, g) = -6 + \frac{1}{2}Q(g, g) = -5$$

$$\begin{aligned} Q(s, a) &= -2 + \frac{1}{2}Q(a, g) \\ Q(a, g) &= -5 + \frac{1}{2}Q(g, g) \end{aligned}$$

$$Q(s, s') := Q_0(s, s') + \frac{1}{2} \max\{Q(s', s'') \mid \text{arc}_=(s', s'')\}$$

$$Q_0(s, s') := \begin{cases} 1 & \text{if } s = s' \in G \\ -\text{cost}(s, s') & \text{else if } \text{arc}(s, s') \end{cases}$$



$$\begin{aligned} Q(s, g) &= -5 \\ Q(a, g) &= -4 \\ &= Q(s, a) \end{aligned}$$

$$\begin{aligned} Q(g, g) &= 1 + \frac{1}{2}Q(g, g) \\ \therefore Q(g, g) &= 2 \end{aligned}$$

$$Q(s, g) = -6 + \frac{1}{2}Q(g, g) = -5$$

$$\begin{aligned} Q(s, a) &= -2 + \frac{1}{2}Q(a, g) \\ Q(a, g) &= -5 + \frac{1}{2}Q(g, g) \end{aligned}$$

Compare to iterates

$$Q_{n+1}(s, s') := Q_0(s, s') + \frac{1}{2} \max\{Q_n(s', s'') \mid \text{arc}_=(s', s'')\}$$

Given a

specification R of immediate rewards after particular actions

calculate the return Q of particular actions over time via

$$Q = \lim_{n \rightarrow \infty} Q_n$$

A generalisation

Given a

specification R of immediate rewards after particular actions

calculate the return Q of particular actions over time via

$$Q = \lim_{n \rightarrow \infty} Q_n$$

$$Q_{n+1}(s, s') := R(s, s') + \frac{1}{2} \max\{Q_n(s', s'') \mid arc_=(s', s'')\} \quad (1)$$

\rightsquigarrow

$$\begin{aligned} Q_{n+1}(s, a) &\approx \alpha [R(s, a) + \gamma \max\{Q_n(\textcolor{red}{s}', a') \mid a' \in A\}] \\ &+ (1 - \alpha) Q_n(s, a) \end{aligned} \quad (2)$$

A generalisation

Given a

specification R of immediate rewards after particular actions

calculate the return Q of particular actions over time via

$$Q = \lim_{n \rightarrow \infty} Q_n$$

$$Q_{n+1}(s, s') := R(s, s') + \frac{1}{2} \max\{Q_n(s', s'') \mid arc_=(s', s'')\} \quad (1)$$

\leadsto

$$\begin{aligned} Q_{n+1}(s, a) &\approx \alpha [R(s, a) + \gamma \max\{Q_n(\textcolor{red}{s'}, a') \mid a' \in A\}] \\ &\quad + (1 - \alpha) Q_n(s, a) \end{aligned} \quad (2)$$

(1)	(2)
s'	a
1	α
$\frac{1}{2}$	γ

(1) is (2) with action a resulting in s'
deterministically for $\alpha = 1$, with $\gamma = \frac{1}{2}$

s' is learned from experience (environment)

Markov decision process (MDP)

a 5-tuple $\langle S, A, p, r, \gamma \rangle$ consisting of

- ▶ a finite set S of states s, s', \dots
- ▶ a finite set A of actions a, \dots
- ▶ a function $p : S \times A \times S \rightarrow [0, 1]$

$p(s, a, s') = \text{prob}(s' | s, a) = \text{how probable is } s' \text{ after doing } a \text{ at } s$

$$\sum_{s'} p(s, a, s') = 1 \text{ for all } a \in A, s \in S$$

Markov decision process (MDP)

a 5-tuple $\langle S, A, p, r, \gamma \rangle$ consisting of

- ▶ a finite set S of states s, s', \dots
- ▶ a finite set A of actions a, \dots
- ▶ a function $p : S \times A \times S \rightarrow [0, 1]$

$p(s, a, s') = \text{prob}(s' | s, a) = \text{how probable is } s' \text{ after doing } a \text{ at } s$

$$\sum_{s'} p(s, a, s') = 1 \text{ for all } a \in A, s \in S$$

- ▶ a function $r : S \times A \times S \rightarrow \mathbb{R}$

$r(s, a, s') = \text{immediate reward at } s' \text{ after } a \text{ is done at } s$

Markov decision process (MDP)

a 5-tuple $\langle S, A, p, r, \gamma \rangle$ consisting of

- ▶ a finite set S of states s, s', \dots
- ▶ a finite set A of actions a, \dots
- ▶ a function $p : S \times A \times S \rightarrow [0, 1]$

$p(s, a, s') = \text{prob}(s' | s, a) = \text{how probable is } s' \text{ after doing } a \text{ at } s$

$$\sum_{s'} p(s, a, s') = 1 \text{ for all } a \in A, s \in S$$

- ▶ a function $r : S \times A \times S \rightarrow \mathbb{R}$

$r(s, a, s') = \text{immediate reward at } s' \text{ after } a \text{ is done at } s$

- ▶ a discount factor $\gamma \in [0, 1]$

Markov decision process (MDP)

a 5-tuple $\langle S, A, p, r, \gamma \rangle$ consisting of

- ▶ a finite set S of states s, s', \dots
- ▶ a finite set A of actions a, \dots
- ▶ a function $p : S \times A \times S \rightarrow [0, 1]$

$p(s, a, s') = \text{prob}(s' | s, a) = \text{how probable is } s' \text{ after doing } a \text{ at } s$

$$\sum_{s'} p(s, a, s') = 1 \text{ for all } a \in A, s \in S$$

- ▶ a function $r : S \times A \times S \rightarrow \mathbb{R}$

$r(s, a, s') = \text{immediate reward at } s' \text{ after } a \text{ is done at } s$

- ▶ a discount factor $\gamma \in [0, 1]$

Missing: policy $\pi : S \rightarrow A$ (what to do at s)

Exercise (Poole & Mackworth, chap 9)

Sam is either fit or unfit

$$S = \{\text{fit, unfit}\}$$

and has to decide whether to exercise or relax

$$A = \{\text{exercise, relax}\}.$$

Exercise (Poole & Mackworth, chap 9)

Sam is either fit or unfit

$$S = \{\text{fit, unfit}\}$$

and has to decide whether to exercise or relax

$$A = \{\text{exercise, relax}\}.$$

$p(s, a, s')$ and $r(s, a, s')$ are a -table entries for row s , col s'

exercise	fit	unfit	relax	fit	unfit
fit	.99, 8		fit	.7, 10	
unfit	.2, 0		unfit	0, 5	

immediate rewards do not
depend on the resulting state

Exercise (Poole & Mackworth, chap 9)

Sam is either fit or unfit

$$S = \{\text{fit, unfit}\}$$

and has to decide whether to exercise or relax

$$A = \{\text{exercise, relax}\}.$$

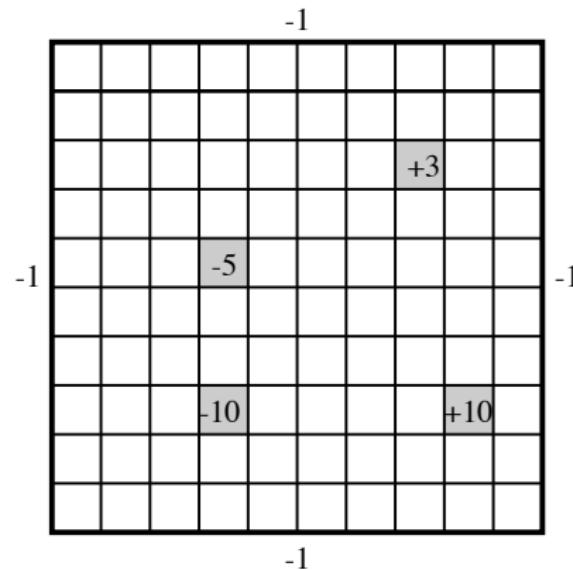
$p(s, a, s')$ and $r(s, a, s')$ are a -table entries for row s , col s'

exercise	fit	unfit	relax	fit	unfit
fit	.99, 8	.01, 8	fit	.7, 10	.3, 10
unfit	.2, 0	.8, 0	unfit	0, 5	1, 5

Entries in red follow from assuming immediate rewards do not depend on the resulting state, and

$$\sum_{s'} p(s, a, s') = 1$$

Grid World



states: 100 positions

actions: up, down, left, right

punish -1 when banging into wall

& 4 reward/punish states

prob: 0.7 as directed (if possible)

...

Policy from an MDP

Given state s , pick action a that maximizes return

$$Q(s, a) := \sum_{s'} \overbrace{p(s, a, s')}^{\text{different outcomes } s'} \left(\underbrace{r(s, a, s')}_{\text{immediate}} + \gamma \overbrace{V(s')}^{\text{discounted future}} \right)$$

for V tied back to Q via policy $\pi : S \rightarrow A$

$$V_\pi(s) := Q(s, \pi(s))$$

Policy from an MDP

Given state s , pick action a that maximizes return

$$Q(s, a) := \sum_{s'} \overbrace{p(s, a, s')}^{\text{different outcomes } s'} \left(\underbrace{r(s, a, s')}_{\text{immediate}} + \gamma \overbrace{V(s')}^{\text{discounted future}} \right)$$

for V tied back to Q via policy $\pi : S \rightarrow A$

$$V_\pi(s) := Q(s, \pi(s))$$

e.g., the greedy Q -policy above

$$\pi(s) := \arg \max_a Q(s, a)$$

for

$$Q(s, a) = \sum_{s'} p(s, a, s') (r(s, a, s') + \gamma \max_{a'} Q(s', a'))$$

Value iteration

Mutual recursion between Q/V and π
value of an action/state vs what to do at a state

Value iteration

Mutual recursion between Q/V and π

value of an action/state vs what to do at a state

Focus on Q , approached in the limit

$$\lim_{n \rightarrow \infty} q_n$$

from iterates

$$q_0(s, a) := \sum_{s'} p(s, a, s') r(s, a, s')$$

$$q_{n+1}(s, a) := \sum_{s'} p(s, a, s') (r(s, a, s') + \gamma \max_{a'} q_n(s', a'))$$

Value iteration

Mutual recursion between Q/V and π

value of an action/state vs what to do at a state

Focus on Q , approached in the limit

$$\lim_{n \rightarrow \infty} q_n$$

from iterates

$$q_0(s, a) := \sum_{s'} p(s, a, s') r(s, a, s')$$

$$q_{n+1}(s, a) := \sum_{s'} p(s, a, s') (r(s, a, s') + \gamma \max_{a'} q_n(s', a'))$$

In case $p(s, a, s') = 1$ for some s' (necessarily unique),
the iterates simplify to

$$q_0(s, a) := r(s, a, s')$$

$$q_{n+1}(s, a) := r(s, a, s') + \gamma \max_{a'} q_n(s', a')$$

Deterministic actions and absorbing states (game over)

Fix an MDP with min reward m .

An action a is *s-deterministic* if $p(s, a, s') = 1$ for some s' .

Deterministic actions and absorbing states (game over)

Fix an MDP with min reward m .

An action a is *s-deterministic* if $p(s, a, s') = 1$ for some s' .

A state s is *absorbing* if $p(s, a, s) = 1$ for every action a , whence

$$Q(s, a) = r(s, a, s) + \gamma V(s)$$

$$V(s) = \frac{r_s}{1 - \gamma} \text{ where } r_s = \max_a r(s, a, s)$$

A state s is a *sink* if it is absorbing and $r(s, a, s) = m$ for all a .

Deterministic actions and absorbing states (game over)

Fix an MDP with min reward m .

An action a is *s-deterministic* if $p(s, a, s') = 1$ for some s' .

A state s is *absorbing* if $p(s, a, s) = 1$ for every action a , whence

$$Q(s, a) = r(s, a, s) + \gamma V(s)$$

$$V(s) = \frac{r_s}{1 - \gamma} \text{ where } r_s = \max_a r(s, a, s)$$

A state s is a *sink* if it is absorbing and $r(s, a, s) = m$ for all a .

An action a is an *s-drain* if for some sink s' ,

$$p(s, a, s') = 1 \text{ and } r(s, a, s') = m$$

Deterministic actions and absorbing states (game over)

Fix an MDP with min reward m .

An action a is *s-deterministic* if $p(s, a, s') = 1$ for some s' .

A state s is *absorbing* if $p(s, a, s) = 1$ for every action a , whence

$$Q(s, a) = r(s, a, s) + \gamma V(s)$$

$$V(s) = \frac{r_s}{1 - \gamma} \text{ where } r_s = \max_a r(s, a, s)$$

A state s is a *sink* if it is absorbing and $r(s, a, s) = m$ for all a .

An action a is an *s-drain* if for some sink s' ,

$$p(s, a, s') = 1 \text{ and } r(s, a, s') = m$$

Let

$$A(s) := \{a \in A \mid a \text{ is not an } s\text{-drain}\}$$

so if $A(s) \neq \emptyset$,

$$V(s) = \max\{Q(s, a) \mid a \in A\} = \max\{Q(s, a) \mid a \in A(s)\}$$

Arcs & goals as a deterministic MDP ($p \in \{0, 1\}$)

Given *arc* and goal set G , let

$$A = \{s \mid (\exists s') \text{ arc}_=(s', s)\} = S$$

where for each $a \in A$,

$$p(s, a, s') = \begin{cases} 1 & \text{if } a = s' \text{ and } \text{arc}_=(s, s') \\ 0 & \text{otherwise} \end{cases}$$

$$r(s, a, s') = \begin{cases} R(s, s') & \text{if } a = s' \text{ and } \text{arc}_=(s, s') \\ \text{anything} & \text{otherwise} \end{cases}$$

Arcs & goals as a deterministic MDP ($p \in \{0, 1\}$)

Given arc and goal set G , let

$$A = \{s \mid (\exists s') \text{ } \text{arc}_=(s', s)\} = S$$

where for each $a \in A$,

$$p(s, a, s') = \begin{cases} 1 & \text{if } a = s' \text{ and } \text{arc}_=(s, s') \\ 0 & \text{otherwise} \end{cases}$$

$$r(s, a, s') = \begin{cases} R(s, s') & \text{if } a = s' \text{ and } \text{arc}_=(s, s') \\ \text{anything} & \text{otherwise} \end{cases}$$

Satisfy prob constraint $\sum_{s'} p(s, a, s') = 1$ via sink state $\perp \notin A$, requiring of every $a \in A$ and $s \in S$,

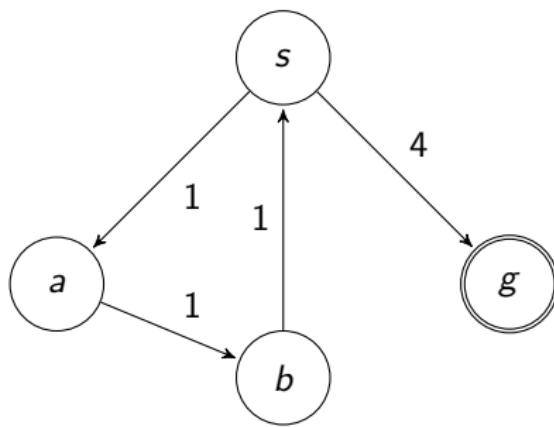
$$p(s, a, \perp) = \begin{cases} 1 & \text{if not } \text{arc}_=(s, a) \\ 0 & \text{otherwise} \end{cases}$$

$$p(\perp, a, s) = \begin{cases} 1 & \text{if } s = \perp \\ 0 & \text{otherwise} \end{cases}$$

$$r(s, a, \perp) = \min \text{ reward}$$

$$Q_{n+1}(x, y) := Q_0(x, y) + \frac{1}{2} \max\{Q_n(y, z) \mid \text{arc}_=(y, z)\}$$

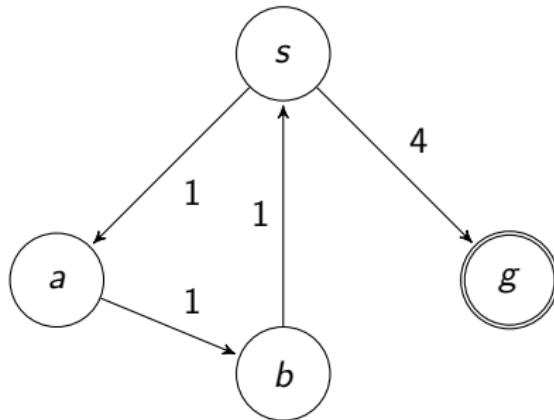
$$Q_0(x, y) := \begin{cases} 4 & \text{if } x = y \in G \\ -\text{cost}(x, y) & \text{else if } \text{arc}(x, y) \end{cases}$$



$$Q_{n+1}(x, y) := Q_0(x, y) + \frac{1}{2} \max\{Q_n(y, z) \mid \text{arc}_=(y, z)\}$$

$$Q_0(x, y) := \begin{cases} 4 & \text{if } x = y \in G \\ -\text{cost}(x, y) & \text{else if } \text{arc}(x, y) \end{cases}$$

$$Q_0(s, a) = -1 > Q_0(s, g) = -4$$

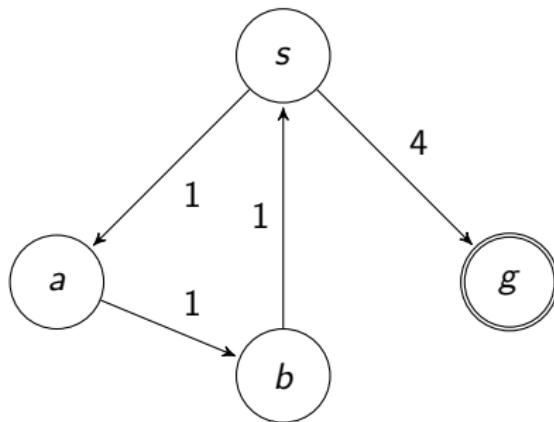


From s , move to a or to g ?

$n = 0$: choose a

$$Q_{n+1}(x, y) := Q_0(x, y) + \frac{1}{2} \max\{Q_n(y, z) \mid \text{arc}_=(y, z)\}$$

$$Q_0(x, y) := \begin{cases} 4 & \text{if } x = y \in G \\ -\text{cost}(x, y) & \text{else if } \text{arc}(x, y) \end{cases}$$



$$Q_0(s, a) = -1 > Q_0(s, g) = -4$$

$$\begin{aligned} Q_1(s, a) &= -1 + \frac{1}{2} Q_0(a, b) = -\frac{3}{2} \\ Q_1(s, g) &= -4 + \frac{1}{2} Q_0(g, g) = -2 \end{aligned}$$

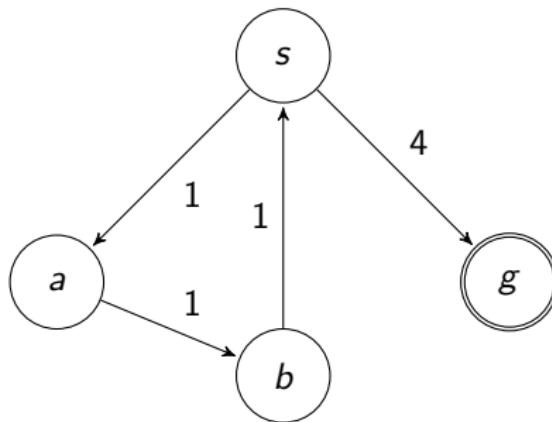
From s , move to a or to g ?

$n = 0$: choose a

$n = 1$: choose a

$$Q_{n+1}(x, y) := Q_0(x, y) + \frac{1}{2} \max\{Q_n(y, z) \mid \text{arc}_=(y, z)\}$$

$$Q_0(x, y) := \begin{cases} 4 & \text{if } x = y \in G \\ -\text{cost}(x, y) & \text{else if } \text{arc}(x, y) \end{cases}$$



From s , move to a or to g ?

$n = 0$: choose a

$n = 1$: choose a

$n = 2$: choose g

$$Q_0(s, a) = -1 > Q_0(s, g) = -4$$

$$\begin{aligned} Q_1(s, a) &= -1 + \frac{1}{2} Q_0(a, b) = -\frac{3}{2} \\ Q_1(s, g) &= -4 + \frac{1}{2} Q_0(g, g) = -2 \end{aligned}$$

$$\begin{aligned} Q_1(a, b) &= -1 + \frac{1}{2} Q_0(b, s) = -\frac{3}{2} \\ Q_1(g, g) &= 4 + \frac{1}{2} Q_0(g, g) = 6 \end{aligned}$$

$$\begin{aligned} Q_2(s, a) &= -1 + \frac{1}{2} Q_1(a, b) = -\frac{7}{4} \\ Q_2(s, g) &= -4 + \frac{1}{2} Q_1(g, g) = -1 \end{aligned}$$

Exercise or relax, for $\gamma = 0.9$

Recall (probability, reward)-matrices for exercise, relax

exercise	fit	unfit	relax	fit	unfit
fit	.99, 8	.01, 8	fit	.7, 10	.3, 10
unfit	.2, 0	.8, 0	unfit	0, 5	1, 5

Exercise or relax, for $\gamma = 0.9$

Recall (probability, reward)-matrices for exercise, relax

exercise	fit	unfit	relax	fit	unfit
fit	.99, 8	.01, 8	fit	.7, 10	.3, 10
unfit	.2, 0	.8, 0	unfit	0, 5	1, 5

$$q_0(s, a) := p(s, a, \text{fit})r(s, a, \text{fit}) + p(s, a, \text{unfit})r(s, a, \text{unfit})$$

$$V_n(s) := \max(q_n(s, \text{exercise}), q_n(s, \text{relax}))$$

$$q_{n+1}(s, a) := q_0(s, a) + .9(p(s, a, \text{fit})V_n(\text{fit}) + p(s, a, \text{unfit})V_n(\text{unfit}))$$

Exercise or relax, for $\gamma = 0.9$

Recall (probability, reward)-matrices for exercise, relax

exercise	fit	unfit	relax	fit	unfit
fit	.99, 8	.01, 8	fit	.7, 10	.3, 10
unfit	.2, 0	.8, 0	unfit	0, 5	1, 5

$$q_0(s, a) := p(s, a, \text{fit})r(s, a, \text{fit}) + p(s, a, \text{unfit})r(s, a, \text{unfit})$$

$$V_n(s) := \max(q_n(s, \text{exercise}), q_n(s, \text{relax}))$$

$$q_{n+1}(s, a) := q_0(s, a) + .9(p(s, a, \text{fit})V_n(\text{fit}) + p(s, a, \text{unfit})V_n(\text{unfit}))$$

	exercise	relax	π
fit	8	10	relax
unfit	0	5	relax

Exercise or relax, for $\gamma = 0.9$

Recall (probability, reward)-matrices for exercise, relax

exercise	fit	unfit	relax	fit	unfit
fit	.99, 8	.01, 8	fit	.7, 10	.3, 10
unfit	.2, 0	.8, 0	unfit	0, 5	1, 5

$$q_0(s, a) := p(s, a, \text{fit})r(s, a, \text{fit}) + p(s, a, \text{unfit})r(s, a, \text{unfit})$$

$$V_n(s) := \max(q_n(s, \text{exercise}), q_n(s, \text{relax}))$$

$$q_{n+1}(s, a) := q_0(s, a) + .9(p(s, a, \text{fit})V_n(\text{fit}) + p(s, a, \text{unfit})V_n(\text{unfit}))$$

	exercise	relax	π
fit	8, 16.955	10, 17.65	relax, relax
unfit	0, 5.4	5, 9.5	relax, relax

Exercise or relax, for $\gamma = 0.9$

Recall (probability, reward)-matrices for exercise, relax

exercise	fit	unfit	relax	fit	unfit
fit	.99, 8	.01, 8	fit	.7, 10	.3, 10
unfit	.2, 0	.8, 0	unfit	0, 5	1, 5

$$q_0(s, a) := p(s, a, \text{fit})r(s, a, \text{fit}) + p(s, a, \text{unfit})r(s, a, \text{unfit})$$

$$V_n(s) := \max(q_n(s, \text{exercise}), q_n(s, \text{relax}))$$

$$q_{n+1}(s, a) := q_0(s, a) + .9(p(s, a, \text{fit})V_n(\text{fit}) + p(s, a, \text{unfit})V_n(\text{unfit}))$$

	exercise	relax	π
fit	8, 16.955, 23.812	10, 17.65, 23.685	relax, relax, exercise
unfit	0, 5.4, 10.017	5, 9.5, 13.55	relax, relax, relax

Temporal difference (TD)

A sequence of values

$$v_1, v_2, v_3, \dots$$

averages at time k to

$$A_k := \frac{v_1 + \dots + v_k}{k}$$

Temporal difference (TD)

A sequence of values

$$v_1, v_2, v_3, \dots$$

averages at time k to

$$A_k := \frac{v_1 + \dots + v_k}{k}$$

which learning v_{k+1} updates to

$$\begin{aligned} A_{k+1} &= \frac{v_1 + \dots + v_k + v_{k+1}}{k+1} \\ &= \frac{k}{k+1} A_k + \frac{1}{k+1} v_{k+1} \end{aligned}$$

Temporal difference (TD)

A sequence of values

$$v_1, v_2, v_3, \dots$$

averages at time k to

$$A_k := \frac{v_1 + \dots + v_k}{k}$$

which learning v_{k+1} updates to

$$\begin{aligned} A_{k+1} &= \frac{v_1 + \dots + v_k + v_{k+1}}{k+1} \\ &= \frac{k}{k+1} A_k + \frac{1}{k+1} v_{k+1} \end{aligned}$$

so if $\alpha_k = \frac{1}{k}$,

new

$$\begin{aligned} A_{k+1} &= (1 - \alpha_{k+1}) A_k + \alpha_{k+1} \overbrace{v_{k+1}}^{\text{new}} \\ &= \underbrace{A_k}_{\text{old}} + \alpha_{k+1} (\underbrace{v_{k+1} - A_k}_{\text{temp diff: new-old}}) \end{aligned}$$

Q-Learning

Assume v_{k+1} is derived from r_{k+1}, s_{k+1} , observed sequentially

$$s_1 \xrightarrow{a_1} r_2, s_2 \xrightarrow{a_2} r_3, s_3 \xrightarrow{a_3} \cdots \underbrace{s_k \xrightarrow{a_k} r_{k+1}, s_{k+1}}_{\text{experience}} \xrightarrow{a_{k+1}} \cdots$$

experience from which we learn

$$v_{k+1} := r_{k+1} + \gamma \max_a Q_k(s_{k+1}, a)$$

Q-Learning

Assume v_{k+1} is derived from r_{k+1}, s_{k+1} , observed sequentially

$$s_1 \xrightarrow{a_1} r_2, s_2 \xrightarrow{a_2} r_3, s_3 \xrightarrow{a_3} \cdots \underbrace{s_k \xrightarrow{a_k} r_{k+1}, s_{k+1}}_{\text{experience from which we learn}} \xrightarrow{a_{k+1}} \cdots$$

experience from which we learn

$$v_{k+1} := r_{k+1} + \gamma \max_a Q_k(s_{k+1}, a)$$

given $0 \leq \gamma < 1$, $Q_1 : (S \times A) \rightarrow \mathbb{R}$ and $v_1 \in \mathbb{R}$, with

$$Q_{k+1}(s_k, a_k) := (1 - \alpha)Q_k(s_k, a_k) + \alpha v_{k+1}$$

for $0 \leq \alpha \leq 1$,

Q-Learning

Assume v_{k+1} is derived from r_{k+1}, s_{k+1} , observed sequentially

$$s_1 \xrightarrow{a_1} r_2, s_2 \xrightarrow{a_2} r_3, s_3 \xrightarrow{a_3} \cdots \underbrace{s_k \xrightarrow{a_k} r_{k+1}, s_{k+1}}_{\text{experience from which we learn}} \xrightarrow{a_{k+1}} \cdots$$

experience from which we learn

$$v_{k+1} := r_{k+1} + \gamma \max_a Q_k(s_{k+1}, a)$$

given $0 \leq \gamma < 1$, $Q_1 : (S \times A) \rightarrow \mathbb{R}$ and $v_1 \in \mathbb{R}$, with

$$Q_{k+1}(s_k, a_k) := (1 - \alpha)Q_k(s_k, a_k) + \alpha v_{k+1}$$

for $0 \leq \alpha \leq 1$, smelling like

$$A_{k+1} = (1 - \alpha_{k+1})A_k + \alpha_{k+1}v_{k+1} \text{ for } \alpha_{k+1} = \frac{1}{k+1}$$

from previous slide (on TD).

Averaging?

$$\begin{aligned} v_{k+1} &= r_{k+1} + \gamma \max_a Q_k(s_{k+1}, a) \\ \underbrace{Q_{k+1}(s_k, a_k)}_{A_{k+1}} &= (1 - \alpha) \underbrace{Q_k(s_k, a_k)}_{\neq Q_k(s_{k-1}, a_{k-1})} + \alpha v_{k+1} \\ &\neq Q_k(s_{k-1}, a_{k-1}) = A_k \end{aligned}$$

Averaging?

$$\begin{aligned} v_{k+1} &= r_{k+1} + \gamma \max_a Q_k(s_{k+1}, a) \\ \underbrace{Q_{k+1}(s_k, a_k)}_{A_{k+1}} &= (1 - \alpha) \underbrace{Q_k(s_k, a_k)}_{\neq Q_k(s_{k-1}, a_{k-1})} + \alpha v_{k+1} \\ &\neq Q_k(s_{k-1}, a_{k-1}) = A_k \end{aligned}$$

for a deterministic MDP

i.e., $p(s, a, s') \in \{0, 1\}$ for all s, a, s'

let $\alpha = 1$ as v_{k+1} may look-ahead further than Q_k for the experience $s_k, a_k, r_{k+1}, s_{k+1}$ (determined by s_k, a_k)

Averaging?

$$\begin{aligned} v_{k+1} &= r_{k+1} + \gamma \max_a Q_k(s_{k+1}, a) \\ \underbrace{Q_{k+1}(s_k, a_k)}_{A_{k+1}} &= (1 - \alpha) \underbrace{Q_k(s_k, a_k)}_{\neq Q_k(s_{k-1}, a_{k-1}) = A_k} + \alpha v_{k+1} \end{aligned}$$

for a deterministic MDP

i.e., $p(s, a, s') \in \{0, 1\}$ for all s, a, s'

let $\alpha = 1$ as v_{k+1} may look-ahead further than Q_k for the experience $s_k, a_k, r_{k+1}, s_{k+1}$ (determined by s_k, a_k)

for $0 < p(s, a, s') < 1$, sample s' at frequency $\propto p(s, a, s')$ to average Q as a whole (not just $Q(s, a)$ at a particular (s, a)), converging to optimal Q under certain assumptions, including

$$\sum \alpha_k = \infty \quad \text{and} \quad \sum \alpha_k^2 < \infty \quad (\text{e.g. } \alpha_k = \frac{1}{k})$$

MDP, one experience at a time

Update $q : (S \times A) \rightarrow \mathbb{R}$ via p, r for

$$q'(s, a) := \sum_{s'} p(s, a, s') (r(s, a, s') + \gamma \max_{a'} q(s', a'))$$

or pointwise via experience $s_1 \xrightarrow{a_1} r_2, s_2$ for

$$q'(s, a) := \begin{cases} \alpha(r_2 + \gamma \max_{a'} q(s_2, a')) \\ \quad + (1 - \alpha)q(s, a) & \text{if } s = s_1 \text{ and } a = a_1 \\ q(s, a) & \text{otherwise.} \end{cases}$$

MDP, one experience at a time

Update $q : (S \times A) \rightarrow \mathbb{R}$ via p, r for

$$q'(s, a) := \sum_{s'} p(s, a, s') (r(s, a, s') + \gamma \max_{a'} q(s', a'))$$

or pointwise via experience $s_1 \xrightarrow{a_1} r_2, s_2$ for

$$q'(s, a) := \begin{cases} \alpha(r_2 + \gamma \max_{a'} q(s_2, a')) \\ \quad + (1 - \alpha)q(s, a) & \text{if } s = s_1 \text{ and } a = a_1 \\ q(s, a) & \text{otherwise.} \end{cases}$$

To converge to MDP's optimal Q -value, visit every state-action pair (s, a) repeatedly (for $s \xrightarrow{a} r', s'$ with diff s', r' under p, r).

MDP, one experience at a time

Update $q : (S \times A) \rightarrow \mathbb{R}$ via p, r for

$$q'(s, a) := \sum_{s'} p(s, a, s') (r(s, a, s') + \gamma \max_{a'} q(s', a'))$$

or pointwise via experience $s_1 \xrightarrow{a_1} r_2, s_2$ for

$$q'(s, a) := \begin{cases} \alpha(r_2 + \gamma \max_{a'} q(s_2, a')) \\ \quad + (1 - \alpha)q(s, a) & \text{if } s = s_1 \text{ and } a = a_1 \\ q(s, a) & \text{otherwise.} \end{cases}$$

To converge to MDP's optimal Q -value, visit every state-action pair (s, a) repeatedly (for $s \xrightarrow{a} r', s'$ with diff s', r' under p, r).

End **episode**

$$s_1 \xrightarrow{a_1} r_2, s_2 \xrightarrow{a_2} r_3, s_3 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} r_n, s_n$$

at an absorbing state s_n with $r(s_n, a, s_n) = 0$ for every action a .

Exploration-exploitation tradeoff

$s \xrightarrow{a} r', s'$ r', s' from environment, but a ?

$$Q_{n+1}(s, a) := \alpha[r' + \gamma \max_{a'} Q_n(s', a')] + (1 - \alpha)Q_n(s, a)$$

from functional policy $\pi : S \rightarrow A$ [e.g. $\pi_Q(s) = \arg \max_a Q(s, a)$]

Exploration-exploitation tradeoff

$s \xrightarrow{a} r', s'$ r', s' from environment, but a ?

$$Q_{n+1}(s, a) := \alpha[r' + \gamma \max_{a'} Q_n(s', a')] + (1 - \alpha)Q_n(s, a)$$

from functional policy $\pi : S \rightarrow A$ [e.g. $\pi_Q(s) = \arg \max_a Q(s, a)$]

to $\pi : (S \times A) \rightarrow [0, 1]$ s.t. $\sum_{a \in A} \pi(s, a) = 1$ for each $s \in S$

e.g. for n actions, m having $\max Q(s, \cdot)$

$$\pi_Q^\epsilon(s, a) = \begin{cases} \frac{1-\epsilon}{m} + \frac{\epsilon}{n} & \text{if } Q(s, a) \text{ is max} \\ \frac{\epsilon}{n} & \text{otherwise} \end{cases} \quad (\dagger)$$

(\dagger) says exploit: use what we know

(\ddagger) says explore: try something new (for the future)

Exploration-exploitation tradeoff

$s \xrightarrow{a} r', s' \xrightarrow{a'} \dots$ r', s' from environment, but a ?

$$Q_{n+1}(s, a) := \alpha[r' + \gamma \max_{a'} Q_n(s', a')] + (1 - \alpha)Q_n(s, a)$$

from functional policy $\pi : S \rightarrow A$ [e.g. $\pi_Q(s) = \arg \max_a Q(s, a)$]

to $\pi : (S \times A) \rightarrow [0, 1]$ s.t. $\sum_{a \in A} \pi(s, a) = 1$ for each $s \in S$

e.g. for n actions, m having $\max Q(s, \cdot)$

$$\pi_Q^\epsilon(s, a) = \begin{cases} \frac{1-\epsilon}{m} + \frac{\epsilon}{n} & \text{if } Q(s, a) \text{ is max} \\ \frac{\epsilon}{n} & \text{otherwise} \end{cases} \quad (\dagger)$$

(\dagger) says exploit: use what we know

(\ddagger) says explore: try something new (for the future)

SARSA: replace $\arg \max$ by policy in use

$$Q_{n+1}(s, a) := \alpha[r' + \gamma Q_n(s', a')] + (1 - \alpha)Q_n(s, a)$$

Back to computation as search

```
search(Node) :- goal(Node).
```

```
search(Node) :- arc(Node,Next), search(Next).
```

Back to computation as search

```
search(Node) :- goal(Node).  
search(Node) :- arc(Node,Next), search(Next).
```

More than one Next may satisfy `arc(Node,Next)`
~~~ non-determinism

## Back to computation as search

```
search(Node) :- goal(Node).  
search(Node) :- arc(Node,Next), search(Next).
```

More than one Next may satisfy `arc(Node,Next)`

~~~ non-determinism

Choose Next closest to goal (heuristic h , Q)
perhaps keeping track of costs (min cost, A*)

Back to computation as search

```
search(Node) :- goal(Node).  
search(Node) :- arc(Node,Next), search(Next).
```

More than one Next may satisfy `arc(Node,Next)`

~~~ non-determinism

Choose Next closest to goal (heuristic  $h$ ,  $Q$ )  
perhaps keeping track of costs (min cost, A\*)

Available choices depend on `arc`

- actions specified by Turing machine (graph)

## Back to computation as search

```
search(Node) :- goal(Node).  
search(Node) :- arc(Node,Next), search(Next).
```

More than one Next may satisfy `arc(Node,Next)`  
~~ non-determinism

Choose Next closest to goal (heuristic  $h$ ,  $Q$ )  
perhaps keeping track of costs (min cost, A\*)

Available choices depend on `arc`  
- actions specified by Turing machine (graph)

Computation eliminates non-determinism (determinization)

## Back to computation as search

```
search(Node) :- goal(Node).  
search(Node) :- arc(Node,Next), search(Next).
```

More than one Next may satisfy `arc(Node,Next)`

~~ non-determinism

Choose Next closest to goal (heuristic  $h$ ,  $Q$ )  
perhaps keeping track of costs (min cost, A\*)

Available choices depend on `arc`

- actions specified by Turing machine (graph)

Computation eliminates non-determinism (determinization)

Bound number of calls to `arc` (iterations of search)

## Terminating search

Search times out after too many ticks

```
bSearch(Node,_) :- goal(Node).  
bSearch(Node,Bound) :- arc(Node,Next),  
                     tick(Bound,Less),  
                     bSearch(Next,Less).
```

## Terminating search

Search times out after too many ticks

```
bSearch(Node,_) :- goal(Node).  
bSearch(Node, Bound) :- arc(Node, Next),  
                     tick(Bound, Less),  
                     bSearch(Next, Less).
```

Design *tick* to be *terminating*

there is **no** infinite sequence  $x_1, x_2, \dots$  s.t.

$\text{tick}(x_i, x_{i+1})$  for every integer  $i > 0$

## Terminating search

Search times out after too many ticks

```
bSearch(Node,_) :- goal(Node).  
bSearch(Node, Bound) :- arc(Node, Next),  
                     tick(Bound, Less),  
                     bSearch(Next, Less).
```

Design tick to be *terminating*

there is **no** infinite sequence  $x_1, x_2, \dots$  s.t.

$\text{tick}(x_i, x_{i+1})$  for every integer  $i > 0$

and set Bound based on Start

```
search(Start) :- bound(Start, Bound),  
               bSearch(Start, Bound).
```

## Feasibility and non-determinism: P vs NP

### Cobham's Thesis

*A problem is feasibly solvable iff some deterministic Turing machine (dTm) solves it in polynomial time.*

$$P = \{\text{problems a dTm solves in polynomial time}\}$$

## Feasibility and non-determinism: P vs NP

### Cobham's Thesis

*A problem is feasibly solvable iff some deterministic Turing machine (dTm) solves it in polynomial time.*

$$P = \{\text{problems a dTm solves in polynomial time}\}$$

$$NP = \{\text{problems a non-deterministic Tm solves in polynomial time}\}$$

## Feasibility and non-determinism: P vs NP

### Cobham's Thesis

*A problem is feasibly solvable iff some deterministic Turing machine (dTm) solves it in polynomial time.*

$$P = \{\text{problems a dTm solves in polynomial time}\}$$

$$NP = \{\text{problems a non-deterministic Tm solves in polynomial time}\}$$

Clearly,  $P \subseteq NP$ .

## Feasibility and non-determinism: P vs NP

### Cobham's Thesis

*A problem is feasibly solvable iff some deterministic Turing machine (dTm) solves it in polynomial time.*

$$P = \{\text{problems a dTm solves in polynomial time}\}$$

$$NP = \{\text{problems a non-deterministic Tm solves in polynomial time}\}$$

Clearly,  $P \subseteq NP$ .

Whether  $P = NP$  is the most celebrated open mathematical problem in computer science.

## Feasibility and non-determinism: P vs NP

### Cobham's Thesis

*A problem is feasibly solvable iff some deterministic Turing machine (dTm) solves it in polynomial time.*

$$P = \{\text{problems a dTm solves in polynomial time}\}$$

$$NP = \{\text{problems a non-deterministic Tm solves in polynomial time}\}$$

Clearly,  $P \subseteq NP$ .

Whether  $P = NP$  is the most celebrated open mathematical problem in computer science.

$P \neq NP$  would mean non-determinism wrecks feasibility.

## Feasibility and non-determinism: P vs NP

### Cobham's Thesis

*A problem is feasibly solvable iff some deterministic Turing machine (dTm) solves it in polynomial time.*

$$P = \{\text{problems a dTm solves in polynomial time}\}$$

$$NP = \{\text{problems a non-deterministic Tm solves in polynomial time}\}$$

Clearly,  $P \subseteq NP$ .

Whether  $P = NP$  is the most celebrated open mathematical problem in computer science.

$P \neq NP$  would mean non-determinism wrecks feasibility.

$P = NP$  says non-determinism makes no difference to feasibility.

## A closer look

Given a set  $L$  of strings, and a Tm  $M$ .

## A closer look

Given a set  $L$  of strings, and a Tm  $M$ .

$M$  solves  $L$  in time  $n^k$  if there is a fixed integer  $c > 0$  such that for every string  $s$  of size  $n$ ,

$$s \in L \iff M \text{ accepts } s \text{ within } c \cdot n^k \text{ steps.}$$

## A closer look

Given a set  $L$  of strings, and a Tm  $M$ .

$M$  solves  $L$  in time  $n^k$  if there is a fixed integer  $c > 0$  such that for every string  $s$  of size  $n$ ,

$$s \in L \iff M \text{ accepts } s \text{ within } c \cdot n^k \text{ steps.}$$

$$\text{TIME}(n^k) := \{L \mid \text{some dTm solves } L \text{ in time } n^k\}$$

e.g.  $\text{TIME}(n)$  includes every regular language

## A closer look

Given a set  $L$  of strings, and a Tm  $M$ .

$M$  solves  $L$  in time  $n^k$  if there is a fixed integer  $c > 0$  such that for every string  $s$  of size  $n$ ,

$$s \in L \iff M \text{ accepts } s \text{ within } c \cdot n^k \text{ steps.}$$

$$\text{TIME}(n^k) := \{L \mid \text{some dTm solves } L \text{ in time } n^k\}$$

e.g.  $\text{TIME}(n)$  includes every regular language

$$P := \bigcup_{k \geq 1} \text{TIME}(n^k)$$

## A closer look

Given a set  $L$  of strings, and a Tm  $M$ .

$M$  solves  $L$  in time  $n^k$  if there is a fixed integer  $c > 0$  such that for every string  $s$  of size  $n$ ,

$$s \in L \iff M \text{ accepts } s \text{ within } c \cdot n^k \text{ steps.}$$

$$\text{TIME}(n^k) := \{L \mid \text{some dTm solves } L \text{ in time } n^k\}$$

e.g.  $\text{TIME}(n)$  includes every regular language

$$P := \bigcup_{k \geq 1} \text{TIME}(n^k)$$

$$\text{NTIME}(n^k) := \{L \mid \text{some nTm solves } L \text{ in time } n^k\}$$

$$NP := \bigcup_{k \geq 1} \text{NTIME}(n^k)$$

## Boolean satisfiability (SAT)

**SAT.** Given a Boolean expression  $\varphi$  with variables  $x_1, \dots, x_n$ , can we make  $\varphi$  true by assigning true/false to  $x_1, \dots, x_n$ ?

e.g.,  $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3})$

## Boolean satisfiability (SAT)

**SAT.** Given a Boolean expression  $\varphi$  with variables  $x_1, \dots, x_n$ , can we make  $\varphi$  true by assigning true/false to  $x_1, \dots, x_n$ ?

Checking that a particular assignment makes  $\varphi$  true is easy ( $P$ ).

$$\text{e.g., } (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3})$$

## Boolean satisfiability (SAT)

**SAT.** Given a Boolean expression  $\varphi$  with variables  $x_1, \dots, x_n$ , can we make  $\varphi$  true by assigning true/false to  $x_1, \dots, x_n$ ?

Checking that a particular assignment makes  $\varphi$  true is easy ( $P$ ). Non-determinism (guessing the assignment) puts SAT in  $NP$ .

e.g.,  $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3})$

## Boolean satisfiability (SAT)

**SAT.** Given a Boolean expression  $\varphi$  with variables  $x_1, \dots, x_n$ , can we make  $\varphi$  true by assigning true/false to  $x_1, \dots, x_n$ ?

Checking that a particular assignment makes  $\varphi$  true is easy ( $P$ ). Non-determinism (guessing the assignment) puts SAT in  $NP$ . But is SAT in  $P$ ? There are  $2^n$  assignments to try.

$$\text{e.g., } (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3})$$

## Boolean satisfiability (SAT)

**SAT.** Given a Boolean expression  $\varphi$  with variables  $x_1, \dots, x_n$ , can we make  $\varphi$  true by assigning true/false to  $x_1, \dots, x_n$ ?

Checking that a particular assignment makes  $\varphi$  true is easy ( $P$ ). Non-determinism (guessing the assignment) puts SAT in  $NP$ . But is SAT in  $P$ ? There are  $2^n$  assignments to try.

**Cook-Levin Theorem.** *SAT is in P iff P = NP.*

e.g.,  $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3})$

## Boolean satisfiability (SAT)

**SAT.** Given a Boolean expression  $\varphi$  with variables  $x_1, \dots, x_n$ , can we make  $\varphi$  true by assigning true/false to  $x_1, \dots, x_n$ ?

Checking that a particular assignment makes  $\varphi$  true is easy ( $P$ ). Non-determinism (guessing the assignment) puts SAT in  $NP$ . But is SAT in  $P$ ? There are  $2^n$  assignments to try.

**Cook-Levin Theorem.** *SAT is in P iff P = NP.*

e.g.,  $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3})$

CSAT:  $\varphi$  is a conjunction of clauses, where a *clause* is an OR of literals, and a *literal* is a variable  $x_i$  or negated variable  $\overline{x_i}$

## Boolean satisfiability (SAT)

**SAT.** Given a Boolean expression  $\varphi$  with variables  $x_1, \dots, x_n$ , can we make  $\varphi$  true by assigning true/false to  $x_1, \dots, x_n$ ?

Checking that a particular assignment makes  $\varphi$  true is easy ( $P$ ). Non-determinism (guessing the assignment) puts SAT in  $NP$ . But is SAT in  $P$ ? There are  $2^n$  assignments to try.

**Cook-Levin Theorem.** *SAT is in P iff P = NP.*

$$\text{e.g., } (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3})$$

CSAT:  $\varphi$  is a conjunction of clauses, where a *clause* is an OR of literals, and a *literal* is a variable  $x_i$  or negated variable  $\overline{x_i}$

$k$ -SAT: every clause has exactly  $k$  literals  
3-SAT is as hard as SAT, 2-SAT is in  $P$

## Boolean satisfiability (SAT)

**SAT.** Given a Boolean expression  $\varphi$  with variables  $x_1, \dots, x_n$ , can we make  $\varphi$  true by assigning true/false to  $x_1, \dots, x_n$ ?

Checking that a particular assignment makes  $\varphi$  true is easy ( $P$ ). Non-determinism (guessing the assignment) puts SAT in  $NP$ . But is SAT in  $P$ ? There are  $2^n$  assignments to try.

**Cook-Levin Theorem.** *SAT is in P iff  $P = NP$ .*

$$\text{e.g., } (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3})$$

CSAT:  $\varphi$  is a conjunction of clauses, where a *clause* is an OR of literals, and a *literal* is a variable  $x_i$  or negated variable  $\overline{x_i}$

$k$ -SAT: every clause has exactly  $k$  literals

3-SAT is as hard as SAT, 2-SAT is in  $P$

Horn-SAT: every clause has at most one positive literal — linear

## Prolog and SAT

Prolog KB (definite clauses)

x1 :- x2, x4.

x2 :- x3.               $\rightsquigarrow$        $[[x_1, x_2, x_4], [x_2, x_3], [x_4]]$

x4.

## Prolog and SAT

Prolog KB (definite clauses)

x1 :- x2, x4.

x2 :- x3.       $\rightsquigarrow$     [[x1, x2, x4], [x2, x3], [x4]]

x4.

CSAT-input

x1  $\vee$   $\overline{x2}$   $\vee$   $\overline{x4}$

x2  $\vee$   $\overline{x3}$        $\rightsquigarrow$     [[1, -2, -4], [2, -3], [4]]

x4.

## Prolog and SAT

Prolog KB (definite clauses)

```
x1 :- x2,x4.  
x2 :- x3.          ~~~ [[x1,x2,x4],[x2,x3],[x4]]  
x4.
```

CSAT-input

```
x1 ∨  $\overline{x2}$  ∨  $\overline{x4}$   
x2 ∨  $\overline{x3}$           ~~~ [[1,-2,-4],[2,-3],[4]]  
x4.
```

The assignment making **all** variables TRUE satisfies all CSAT-inputs in which every clause has a positive literal.  
(All definite clause KBs are satisfiable.)

## Prolog and SAT

Prolog KB (definite clauses)

```
x1 :- x2, x4.  
x2 :- x3.          ~~~ [[x1,x2,x4], [x2,x3], [x4]]  
x4.
```

CSAT-input

```
x1 ∨  $\overline{x2}$  ∨  $\overline{x4}$   
x2 ∨  $\overline{x3}$           ~~~ [[1,-2,-4], [2,-3], [4]]  
x4.
```

The assignment making **all** variables TRUE satisfies all CSAT-inputs in which every clause has a positive literal.  
(All definite clause KBs are satisfiable.)

From proofs to **unsatisfiability**:

$$\underbrace{KB \text{ proves } \varphi}_{\text{Prolog}} \iff \underbrace{KB, \overline{\varphi}}_{\text{Horn (linear SAT)}} \text{ is not satisfiable}$$

## Constraint Satisfaction Problem [Var, Dom, Con]

- ▶ a list  $\text{Var} = [X_1, \dots, X_n]$  of *variables*  $X_i$ ;
- ▶ a list  $\text{Dom} = [D_1, \dots, D_n]$  of finite sets  $D_i$  of size  $s_i$ ;
- ▶ a finite set  $\text{Con}$  of *constraints* that may or may not be satisfied by (a node) instantiating  $X_i$  with a value in  $D_i$ ;  
(search space size  $\prod_{i=1}^n s_i$ )

## Constraint Satisfaction Problem [Var, Dom, Con]

- ▶ a list  $\text{Var} = [X_1, \dots, X_n]$  of *variables*  $X_i$ ;
- ▶ a list  $\text{Dom} = [D_1, \dots, D_n]$  of finite sets  $D_i$  of size  $s_i$ ;
- ▶ a finite set  $\text{Con}$  of *constraints* that may or may not be satisfied by (a node) instantiating  $X_i$  with a value in  $D_i$  (search space size  $\prod_{i=1}^n s_i$ )

**E.g. SAT:**  $D_i = \{0, 1\}$ ,  $s_i = 2$  for search space of size  $2^n$

## Constraint Satisfaction Problem [Var, Dom, Con]

- ▶ a list  $\text{Var} = [X_1, \dots, X_n]$  of *variables*  $X_i$ ;
- ▶ a list  $\text{Dom} = [D_1, \dots, D_n]$  of finite sets  $D_i$  of size  $s_i$ ;
- ▶ a finite set  $\text{Con}$  of *constraints* that may or may not be satisfied by (a node) instantiating  $X_i$  with a value in  $D_i$  (search space size  $\prod_{i=1}^n s_i$ )

**E.g. SAT:**  $D_i = \{0, 1\}$ ,  $s_i = 2$  for search space of size  $2^n$

**Problem:** satisfy all constraints in  $\text{Con}$ , instantiating variables if necessary/convenient

## Constraint Satisfaction Problem [Var, Dom, Con]

- ▶ a list  $\text{Var} = [X_1, \dots, X_n]$  of *variables*  $X_i$ ;
- ▶ a list  $\text{Dom} = [D_1, \dots, D_n]$  of finite sets  $D_i$  of size  $s_i$ ;
- ▶ a finite set  $\text{Con}$  of *constraints* that may or may not be satisfied by (a node) instantiating  $X_i$  with a value in  $D_i$  (search space size  $\prod_{i=1}^n s_i$ )

**E.g. SAT:**  $D_i = \{0, 1\}$ ,  $s_i = 2$  for search space of size  $2^n$

**Problem:** satisfy all constraints in  $\text{Con}$ , instantiating variables if necessary/convenient

```
| ?- X \= Y, X=a, Y=b.
```

no

## Constraint Satisfaction Problem [Var, Dom, Con]

- ▶ a list  $\text{Var} = [X_1, \dots, X_n]$  of *variables*  $X_i$ ;
- ▶ a list  $\text{Dom} = [D_1, \dots, D_n]$  of finite sets  $D_i$  of size  $s_i$ ;
- ▶ a finite set  $\text{Con}$  of *constraints* that may or may not be satisfied by (a node) instantiating  $X_i$  with a value in  $D_i$  (search space size  $\prod_{i=1}^n s_i$ )

**E.g. SAT:**  $D_i = \{0, 1\}$ ,  $s_i = 2$  for search space of size  $2^n$

**Problem:** satisfy all constraints in  $\text{Con}$ , instantiating variables if necessary/convenient

| ?-  $X \backslash= Y$ ,  $X = a$ ,  $Y = b$ .  
no

| ?-  $X = a$ ,  $Y = b$ ,  $X \backslash= Y$ .  
 $X = a$ ,  $Y = b$

## Order-independent unification (Martelli-Montanari)

**Input:** set  $\mathcal{E}$  of pairs  $[t, t']$

**Output:** substitution  $[[X_1, t_1], \dots, [X_k, t_k]]$  unifying pairs in  $\mathcal{E}$

## Order-independent unification (Martelli-Montanari)

**Input:** set  $\mathcal{E}$  of pairs  $[t, t']$

**Output:** substitution  $[[X_1, t_1], \dots, [X_k, t_k]]$  unifying pairs in  $\mathcal{E}$

Simplify  $\mathcal{E}$  non-deterministically until no longer possible

1.  $[f(s_1, \dots, s_k), f(t_1, \dots, t_k)]$  (allowing  $k = 0$ )  
     $\Rightarrow$  replace by pairs  $[s_1, t_1], \dots, [s_k, t_k]$
2.  $[f(s_1, \dots, s_k), g(t_1, \dots, t_m)]$  where  $f \neq g$  or  $k \neq m$   
     $\Rightarrow$  halt with failure
3.  $[X, X] \Rightarrow$  delete
4.  $[t, X]$  where  $t$  is not a var  $\Rightarrow$  replace by  $[X, t]$
5.  $[X, t]$  where  $X \notin \text{Var}(t)$  and  $X$  occurs elsewhere  
     $\Rightarrow$  apply  $[X, t]$  to all other pairs
6.  $[X, t]$  where  $X \in \text{Var}(t)$  and  $X \neq t \Rightarrow$  halt with failure

## Order-independent unification (Martelli-Montanari)

**Input:** set  $\mathcal{E}$  of pairs  $[t, t']$

**Output:** substitution  $[[X_1, t_1], \dots, [X_k, t_k]]$  unifying pairs in  $\mathcal{E}$

Simplify  $\mathcal{E}$  non-deterministically until no longer possible

1.  $[f(s_1, \dots, s_k), f(t_1, \dots, t_k)]$  (allowing  $k = 0$ )  
     $\Rightarrow$  replace by pairs  $[s_1, t_1], \dots, [s_k, t_k]$
2.  $[f(s_1, \dots, s_k), g(t_1, \dots, t_m)]$  where  $f \neq g$  or  $k \neq m$   
     $\Rightarrow$  halt with failure
3.  $[X, X] \Rightarrow$  delete
4.  $[t, X]$  where  $t$  is not a var  $\Rightarrow$  replace by  $[X, t]$
5.  $[X, t]$  where  $X \notin \text{Var}(t)$  and  $X$  occurs elsewhere  
     $\Rightarrow$  apply  $[X, t]$  to all other pairs
6.  $[X, t]$  where  $X \in \text{Var}(t)$  and  $X \neq t \Rightarrow$  halt with failure

**N.B.** Prolog omits *occurs check*  $X \in \text{Var}(t)$  in 5, 6 for speed-up

## Instantiate before negating (as failure)

```
% \+p :- (p,! ,fail); true.
```

```
p(X) :- \+q(X), r(X).
```

```
q(a). q(b).
```

```
r(a). r(c).
```

-----

```
| ?- p(X). % contra ?- p(c).
```

## Instantiate before negating (as failure)

```
% \+p :- (p,! ,fail); true.
```

```
p(X) :- \+q(X), r(X).
```

```
q(a). q(b).
```

```
r(a). r(c).
```

-----

```
| ?- p(X). % contra ?- p(c).
```

```
p(X) :- \+q(X).
```

```
q(X) :- \+r(X).
```

```
r(c).
```

-----

```
| ?- p(X). % contra ?- p(a).
```

## Generate-and-test

brute force: instantiate all variables before testing constraints

```
genTest(D1...Dn) :- node(X1...Xn,D1...Dn),  
                  constraint(X1...Xn).
```

```
node(X1...Xn,D1...Dn) :- member(X1,D1),...,  
                         member(Xn,Dn).
```

## Generate-and-test

brute force: instantiate all variables before testing constraints

```
genTest(D1...Dn) :- node(X1...Xn,D1...Dn),  
                  constraint(X1...Xn).  
  
node(X1...Xn,D1...Dn) :- member(X1,D1),...,  
                      member(Xn,Dn).
```

For each of the  $\prod_{i=1}^n s_i$ -choices of  $X_1 \dots X_n$  such that

$\text{node}(X_1 \dots X_n, D_1 \dots D_n)$

(with  $D_i$  of size  $s_i$ ), assume

$\text{constraint}(X_1 \dots X_n)$

can be checked within a polynomial of  $X_1 \dots X_n$ .

## Generate-and-test

brute force: instantiate all variables before testing constraints

```
genTest(D1...Dn) :- node(X1...Xn,D1...Dn),  
                  constraint(X1...Xn).  
  
node(X1...Xn,D1...Dn) :- member(X1,D1),...,  
                      member(Xn,Dn).
```

For each of the  $\prod_{i=1}^n s_i$ -choices of  $X_1 \dots X_n$  such that

$\text{node}(X_1 \dots X_n, D_1 \dots D_n)$

(with  $D_i$  of size  $s_i$ ), assume

$\text{constraint}(X_1 \dots X_n)$

can be checked within a polynomial of  $X_1 \dots X_n$ .

Nodes are generated in lexicographic order *without* regard to constraints.

## Inferring changes

Horn-SAT by minimal changes to  $00 \cdots 0$  (all variables 0/false)

| CSAT                          | definite clause  | list encoding   |
|-------------------------------|------------------|-----------------|
| $\bar{u} \vee x \vee \bar{z}$ | $x :- u, z.$     | $[x, u, z]$     |
| $\bar{u} \vee \bar{v}$        | $false :- u, v.$ | $[false, u, v]$ |

## Inferring changes

Horn-SAT by minimal changes to  $00 \dots 0$  (all variables 0/false)

| CSAT                          | definite clause  | list encoding   |
|-------------------------------|------------------|-----------------|
| $\bar{u} \vee x \vee \bar{z}$ | $x :- u, z.$     | $[x, u, z]$     |
| $\bar{u} \vee \bar{v}$        | $false :- u, v.$ | $[false, u, v]$ |

For each stage  $i$ , collect the variables set at stage  $i$  to 1/true in  $A_i$

$$A_0 := \emptyset \quad (\text{all variables false})$$

$$A_{i+1} := \{x \mid \underbrace{\text{member}([x|T], KB)}_{x :- t_1 \dots t_k \text{ in } KB} \text{ and } \underbrace{\text{all}(T, A_i)}_{\{t_1 \dots t_k\} \subseteq A_i}\}$$

check:  $false \notin A_n$

## Inferring changes

Horn-SAT by minimal changes to  $00 \dots 0$  (all variables 0/false)

| CSAT                          | definite clause  | list encoding   |
|-------------------------------|------------------|-----------------|
| $\bar{u} \vee x \vee \bar{z}$ | $x :- u, z.$     | $[x, u, z]$     |
| $\bar{u} \vee \bar{v}$        | $false :- u, v.$ | $[false, u, v]$ |

For each stage  $i$ , collect the variables set at stage  $i$  to 1/true in  $A_i$

$$A_0 := \emptyset \quad (\text{all variables false})$$

$$A_{i+1} := \{x \mid \underbrace{\text{member}([x|T], KB)}_{x :- t_1 \dots t_k \text{ in } KB} \text{ and } \underbrace{\text{all}(T, A_i)}_{\{t_1 \dots t_k\} \subseteq A_i}\}$$

check:  $false \notin A_n$

No minimal set for non-Horn  $x \vee y$  (or xor).

## Instantiate one variable at a time

allow node to map  $X_i$  to ?, raising search space size from

$$\prod_{i=1}^n s_i \text{ to } \prod_{i=1}^n (s_i + 1) \text{ from adding ? to } D_i$$

PAY-OFF: search tree of depth  $n$  and branching factor  $\max_i s_i$   
with start node instantiating no variable, and  
an arc instantiating least uninstantiated variable

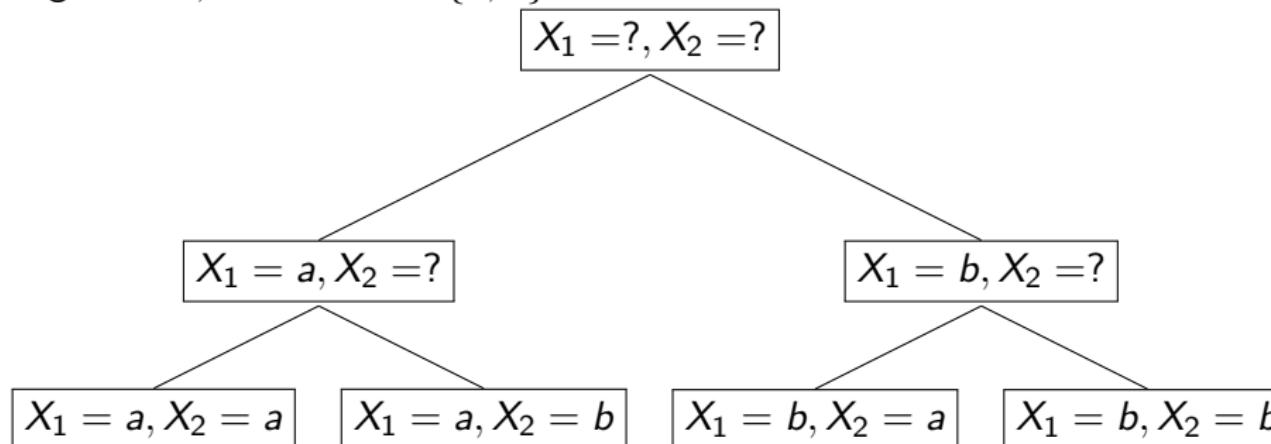
## Instantiate one variable at a time

allow node to map  $X_i$  to ?, raising search space size from

$$\prod_{i=1}^n s_i \text{ to } \prod_{i=1}^n (s_i + 1) \text{ from adding ? to } D_i$$

PAY-OFF: search tree of depth  $n$  and branching factor  $\max_i s_i$   
with start node instantiating no variable, and  
an arc instantiating least uninstantiated variable

E.g.  $n = 2, D_1 = D_2 = \{a, b\}$



## Interleave generation with testing + backtracking

whenever  $\text{arc}(N0, N1)$ ,

$N1$  instantiates one more variable than  $N0$ , and

$N1$  satisfies every constraint on instantiated variables

## Interleave generation with testing + backtracking

whenever  $\text{arc}(N0, N1)$ ,

$N1$  instantiates one more variable than  $N0$ , and

$N1$  satisfies every constraint on instantiated variables

Reduce domains of un-instantiated variables via constraints

**Constraint Graph:** node = variable (e.g. 3-Color)

$$\text{arc}(X_i, X_j) \iff \text{Con}[X_i, X_j] \neq \emptyset$$

*Arc Consistency:* for  $\text{arc}(X_i, X_j)$  and  $i < j$ ,

$$(\forall d \in D(X_i))(\exists d' \in D(X_j)) \text{ } d, d' \text{ satisfy } \text{Con}[X_i, X_j]$$

removing  $d$  from  $D(X_i)$  when no such  $d'$  exists

## Interleave generation with testing + backtracking

whenever  $\text{arc}(N0, N1)$ ,

$N1$  instantiates one more variable than  $N0$ , and

$N1$  satisfies every constraint on instantiated variables

Reduce domains of un-instantiated variables via constraints

**Constraint Graph:** node = variable (e.g. 3-Color)

$$\text{arc}(X_i, X_j) \iff \text{Con}[X_i, X_j] \neq \emptyset$$

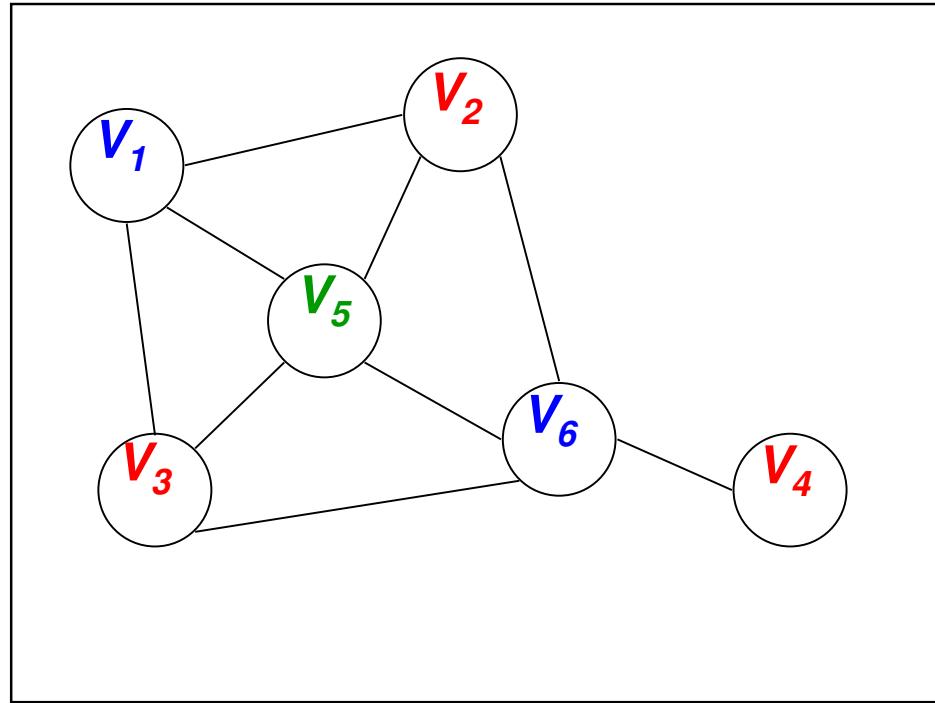
*Arc Consistency:* for  $\text{arc}(X_i, X_j)$  and  $i < j$ ,

$$(\forall d \in D(X_i))(\exists d' \in D(X_j)) \text{ } d, d' \text{ satisfy } \text{Con}[X_i, X_j]$$

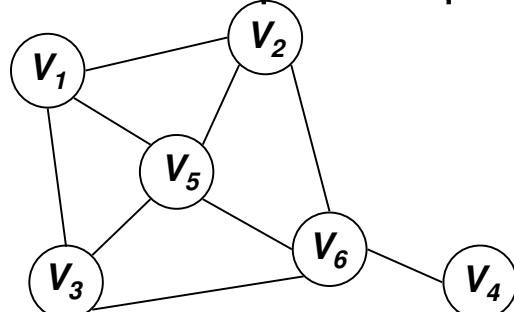
removing  $d$  from  $D(X_i)$  when no such  $d'$  exists

Optimizing the backtracking search

- ▶ MRV: instantiate variable with **minimum remaining values** (to minimize branching/cases)
- ▶ LCV: assign **value** that is **least constraining** (for greatest chance of success)



### Canonical Example: Graph Coloring



- Consider  $N$  nodes in a graph
- Assign values  $V_1, \dots, V_N$  to each of the  $N$  nodes
- The values are taken in  $\{R, G, B\}$
- Constraints: If there is an edge between  $i$  and  $j$ , then  $V_i$  must be different of  $V_j$

# Slides mainly from Poole & Mackworth, chap 13

Datalog based on the following assumptions

- An agent's knowledge can be usefully described in terms of *individuals* and *relations* among individuals.
- An agent's knowledge base consists of *definite* and *positive* statements.
- The environment is *static*.
- There are only a finite number of individuals of interest in the domain.

An individual can be named.

## Datalog syntax

- A **variable** starts with upper-case letter.
- A **constant** starts with lower-case letter or is a sequence of digits (numeral).
- A **predicate symbol** starts with lower-case letter.
- A **term** is either a variable or a constant.
- An **atomic symbol** (atom) is of the form  $p$  or  $p(t_1, \dots, t_n)$  where  $p$  is a predicate symbol and  $t_i$  are terms.

## Datalog syntax (ctd)

- A **definite clause** is either an atomic symbol (a fact) or of the form:

$$\underbrace{a}_{\text{head}} \quad \leftarrow \quad \underbrace{b_1 \wedge \cdots \wedge b_m}_{\text{body}}$$

where  $a$  and  $b_i$  are atomic symbols.

- **query** is of the form  $?b_1 \wedge \cdots \wedge b_m$ .
- **knowledge base** is a set of definite clauses.

## Semantics: General Idea

A **semantics** specifies the meaning of sentences in the language.

An **interpretation** specifies:

- what objects (individuals) are in the world
- the correspondence between symbols in the computer and objects & relations in world
  - ▶ constants denote individuals
  - ▶ predicate symbols denote relations

# Formal Semantics

An **interpretation** is a triple  $I = \langle D, \phi, \pi \rangle$ , where

- $D$ , the **domain**, is a nonempty set. Elements of  $D$  are **individuals**.
- $\phi$  is a mapping that assigns to each constant an element of  $D$ . Constant  $c$  **denotes** individual  $\phi(c)$ .
- $\pi$  is a mapping that assigns to each  $n$ -ary predicate symbol a relation: a function from  $D^n$  into  $\{\text{TRUE}, \text{FALSE}\}$ .

## Example Interpretation

Constants: *phone*, *pencil*, *telephone*.

Predicate Symbol: *noisy* (unary), *left\_of* (binary).

- $D = \{\text{☒}, \text{☎}, \text{✏}\}$ .
- $\phi(\text{phone}) = \text{☎}$ ,  $\phi(\text{pencil}) = \text{✏}$ ,  $\phi(\text{telephone}) = \text{☒}$ .
- $\pi(\text{noisy})$ : 

|                            |       |                            |      |                            |       |
|----------------------------|-------|----------------------------|------|----------------------------|-------|
| $\langle \text{☒} \rangle$ | FALSE | $\langle \text{☎} \rangle$ | TRUE | $\langle \text{✏} \rangle$ | FALSE |
|----------------------------|-------|----------------------------|------|----------------------------|-------|

  
 $\pi(\text{left\_of})$ : 

|                                      |       |                                      |       |                                      |       |
|--------------------------------------|-------|--------------------------------------|-------|--------------------------------------|-------|
| $\langle \text{☒}, \text{☒} \rangle$ | FALSE | $\langle \text{☒}, \text{☎} \rangle$ | TRUE  | $\langle \text{☒}, \text{✏} \rangle$ | TRUE  |
| $\langle \text{☎}, \text{☒} \rangle$ | FALSE | $\langle \text{☎}, \text{☎} \rangle$ | FALSE | $\langle \text{☎}, \text{✏} \rangle$ | TRUE  |
| $\langle \text{✏}, \text{☒} \rangle$ | FALSE | $\langle \text{✏}, \text{☎} \rangle$ | FALSE | $\langle \text{✏}, \text{✏} \rangle$ | FALSE |

## Important points to note

- The domain  $D$  can contain real objects. (e.g., a person, a room, a course).  $D$  can't necessarily be stored in a computer.
- $\pi(p)$  specifies whether the relation denoted by the  $n$ -ary predicate symbol  $p$  is true or false for each  $n$ -tuple of individuals.
- If predicate symbol  $p$  has no arguments, then  $\pi(p)$  is either *TRUE* or *FALSE*.

## Truth in an interpretation

A constant  $c$  denotes in  $I$  the individual  $\phi(c)$ .

Ground (variable-free) atom  $p(t_1, \dots, t_n)$  is

- true in interpretation  $I$  if  $\pi(p)(\langle \phi(t_1), \dots, \phi(t_n) \rangle) = \text{TRUE}$  in interpretation  $I$  and
- false otherwise.

Ground clause  $h \leftarrow b_1 \wedge \dots \wedge b_m$  is false in interpretation  $I$  if  $h$  is false in  $I$  and each  $b_i$  is true in  $I$ , and is true in interpretation  $I$  otherwise.

## Example Truths

In the interpretation given before, which of following are true?

*noisy(phone)*

*noisy(telephone)*

*noisy(pencil)*

*left\_of(phone, pencil)*

*left\_of(phone, telephone)*

*noisy(phone) ← left\_of(phone, telephone)*

*noisy(pencil) ← left\_of(phone, telephone)*

*noisy(pencil) ← left\_of(phone, pencil)*

*noisy(phone) ← noisy(telephone) ∧ noisy(pencil)*

## Example Truths

In the interpretation given before, which of following are true?

|                                                                                                           |       |
|-----------------------------------------------------------------------------------------------------------|-------|
| $\text{noisy}(\text{phone})$                                                                              | true  |
| $\text{noisy}(\text{telephone})$                                                                          | true  |
| $\text{noisy}(\text{pencil})$                                                                             | false |
| $\text{left\_of}(\text{phone}, \text{pencil})$                                                            | true  |
| $\text{left\_of}(\text{phone}, \text{telephone})$                                                         | false |
| $\text{noisy}(\text{phone}) \leftarrow \text{left\_of}(\text{phone}, \text{telephone})$                   | true  |
| $\text{noisy}(\text{pencil}) \leftarrow \text{left\_of}(\text{phone}, \text{telephone})$                  | true  |
| $\text{noisy}(\text{pencil}) \leftarrow \text{left\_of}(\text{phone}, \text{pencil})$                     | false |
| $\text{noisy}(\text{phone}) \leftarrow \text{noisy}(\text{telephone}) \wedge \text{noisy}(\text{pencil})$ | true  |

## Models and logical consequences

- A knowledge base,  $KB$ , is true in interpretation  $I$  if and only if every clause in  $KB$  is true in  $I$ .
- A **model** of a set of clauses is an interpretation in which all the clauses are true.
- If  $KB$  is a set of clauses and  $g$  is a conjunction of atoms,  $g$  is a **logical consequence** of  $KB$ , written  $KB \models g$ , if  $g$  is true in every model of  $KB$ .
- That is,  $KB \models g$  if there is no interpretation in which  $KB$  is true and  $g$  is false.

## User's view of Semantics

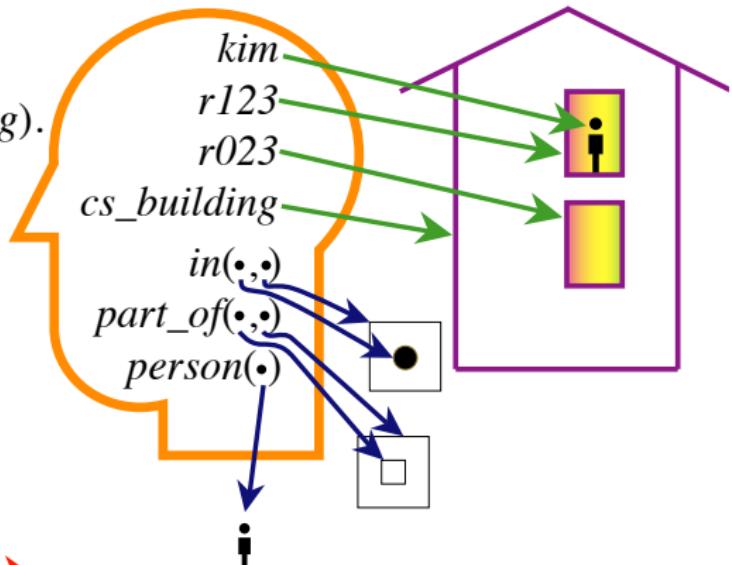
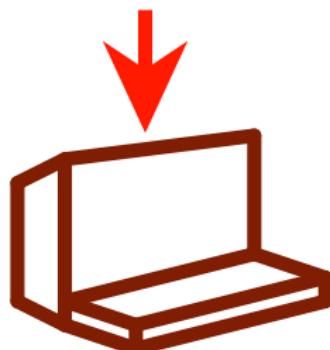
1. Choose a task domain: intended interpretation.
2. Associate constants with individuals you want to name.
3. For each relation you want to represent, associate a predicate symbol in the language.
4. Tell the system clauses that are true in the intended interpretation: axiomatizing the domain.
5. Ask questions about the intended interpretation.
6. If  $KB \models g$ , then  $g$  must be true in the intended interpretation.

## Computer's view of semantics

- The computer doesn't have access to the intended interpretation.
- All it knows is the knowledge base.
- The computer can determine if a formula is a logical consequence of KB.
- If  $KB \models g$  then  $g$  must be true in the intended interpretation.
- If  $KB \not\models g$  then there is a model of  $KB$  in which  $g$  is false.  
This could be the intended interpretation.

# Semantics in the mind

```
in(kim,r123).  
part_of(r123,cs_building).  
in(X,Y) ←  
    part_of(Z,Y) ∧  
    in(X,Z).
```



*in(kim,cs\_building)*

## Soundness and completeness

Recall that  $g$  is a *logical consequence* of  $KB$ ,  $KB \models g$ , precisely if  $g$  is true in all models of  $KB$ .

Let  $\vdash$  be a mechanical procedure for deriving a formula  $g$  from a knowledge base  $KB$ , written  $KB \vdash g$ .

$\vdash$  is **sound** if  $KB \models g$  whenever  $KB \vdash g$ .

## Soundness and completeness

Recall that  $g$  is a *logical consequence* of  $KB$ ,  $KB \models g$ , precisely if  $g$  is true in all models of  $KB$ .

Let  $\vdash$  be a mechanical procedure for deriving a formula  $g$  from a knowledge base  $KB$ , written  $KB \vdash g$ .

$\vdash$  is **sound** if  $KB \models g$  whenever  $KB \vdash g$ .

$\vdash$  is **complete** if  $KB \vdash g$  whenever  $KB \models g$ .

## Soundness and completeness

Recall that  $g$  is a *logical consequence* of  $KB$ ,  $KB \models g$ , precisely if  $g$  is true in all models of  $KB$ .

Let  $\vdash$  be a mechanical procedure for deriving a formula  $g$  from a knowledge base  $KB$ , written  $KB \vdash g$ .

$\vdash$  is **sound** if  $KB \models g$  whenever  $KB \vdash g$ .

$\vdash$  is **complete** if  $KB \vdash g$  whenever  $KB \models g$ .

Two extreme examples:

(1)  $KB \vdash g$  for **no**  $g$   
sound

(2)  $KB \vdash g$  for **all**  $g$   
complete

## Soundness and completeness

Recall that  $g$  is a *logical consequence* of  $KB$ ,  $KB \models g$ , precisely if  $g$  is true in all models of  $KB$ .

Let  $\vdash$  be a mechanical procedure for deriving a formula  $g$  from a knowledge base  $KB$ , written  $KB \vdash g$ .

$\vdash$  is **sound** if  $KB \models g$  whenever  $KB \vdash g$ .

$\vdash$  is **complete** if  $KB \vdash g$  whenever  $KB \models g$ .

Two extreme examples:

- (1)  $KB \vdash g$  for **no**  $g$       'say nothing' undergenerates  
sound
- (2)  $KB \vdash g$  for **all**  $g$       'say everything' overgenerates  
complete

# Propositional KBs

## Recall

```
i :- p,q.  
i :- r.  
p.  
r.
```

# Propositional KBs

Recall

```
i :- p,q.  
i :- r.  
p.  
r.
```

```
KB = [[i,p,q],[i,r],[p],[r]]  
arc([H|T],N,KB) :- member([H|B],KB),  
                      append(B,T,N).  
prove([] ,KB).  
prove(Node,KB) :- arc(Node,Next,KB),  
                  prove(Next,KB).
```

# Propositional KBs

Recall

```
i :- p,q.  
i :- r.  
p.  
r.
```

```
KB = [[i,p,q],[i,r],[p],[r]]  
arc([H|T],N,KB) :- member([H|B],KB),  
                      append(B,T,N).  
prove([] ,KB).  
prove(Node,KB) :- arc(Node,Next,KB),  
                  prove(Next,KB).
```

Let

$$KB \vdash G \iff \text{prove}([G], KB)$$

## Theorem.

- (1)  $\vdash$  is sound (proved by induction)
- (2)  $\vdash$  is *not* complete (why?)

## Logical consequences bottom-up

$$C_0 := \emptyset$$

$$C_{n+1} := \{H \mid (\text{for some } B \subseteq C_n) \text{ member}([H|B], KB)\}$$

$$C := \bigcup_{n \geq 0} C_n$$

$$= \bigcup_{n \leq k} C_n \quad \text{where } k = \text{number of clauses in } KB$$

## Logical consequences bottom-up

$$C_0 := \emptyset$$

$$C_{n+1} := \{H \mid (\text{for some } B \subseteq C_n) \text{ member}([H|B], KB)\}$$

$$C := \bigcup_{n \geq 0} C_n$$

$$= \bigcup_{n \leq k} C_n \quad \text{where } k = \text{number of clauses in } KB$$

i :- p,q.

KB = [[i,p,q],[i,r],[p],[r]]

i :- r.

arc([H|T], N, KB) :- member([H|B], KB),  
append(B, T, N).

p.

$$C_1 = \{p, r\}$$

$$C_2 = \{p, r, i\} = C_n \text{ for } n \geq 2$$

A 0-ary predicate  $p$  is interpreted by  $I = \langle D, \phi, \pi \rangle$  as

$$\pi(p) : D^0 \rightarrow \{\text{true}, \text{false}\}.$$

## Substitutions and instances

A 0-ary predicate  $p$  is interpreted by  $I = \langle D, \phi, \pi \rangle$  as

$$\pi(p) : D^0 \rightarrow \{\text{true}, \text{false}\}.$$

Let  $K$  be a set of constants.

A  **$K$ -substitution** is a function from a finite set of variables to  $K$  — i.e. a set  $\{V_1/c_1, \dots, V_n/c_n\}$  of  $c_i \in K$  and distinct variables  $V_i$ .

The **application  $e\theta$**  of a  $K$ -substitution  $\theta = \{V_1/c_1, \dots, V_n/c_n\}$  to a clause  $e$  is  $e$  with each  $V_i$  replaced by  $c_i$

$$\text{e.g. } p(Z, U, Y, a, X)\{X/b, U/a, Z/b\} = p(b, a, Y, a, b).$$

A  **$K$ -instance** of  $e$  is  $e\theta$  for some  $K$ -substitution  $\theta$ .

Given a set  $B$  of clauses and a  $K$ -substitution  $\theta$ , let

$$B\theta := \{e\theta \mid e \in B\}.$$

## Bottom-up with substitutions

If  $KB$  has constants from some non-empty finite set  $K$ , let

$$C_0^K := \emptyset$$

$$C_{n+1}^K := \{H\theta \mid \theta \text{ is a } K\text{-substitution s.t. } B\theta \subseteq C_n^K \\ \text{for some } B \text{ s.t. member}([H|B], KB)\}$$

$$C^K := \bigcup_{n \geq 0} C_n^K$$

## Bottom-up with substitutions

If  $KB$  has constants from some non-empty finite set  $K$ , let

$$C_0^K := \emptyset$$

$$C_{n+1}^K := \{H\theta \mid \theta \text{ is a } K\text{-substitution s.t. } B\theta \subseteq C_n^K \text{ for some } B \text{ s.t. member}([H|B], KB)\}$$

$$C^K := \bigcup_{n \geq 0} C_n^K$$

E.g. for  $KB = [[p(a, b)], [q(X), p(X, Y)]]$  and  $K = \{a, b\}$ ,

$$C_1^K = \{p(a, b)\}$$

$$C_2^K = \{p(a, b), q(a)\} = C^K$$

## Soundness & completeness via Herbrand

The **Herbrand interpretation** of a set  $KB$  of clauses with constants from a non-empty set  $K$  is the triple  $I = \langle D, \phi, \pi \rangle$  where

- the domain  $D$  is the set  $K$  of constants
- $\phi$  is the identity function on  $K$  (each constant in  $K$  refers to itself)
- for each  $n$ -ary  $p$  and  $n$ -tuple  $c_1 \dots c_n$  from  $K$ ,

$$\pi(p)(c_1 \dots c_n) = \text{ true} \iff p(c_1 \dots c_n) \in C^K$$

## Soundness & completeness via Herbrand

The **Herbrand interpretation** of a set  $KB$  of clauses with constants from a non-empty set  $K$  is the triple  $I = \langle D, \phi, \pi \rangle$  where

- the domain  $D$  is the set  $K$  of constants
- $\phi$  is the identity function on  $K$  (each constant in  $K$  refers to itself)
- for each  $n$ -ary  $p$  and  $n$ -tuple  $c_1 \dots c_n$  from  $K$ ,

$$\pi(p)(c_1 \dots c_n) = \text{ true } \iff p(c_1 \dots c_n) \in C^K$$

**Fact.**  $I$  is a model of  $KB$ , and every clause true in  $I$  is true in every model of  $KB$  (interpreting constants in  $K$ ).

## Soundness & completeness via Herbrand

The **Herbrand interpretation** of a set  $KB$  of clauses with constants from a non-empty set  $K$  is the triple  $I = \langle D, \phi, \pi \rangle$  where

- the domain  $D$  is the set  $K$  of constants
- $\phi$  is the identity function on  $K$  (each constant in  $K$  refers to itself)
- for each  $n$ -ary  $p$  and  $n$ -tuple  $c_1 \dots c_n$  from  $K$ ,

$$\pi(p)(c_1 \dots c_n) = \text{ true } \iff p(c_1 \dots c_n) \in C^K$$

**Fact.**  $I$  is a model of  $KB$ , and every clause true in  $I$  is true in every model of  $KB$  (interpreting constants in  $K$ ).

**Corollary.** The bottom-up procedure with substitutions is sound and complete (for Datalog).

# These slides are adapted from Poole & Mackworth, chap 8

From a Constraint Satisfaction Problem [Var,Dom,Con] to  
random variables with probabilities constrained by a graph

- The domain (range) of a variable  $X$ , written  $\text{Dom}(X)$ , is the set of (possible) values  $X$  can take.

From a Constraint Satisfaction Problem [Var,Dom,Con] to  
**random** variables with **probabilities** constrained by a graph

- The **domain** (range) of a variable  $X$ , written  $\text{Dom}(X)$ , is the set of (possible) values  $X$  can take.
- A **proposition**  $\alpha$  is an equation  $X = x$  between a variable  $X$  and a value  $x \in \text{Dom}(X)$ , or a Boolean combination of such.

From a Constraint Satisfaction Problem [Var,Dom,Con] to  
**random** variables with **probabilities** constrained by a graph

- The **domain** (range) of a variable  $X$ , written  $\text{Dom}(X)$ , is the set of (possible) values  $X$  can take.
- A **proposition**  $\alpha$  is an equation  $X = x$  between a variable  $X$  and a value  $x \in \text{Dom}(X)$ , or a Boolean combination of such.
- A proposition  $\alpha$  is assigned a probability through
  - ▶ a notion  $\models$  of a possible world  $\omega$  **satisfying**  $\alpha$ , and
  - ▶ a **measure**  $\mu$  for weighing a set of possible worlds.

## Satisfaction, measure and probability

Fix a set  $\Omega$  of **possible worlds**  $\omega$  that assign a value to each random variable, and interpret a proposition via  $\models$

$\omega \models X = x \iff \omega \text{ assigns } X \text{ the value } x$

$\omega \models \alpha \wedge \beta \iff \omega \models \alpha \text{ and } \omega \models \beta$

$\omega \models \alpha \vee \beta \iff \omega \models \alpha \text{ or } \omega \models \beta$

$\omega \models \neg \alpha \iff \omega \not\models \alpha.$

## Satisfaction, measure and probability

Fix a set  $\Omega$  of **possible worlds**  $\omega$  that assign a value to each random variable, and interpret a proposition via  $\models$

$$\begin{aligned}\omega \models X = x &\iff \omega \text{ assigns } X \text{ the value } x \\ \omega \models \alpha \wedge \beta &\iff \omega \models \alpha \text{ and } \omega \models \beta \\ \omega \models \alpha \vee \beta &\iff \omega \models \alpha \text{ or } \omega \models \beta \\ \omega \models \neg \alpha &\iff \omega \not\models \alpha.\end{aligned}$$

For finite  $\Omega$ , a **probability measure** is a function

$$\mu : Pow(\Omega) \rightarrow [0, 1]$$

such that  $\mu(\Omega) = 1$  and for any subset  $S$  of  $\Omega$ ,

$$\mu(S) = \sum_{\omega \in S} \mu(\{\omega\}).$$

## Satisfaction, measure and probability

Fix a set  $\Omega$  of **possible worlds**  $\omega$  that assign a value to each random variable, and interpret a proposition via  $\models$

$$\begin{aligned}\omega \models X = x &\iff \omega \text{ assigns } X \text{ the value } x \\ \omega \models \alpha \wedge \beta &\iff \omega \models \alpha \text{ and } \omega \models \beta \\ \omega \models \alpha \vee \beta &\iff \omega \models \alpha \text{ or } \omega \models \beta \\ \omega \models \neg \alpha &\iff \omega \not\models \alpha.\end{aligned}$$

For finite  $\Omega$ , a **probability measure** is a function

$$\mu : Pow(\Omega) \rightarrow [0, 1]$$

such that  $\mu(\Omega) = 1$  and for any subset  $S$  of  $\Omega$ ,

$$\mu(S) = \sum_{\omega \in S} \mu(\{\omega\}).$$

Given  $\mu$ , a proposition  $\alpha$  has probability

$$P(\alpha) = \mu(\{\omega \mid \omega \models \alpha\}).$$

## Tuples, distributions and the sum rule

A tuple  $X_1, \dots, X_n$  of random variables is a random variable with domain

$$\text{Dom}(X_1) \times \cdots \times \text{Dom}(X_n).$$

## Tuples, distributions and the sum rule

A tuple  $X_1, \dots, X_n$  of random variables is a random variable with domain

$$\text{Dom}(X_1) \times \cdots \times \text{Dom}(X_n).$$

A **probability distribution** on a random variable  $X$  is a function  $P_X : \text{Dom}(X) \rightarrow [0, 1]$  s.t.

$$P_X(x) = P(X = x).$$

$P_X$  is often written as  $P(X)$ , and  $P_X(x)$  as  $P(x)$ .

## Tuples, distributions and the sum rule

A tuple  $X_1, \dots, X_n$  of random variables is a random variable with domain

$$\text{Dom}(X_1) \times \cdots \times \text{Dom}(X_n).$$

A **probability distribution** on a random variable  $X$  is a function  $P_X : \text{Dom}(X) \rightarrow [0, 1]$  s.t.

$$P_X(x) = P(X = x).$$

$P_X$  is often written as  $P(X)$ , and  $P_X(x)$  as  $P(x)$ .

**sum rule**       $P(X) = \sum_Y P(X, Y)$

$$P_X(x) = \sum_{y \in \text{Dom}(Y)} P_{X,Y}(x, y) \quad \text{for } x \in \text{Dom}(X)$$

## Conditional probability

To incorporate a proposition  $\alpha$  into the background assumptions, we restrict the set  $\Omega$  of possible worlds to

$$\Omega \upharpoonright \alpha := \{\omega \in \Omega \mid \omega \models \alpha\}$$

and assuming  $\mu(\Omega \upharpoonright \alpha) \neq 0$ , map a subset  $S \subseteq \Omega \upharpoonright \alpha$  to

$$\mu^\alpha(S) := \frac{\mu(S)}{\mu(\Omega \upharpoonright \alpha)}$$

for a probability measure  $\mu^\alpha : Pow(\Omega \upharpoonright \alpha) \rightarrow [0, 1]$  on  $\Omega \upharpoonright \alpha$ .

## Conditional probability

To incorporate a proposition  $\alpha$  into the background assumptions, we restrict the set  $\Omega$  of possible worlds to

$$\Omega \upharpoonright \alpha := \{\omega \in \Omega \mid \omega \models \alpha\}$$

and assuming  $\mu(\Omega \upharpoonright \alpha) \neq 0$ , map a subset  $S \subseteq \Omega \upharpoonright \alpha$  to

$$\mu^\alpha(S) := \frac{\mu(S)}{\mu(\Omega \upharpoonright \alpha)}$$

for a probability measure  $\mu^\alpha : Pow(\Omega \upharpoonright \alpha) \rightarrow [0, 1]$  on  $\Omega \upharpoonright \alpha$ .

The **conditional probability** of  $\alpha'$  given  $\alpha$  is

$$P(\alpha' \mid \alpha) := \mu^\alpha(\Omega \upharpoonright \alpha' \wedge \alpha) = \frac{P(\alpha' \wedge \alpha)}{P(\alpha)}$$

## The product rule and Bayes' theorem

product rule

$$P(X, Y) = P(X|Y)P(Y)$$

$$P_{X,Y}(x,y) = P_X(x|Y=y)P_Y(y)$$

for  $x \in \text{Dom}(X)$ ,  $y \in \text{Dom}(Y)$

## The product rule and Bayes' theorem

product rule

$$P(X, Y) = P(X|Y)P(Y)$$

$$P_{X,Y}(x,y) = P_X(x|Y=y)P_Y(y)$$

for  $x \in \text{Dom}(X)$ ,  $y \in \text{Dom}(Y)$

As conjunction is commutative ( $\Omega \upharpoonright \alpha' \wedge \alpha = \Omega \upharpoonright \alpha \wedge \alpha'$ ),

$$P(X, Y) = P(Y, X)$$

and so the product rule yields

Bayes' theorem

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \quad \text{if } P(Y) \neq 0$$

## The product rule and Bayes' theorem

product rule

$$P(X, Y) = P(X|Y)P(Y)$$

$$P_{X,Y}(x,y) = P_X(x|Y=y)P_Y(y)$$

for  $x \in \text{Dom}(X)$ ,  $y \in \text{Dom}(Y)$

As conjunction is commutative ( $\Omega \restriction \alpha' \wedge \alpha = \Omega \restriction \alpha \wedge \alpha'$ ),

$$P(X, Y) = P(Y, X)$$

and so the product rule yields

Bayes' theorem

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \quad \text{if } P(Y) \neq 0$$

The prior probability of  $\alpha$

$$P(\alpha) = \mu(\Omega \restriction \alpha)$$

is updated by  $\alpha_o$  to the posterior probability given  $\alpha_o$

$$P(\alpha | \alpha_o) = \mu^{\alpha_o}(\Omega \restriction (\alpha \wedge \alpha_o))$$

## Why is Bayes' theorem interesting?

Form a hypothesis  $h$  given evidence  $e$  with  $P(e) \neq 0$  via Bayes

$$P(h|e) = \frac{P(e|h)P(h)}{P(e)} .$$

## Why is Bayes' theorem interesting?

Form a hypothesis  $h$  given evidence  $e$  with  $P(e) \neq 0$  via Bayes

$$P(h|e) = \frac{P(e|h)P(h)}{P(e)} .$$

We often have causal knowledge

$$P(\text{symptom} \mid \text{disease}), \quad P(\text{alarm} \mid \text{fire})$$

$$P(\text{image} = \text{car accident} \mid \text{a tree is in front of a car})$$

but want to do evidential reasoning

$$P(\text{disease} \mid \text{symptom}), \quad P(\text{fire} \mid \text{alarm})$$

$$P(\text{a tree is in front of a car} \mid \text{image} = \text{car accident})$$

## Tuples and the chain rule

Recall: a tuple  $X_1, \dots, X_n$  of random variables is a random variable.

Let us write

$$X_{1:n} \text{ for } X_1, \dots, X_n$$

## Tuples and the chain rule

Recall: a tuple  $X_1, \dots, X_n$  of random variables is a random variable.

Let us write

$$X_{1:n} \text{ for } X_1, \dots, X_n$$

and apply the product rule repeatedly for

$$\begin{aligned} P(X_{1:n}) &= P(X_n | X_{1:n-1})P(X_{1:n-1}) \\ &= P(X_n | X_{1:n-1})P(X_{n-1} | X_{1:n-2})P(X_{1:n-2}) \\ &= \cdots \\ &= \prod_{i=1}^n P(X_i | X_{1:i-1}) \quad \text{chain rule} \end{aligned}$$

with  $X_{1:0}$  as the empty tuple and  $P(X_1 | X_{1:0}) = P(X_1)$ .

## Simplifying the chain rule via conditional independence

Choose a sub-tuple  $\text{parents}(X_i)$  of  $X_{1:i-1}$  such that

$$P(X_i | X_{1:i-1}) = P(X_i | \text{parents}(X_i)) .$$

## Simplifying the chain rule via conditional independence

Choose a sub-tuple  $\text{parents}(X_i)$  of  $X_{1:i-1}$  such that

$$P(X_i | X_{1:i-1}) = P(X_i | \text{parents}(X_i)).$$

$X$  is **independent** of  $Y$  **given**  $Z$ , written  $X \perp\!\!\!\perp Y | Z$ ,

$$P(X | Y, Z) = P(X | Z)$$

## Simplifying the chain rule via conditional independence

Choose a sub-tuple  $\text{parents}(X_i)$  of  $X_{1:i-1}$  such that

$$P(X_i | X_{1:i-1}) = P(X_i | \text{parents}(X_i)).$$

$X$  is **independent** of  $Y$  **given**  $Z$ , written  $X \perp\!\!\!\perp Y | Z$ ,

$$P(X | Y, Z) = P(X | Z)$$

i.e. for all  $x \in \text{Dom}(X)$ ,  $y \in \text{Dom}(Y)$ , and  $z \in \text{Dom}(Z)$ ,

$$P(X = x | Y = y \wedge Z = z) = P(X = x | Z = z)$$

— knowing  $Y$ 's value says nothing about  $X$ 's value, given  $Z$ 's value.

## Simplifying the chain rule via conditional independence

Choose a sub-tuple  $\text{parents}(X_i)$  of  $X_{1:i-1}$  such that

$$P(X_i | X_{1:i-1}) = P(X_i | \text{parents}(X_i)).$$

$X$  is **independent** of  $Y$  **given**  $Z$ , written  $X \perp\!\!\!\perp Y | Z$ ,

$$P(X | Y, Z) = P(X | Z)$$

i.e. for all  $x \in \text{Dom}(X)$ ,  $y \in \text{Dom}(Y)$ , and  $z \in \text{Dom}(Z)$ ,

$$P(X = x | Y = y \wedge Z = z) = P(X = x | Z = z)$$

— knowing  $Y$ 's value says nothing about  $X$ 's value, given  $Z$ 's value.

Note

$$\begin{aligned} X \perp\!\!\!\perp Y | Z &\iff P(X, Y | Z) = P(X | Z)P(Y | Z) \\ &\iff Y \perp\!\!\!\perp X | Z \end{aligned}$$

Totally order the variables of interest

$$X_1 < X_2 < \cdots < X_n$$

and for each  $i$  from 1 to  $n$ , choose  $\text{parents}(X_i)$  from  $X_{1:i-1}$  s.t.

$$P(X_i | X_{1:i-1}) = P(X_i | \text{parents}(X_i)) \quad (\dagger)$$

## Belief networks

Totally order the variables of interest

$$X_1 < X_2 < \cdots < X_n$$

and for each  $i$  from 1 to  $n$ , choose  $\text{parents}(X_i)$  from  $X_{1:i-1}$  s.t.

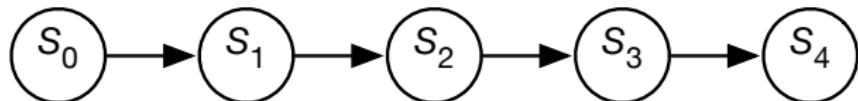
$$P(X_i | X_{1:i-1}) = P(X_i | \text{parents}(X_i)) \quad (\dagger)$$

A **belief network** consists of:

- a directed acyclic graph with nodes = random variables, and an arc from the parents of each node into that node
- a domain for each random variable
- conditional probability tables for each variable given its parents (for a probability distribution respecting  $(\dagger)$ )

## Example: Markov chain

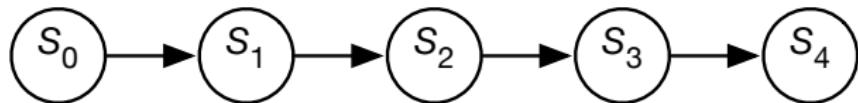
A **Markov chain** is a special sort of belief network:



What probabilities need to be specified?

## Example: Markov chain

A **Markov chain** is a special sort of belief network:

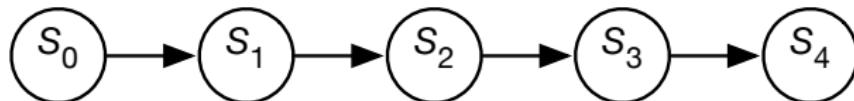


What probabilities need to be specified?

- $P(S_0)$  specifies initial conditions
- $P(S_{t+1}|S_t)$  specifies the dynamics

## Example: Markov chain

A **Markov chain** is a special sort of belief network:



What probabilities need to be specified?

- $P(S_0)$  specifies initial conditions
- $P(S_{t+1}|S_t)$  specifies the dynamics

What independence assumptions are made?

$$P(S_{t+1}|S_{0:t}) = P(S_{t+1}|S_t)$$

$S_t$  represents the **state** at time  $t$ , capturing everything about the past ( $< t$ ) that can affect the future ( $> t$ )

The future is independent of the past given the present.

## Two elaborations

In a **stationary Markov chain**,

$$\text{Dom}(S_i) = \text{Dom}(S_0) \text{ and } P(S_{i+1}|S_i) = P(S_1|S_0) \text{ for all } i \geq 0$$

so it is enough to specify  $P(S_0)$  and  $P(S_1|S_0)$ .

- Simple model, easy to specify
- The network can extend indefinitely

## Two elaborations

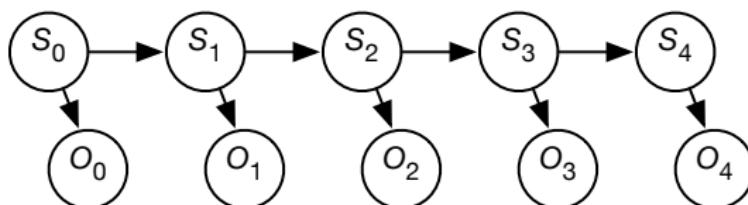
In a **stationary Markov chain**,

$$\text{Dom}(S_i) = \text{Dom}(S_0) \text{ and } P(S_{i+1}|S_i) = P(S_1|S_0) \text{ for all } i \geq 0$$

so it is enough to specify  $P(S_0)$  and  $P(S_1|S_0)$ .

- Simple model, easy to specify
- The network can extend indefinitely

A **Hidden Markov Model (HMM)** is a belief network of the form



- $P(S_0)$  specifies initial conditions
- $P(S_{i+1}|S_i)$  specifies the dynamics
- $P(O_i|S_i)$  specifies the sensor model

## Naive Bayes Classifier

Problem: classify on the basis of features  $F_i$

$$P(\text{Class} | F_{1:n}) = \frac{P(F_{1:n} | \text{Class}) P(\text{Class})}{P(F_{1:n})}$$

## Naive Bayes Classifier

Problem: classify on the basis of features  $F_i$

$$P(\text{Class}|F_{1:n}) = \frac{P(F_{1:n}|\text{Class})P(\text{Class})}{P(F_{1:n})}$$

Assume  $F_i$  are independent of each other given  $\text{Class}$

$$P(F_{1:n}|\text{Class}) = \prod_i P(F_i|\text{Class})$$

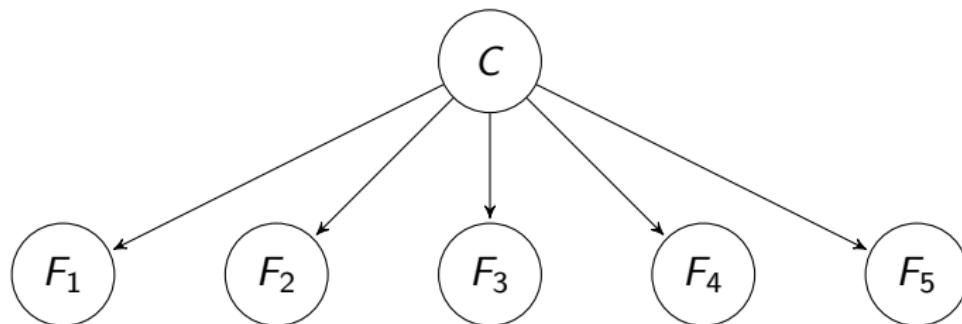
## Naive Bayes Classifier

Problem: classify on the basis of features  $F_i$

$$P(\text{Class}|F_{1:n}) = \frac{P(F_{1:n}|\text{Class})P(\text{Class})}{P(F_{1:n})}$$

Assume  $F_i$  are independent of each other given  $\text{Class}$

$$P(F_{1:n}|\text{Class}) = \prod_i P(F_i|\text{Class})$$



Assume the values of features  $F_i$  are predictable given a class.

Requires  $P(\text{Class})$  and  $P(F_i|\text{Class})$  for each  $F_i$

## Learning Probabilities

| $F_1$    | $F_2$    | $F_3$    | $F_4$    | $C$ | <i>Count</i> |
|----------|----------|----------|----------|-----|--------------|
| :        | :        | :        | :        | :   | :            |
| <i>t</i> | <i>f</i> | <i>t</i> | <i>t</i> | 1   | 40           |
| <i>t</i> | <i>f</i> | <i>t</i> | <i>t</i> | 2   | 10           |
| <i>t</i> | <i>f</i> | <i>t</i> | <i>t</i> | 3   | 50           |
| :        | :        | :        | :        | :   | :            |

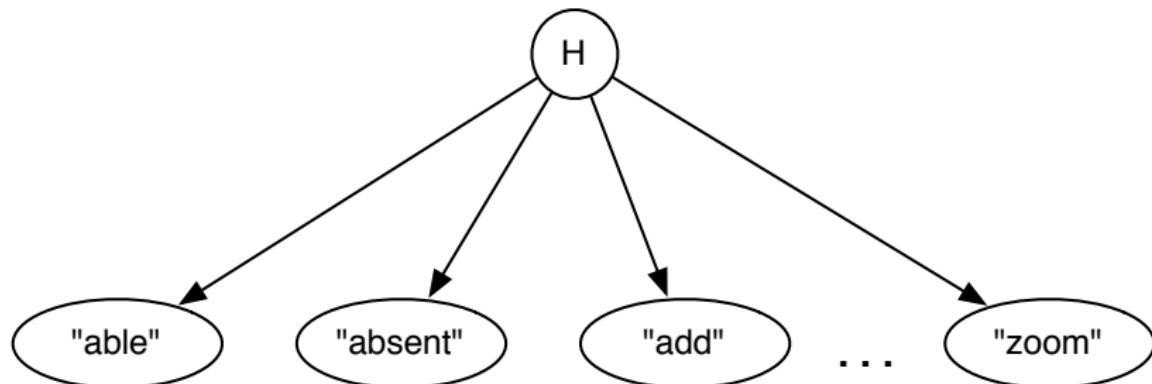
## Learning Probabilities

| $F_1$ | $F_2$ | $F_3$ | $F_4$ | $C$ | <i>Count</i> |
|-------|-------|-------|-------|-----|--------------|
| :     | :     | :     | :     | :   | :            |
| $t$   | $f$   | $t$   | $t$   | 1   | 40           |
| $t$   | $f$   | $t$   | $t$   | 2   | 10           |
| $t$   | $f$   | $t$   | $t$   | 3   | 50           |
| :     | :     | :     | :     | :   | :            |

$$P(C=c) = \frac{\sum_{\omega \models C=c} Count(\omega)}{\sum_{\omega} Count(\omega)}$$
$$P(F_k = b | C=c) = \frac{\sum_{\omega \models C=c \wedge F_k=b} Count(\omega)}{\sum_{\omega \models C=c} Count(\omega)}$$

with pseudo-counts (Cromwell's rule)

# Help System



- The domain of  $H$  is the set of all help pages.  
The observations are the words in the query.
- What probabilities are needed?  
What pseudo-counts and counts are used?  
What data can be used to learn from?

## Constructing a belief network

To represent a domain in a belief network, we need to consider:

- What are the relevant variables?
  - ▶ What will you observe?
  - ▶ What would you like to find out (query)?
  - ▶ What other features make the model simpler?
- What values should these variables take?
- What is the relationship between them?

Express this in terms of a directed graph, representing how each variable  $X_i$  is generated from its predecessors  $X_{1:i-1}$ .

The parents of  $X$  are variables on which  $X$  directly depends

  - ▶  $X$  is independent of its non-descendants given its parents.
- How does the value of each variable depend on its parents?

This is expressed in terms of the conditional probabilities.

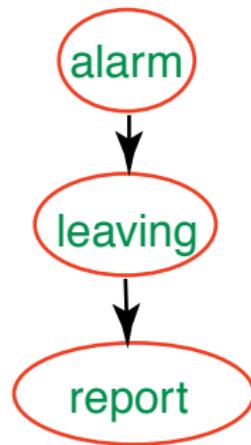
## Example: fire alarm belief network

Variables:

- **Fire**: there is a fire in the building
- **Tampering**: someone has been tampering with the fire alarm
- **Smoke**: what appears to be smoke is coming from an upstairs window
- **Alarm**: the fire alarm goes off
- **Leaving**: people are leaving the building *en masse*.
- **Report**: a colleague says that people are leaving the building *en masse*. (A noisy sensor for leaving.)

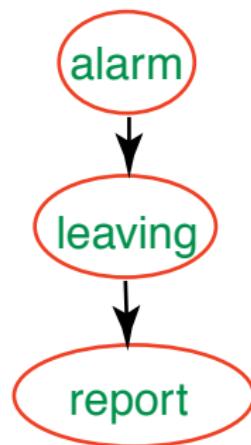
## Head-to-tail: Chain

- *alarm* and *report* are

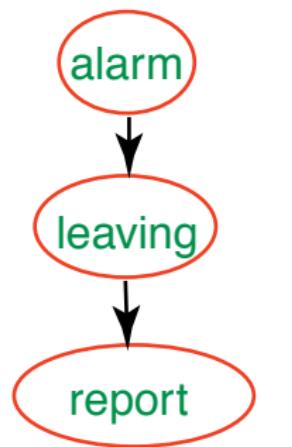


## Head-to-tail: Chain

- *alarm* and *report* are dependent



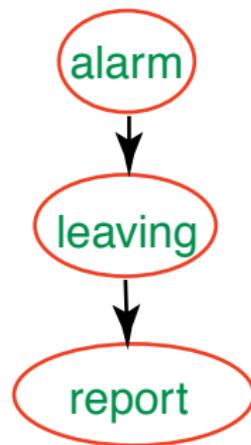
## Head-to-tail: Chain



- *alarm* and *report* are dependent
- *alarm* and *report* are given

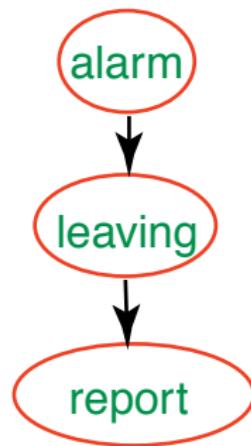
*leaving*

## Head-to-tail: Chain



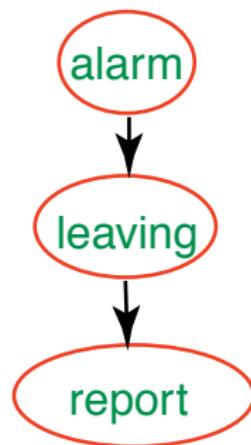
- *alarm* and *report* are dependent
- *alarm* and *report* are independent given *leaving*

## Head-to-tail: Chain



- *alarm* and *report* are dependent
- *alarm* and *report* are independent given *leaving*
- Intuitively, the only way that the *alarm* affects *report* is by affecting *leaving*.

## Head-to-tail: Chain

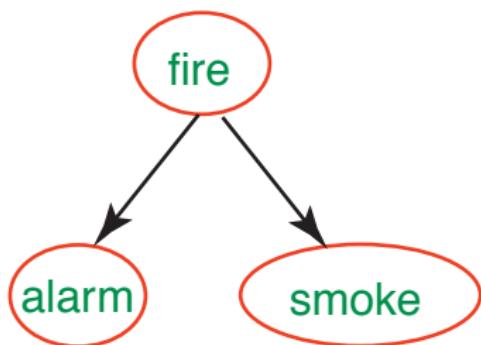


- *alarm* and *report* are dependent
- *alarm* and *report* are independent given *leaving*
- Intuitively, the only way that the *alarm* affects *report* is by affecting *leaving*.

$$\begin{aligned} P(\text{report, alarm} \mid \text{leaving}) &= \frac{P(\text{report, alarm, leaving})}{P(\text{leaving})} \\ &= \frac{P(\text{alarm})P(\text{leaving} \mid \text{alarm})P(\text{report} \mid \text{leaving})}{P(\text{leaving})} \quad \text{net} \\ &= \frac{P(\text{alarm, leaving})}{P(\text{leaving})} P(\text{report} \mid \text{leaving}) \quad \text{product} \\ &= P(\text{alarm} \mid \text{leaving})P(\text{report} \mid \text{leaving}) \quad \text{for } \perp\!\!\!\perp \end{aligned}$$

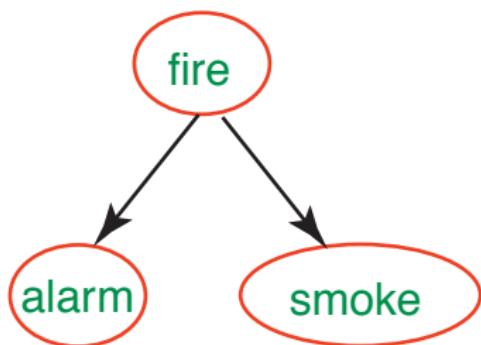
## Tail-to-tail: Common ancestors

- *alarm* and *smoke* are



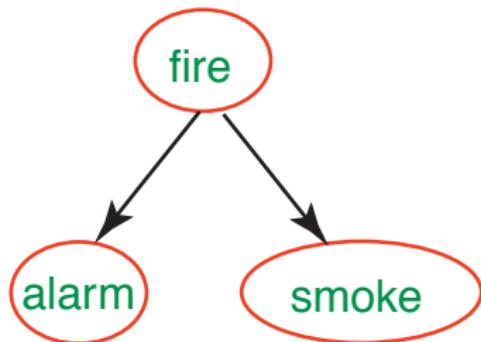
## Tail-to-tail: Common ancestors

- *alarm* and *smoke* are dependent



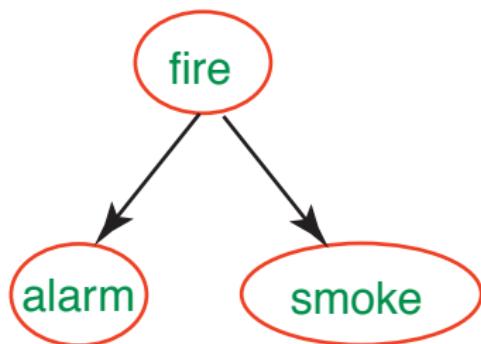
## Tail-to-tail: Common ancestors

- *alarm* and *smoke* are dependent
- *alarm* and *smoke* are given *fire*

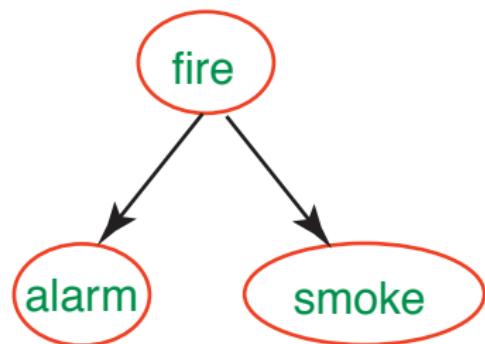


## Tail-to-tail: Common ancestors

- *alarm* and *smoke* are dependent
- *alarm* and *smoke* are independent given *fire*



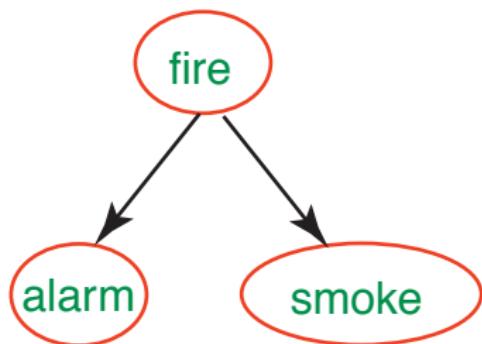
## Tail-to-tail: Common ancestors



- *alarm* and *smoke* are dependent
- *alarm* and *smoke* are independent given *fire*
- Intuitively, *fire* can **explain** *alarm* and *smoke*; learning one can affect the other by changing your belief in *fire*.

## Tail-to-tail: Common ancestors

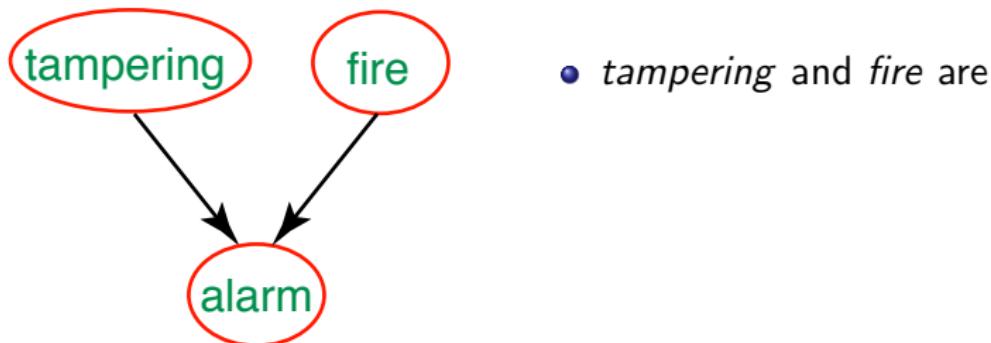
- *alarm* and *smoke* are dependent
- *alarm* and *smoke* are independent given *fire*
- Intuitively, *fire* can **explain** *alarm* and *smoke*; learning one can affect the other by changing your belief in *fire*.



$\text{smoke} \perp\!\!\!\perp \text{alarm} \mid \text{fire}$

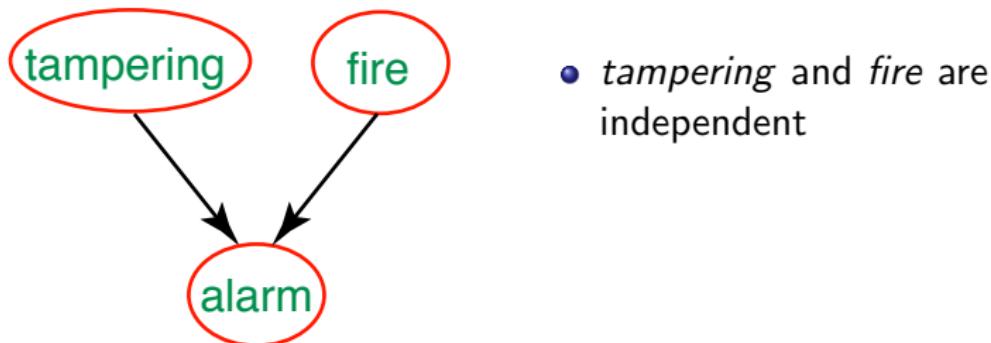
$$\begin{aligned} P(\text{smoke}, \text{alarm} \mid \text{fire}) &= \frac{P(\text{smoke, alarm, fire})}{P(\text{fire})} \\ &= \frac{P(\text{fire})P(\text{alarm} \mid \text{fire})P(\text{smoke} \mid \text{fire})}{P(\text{fire})} \quad \text{net} \\ &= P(\text{alarm} \mid \text{fire})P(\text{smoke} \mid \text{fire}) \quad \text{for } \perp\!\!\!\perp \end{aligned}$$

## Head-to-head: Common descendants



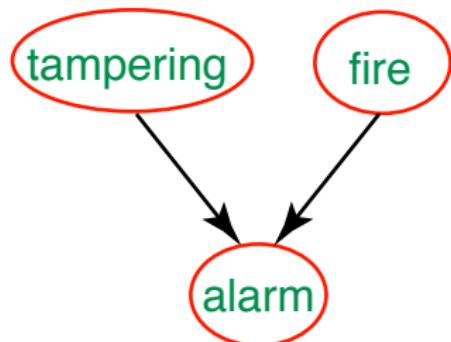
- *tampering* and *fire* are

## Head-to-head: Common descendants



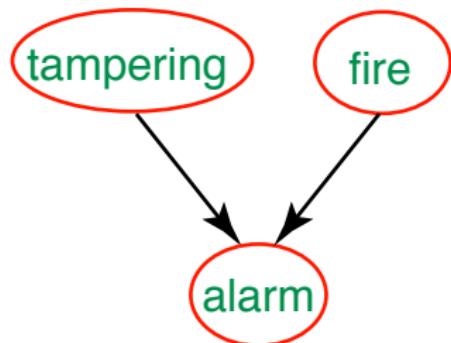
- *tampering* and *fire* are independent

## Head-to-head: Common descendants



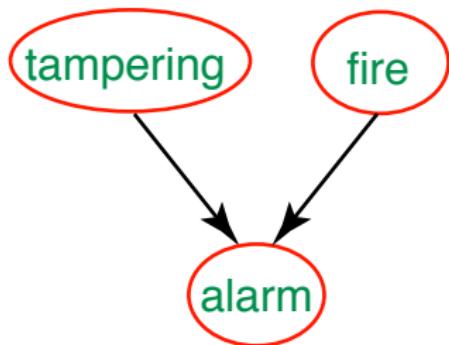
- *tampering* and *fire* are independent
- *tampering* and *fire* are given *alarm*

## Head-to-head: Common descendants



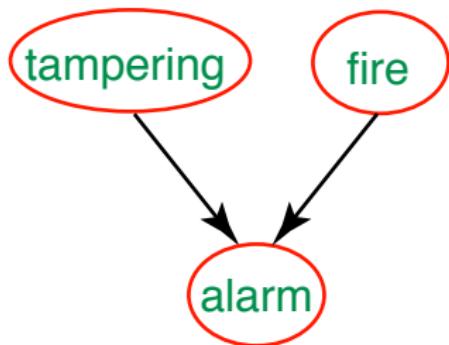
- *tampering* and *fire* are independent
- *tampering* and *fire* are dependent given *alarm*

## Head-to-head: Common descendants



- *tampering* and *fire* are independent
- *tampering* and *fire* are dependent given *alarm*
- Intuitively, *tampering* can **explain away** *fire*

## Head-to-head: Common descendants



- *tampering* and *fire* are independent
- *tampering* and *fire* are dependent given *alarm*
- Intuitively, *tampering* can **explain away** *fire*

$$P(\text{fi} = 1 \mid \text{am} = 1) > P(\text{fi} = 1 \mid \text{am} = 1 \wedge \text{tg} = 1)$$

for  $P(\text{tg} = 0) = 0.9$        $P(\text{fi} = 0) = 0.9$

$$P(\text{am} = 1 \mid \text{tg} = 1 \wedge \text{fi} = 1) = 0.95$$

$$P(\text{am} = 1 \mid \text{tg} = 1 \wedge \text{fi} = 0) = 0.5$$

$$P(\text{am} = 1 \mid \text{tg} = 0 \wedge \text{fi} = 1) = 0.9$$

$$P(\text{am} = 1 \mid \text{tg} = 0 \wedge \text{fi} = 0) = 0.1$$

$$P(\text{fi} = 1 | \text{am} = 1) \approx 0.418$$

$$P(\text{fi} = 1 | \text{am} = 1) = \frac{P(\text{am} = 1 | \text{fi} = 1) P(\text{fi} = 1)}{P(\text{am} = 1)} \quad \text{Bayes}$$

$$\begin{aligned} P(\text{am} = 1 | \text{fi} = 1) &= \sum_{tg} \underbrace{P(\text{am} = 1, tg | \text{fi} = 1)}_{\substack{\text{sum} \\ P(\text{am} = 1 | tg, \text{fi} = 1) \underbrace{P(tg | \text{fi} = 1)}_{\substack{\text{product} \\ P(tg)}}}} \\ &\quad P(tg) \quad \text{net} \end{aligned}$$

$$\begin{aligned} P(\text{am} = 1) &= \sum_{tg} \sum_{fi} \underbrace{P(\text{am} = 1, tg, fi)}_{\substack{\text{sum} \\ P(tg)P(fi)P(\text{am} = 1 | tg, fi)}} \\ &\quad \text{net} \end{aligned}$$

$$P(\text{fi} = 1 | \text{am} = 1, \text{tg} = 1) \approx 0.174$$

$$P(\text{fi} = 1 | \text{am} = 1, \text{tg} = 1) = \frac{P(\text{am} = 1 | \text{fi} = 1, \text{tg} = 1) \overbrace{P(\text{fi} = 1 | \text{tg} = 1)}^{\text{net}}}{P(\text{am} = 1 | \text{tg} = 1)}$$

Bayes

$$P(\text{am} = 1 | \text{tg} = 1) = \sum_{\text{fi}} \underbrace{P(\text{am} = 1, \text{fi} | \text{tg} = 1)}_{P(\text{am} = 1 | \text{fi}, \text{tg} = 1) \underbrace{P(\text{fi} | \text{tg} = 1)}_{\text{net}}} \quad \text{sum}$$
$$P(\text{am} = 1 | \text{fi}, \text{tg} = 1) \underbrace{P(\text{fi} | \text{tg} = 1)}_{\text{product}}$$

sum rule  $P(X) = \sum_Y P(X, Y) = \sum_y P(X, Y = y)$

$$P_X(x) = \sum_{y \in \text{Dom}(Y)} P_{X,Y}(x,y) \quad \text{for } x \in \text{Dom}(X)$$

sum rule  $P(X) = \sum_Y P(X, Y) = \sum_y P(X, Y = y)$

$$P_X(x) = \sum_{y \in \text{Dom}(Y)} P_{X,Y}(x,y) \quad \text{for } x \in \text{Dom}(X)$$

$$P(X = x) = \sum_{y \in \text{Dom}(Y)} P(X = x \wedge Y = y)$$

sum rule  $P(X) = \sum_Y P(X, Y) = \sum_y P(X, Y = y)$

$$P_X(x) = \sum_{y \in \text{Dom}(Y)} P_{X,Y}(x,y) \quad \text{for } x \in \text{Dom}(X)$$

$$P(X = x) = \sum_{y \in \text{Dom}(Y)} P(X = x \wedge Y = y)$$

$$\mu(\{\omega \in \Omega \mid \omega \models X = x\}) = \sum_{y \in \text{Dom}(Y)} \mu(\{\omega \in \Omega \mid \omega \models X = x \wedge Y = y\})$$

sum rule  $P(X) = \sum_Y P(X, Y) = \sum_y P(X, Y = y)$

$$P_X(x) = \sum_{y \in \text{Dom}(Y)} P_{X,Y}(x, y) \quad \text{for } x \in \text{Dom}(X)$$

$$P(X = x) = \sum_{y \in \text{Dom}(Y)} P(X = x \wedge Y = y)$$

$$\mu(\{\omega \in \Omega \mid \omega \models X = x\}) = \sum_{y \in \text{Dom}(Y)} \mu(\{\omega \in \Omega \mid \omega \models X = x \wedge Y = y\})$$

$$\begin{aligned} \mu(\{\omega \in \Omega \mid \omega(X) = x\}) &= \sum_{y \in \text{Dom}(Y)} \mu(\{\omega \in \Omega \mid \omega(X) = x \text{ and} \\ &\qquad \qquad \qquad \omega(Y) = y\}) \end{aligned}$$

**sum rule**  $P(X) = \sum_Y P(X, Y) = \sum_y P(X, Y = y)$

$$P_X(x) = \sum_{y \in \text{Dom}(Y)} P_{X,Y}(x,y) \quad \text{for } x \in \text{Dom}(X)$$

$$P(X = x) = \sum_{y \in \text{Dom}(Y)} P(X = x \wedge Y = y)$$

$$\mu(\{\omega \in \Omega \mid \omega \models X = x\}) = \sum_{y \in \text{Dom}(Y)} \mu(\{\omega \in \Omega \mid \omega \models X = x \wedge Y = y\})$$

$$\begin{aligned} \mu(\{\omega \in \Omega \mid \omega(X) = x\}) &= \sum_{y \in \text{Dom}(Y)} \mu(\{\omega \in \Omega \mid \omega(X) = x \text{ and} \\ &\qquad \qquad \qquad \omega(Y) = y\}) \end{aligned}$$

From additivity of  $\mu$  (for finite  $\Omega$ )

$$\mu(S) = \sum_{\omega \in S} \mu(\{\omega\})$$

## Joint probability from a table

|          | $y_1$         | $y_2$         | $\cdots$ | $y_c$         |
|----------|---------------|---------------|----------|---------------|
| $x_1$    | $P(x_1, y_1)$ | $P(x_1, y_2)$ | $\cdots$ | $P(x_1, y_c)$ |
| $x_2$    | $P(x_2, y_1)$ | $P(x_2, y_2)$ | $\cdots$ | $P(x_2, y_c)$ |
| $\vdots$ |               |               |          |               |
| $x_r$    | $P(x_r, y_1)$ | $P(x_r, y_2)$ | $\cdots$ | $P(x_r, y_c)$ |

## Joint probability from a table

|                                       |          | $y_1$         | $y_2$         | $\cdots$ | $y_c$         |
|---------------------------------------|----------|---------------|---------------|----------|---------------|
| margin<br>$P(x_i) = \sum_y P(x_i, y)$ | $x_1$    | $P(x_1, y_1)$ | $P(x_1, y_2)$ | $\cdots$ | $P(x_1, y_c)$ |
|                                       | $x_2$    | $P(x_2, y_1)$ | $P(x_2, y_2)$ | $\cdots$ | $P(x_2, y_c)$ |
|                                       | $\vdots$ |               |               |          |               |
|                                       | $x_r$    | $P(x_r, y_1)$ | $P(x_r, y_2)$ | $\cdots$ | $P(x_r, y_c)$ |

Wikipedia on *Marginal distribution*

*Marginal variables are those variables in the subset of variables being retained. These concepts are “marginal” because they can be found by summing values in a table along rows or columns, and writing the sum in the margins of the table.*

## Sum rule as marginalisation

$$P(X) = \sum_Y P(X, Y)$$

## Sum rule as marginalisation

joint probability  $P(X, Y)$

$$P(X) = \sum_Y P(X, Y)$$

## Sum rule as marginalisation

joint probability  $P(X, Y)$

$$P(X) = \sum_Y P(X, Y)$$

marginal probability  $P(X)$

## Sum rule as marginalisation

$$P(X) = \sum_Y \text{joint probability } P(X, Y)$$

marginal probability  $P(X)$       marginalising out  $Y$

## Sum rule as marginalisation

joint probability  $P(X, Y)$

$$P(X) = \sum_Y P(X, Y)$$

marginal probability  $P(X)$

marginalising out  $Y$

$\approx$  eliminating  $Y$

## Sum rule as marginalisation

joint probability  $P(X, Y)$

$$P(X) = \sum_Y P(X, Y)$$

marginal probability  $P(X)$

marginalising out  $Y$

$\approx$  eliminating  $Y$

nuisance variable  $Y$

## Sum rule as marginalisation

$$P(X) = \sum_Y P(X, Y)$$

joint probability  $P(X, Y)$

marginal probability  $P(X)$       marginalising out  $Y$

                                            ≈ eliminating  $Y$

                                            nuisance variable  $Y$

We'll define  $P(X|Y)$  so that

$P(X) = \text{expected value of } P(X|Y) \text{ over } Y$

## Sum rule as marginalisation

|                             |                             |
|-----------------------------|-----------------------------|
|                             | joint probability $P(X, Y)$ |
| $P(X)$                      | $= \sum_Y P(X, Y)$          |
| marginal probability $P(X)$ | marginalising out $Y$       |
|                             | $\approx$ eliminating $Y$   |
|                             | nuisance variable $Y$       |

We'll define  $P(X|Y)$  so that

$$P(X) = \text{expected value of } P(X|Y) \text{ over } Y$$

$$= \sum_Y P(X|Y)P(Y)$$

## Sum rule as marginalisation

$$P(X) = \sum_Y P(X, Y)$$

joint probability  $P(X, Y)$

marginal probability  $P(X)$       marginalising out  $Y$

                                            ≈ eliminating  $Y$

                                            nuisance variable  $Y$

We'll define  $P(X|Y)$  so that

$$\begin{aligned} P(X) &= \text{expected value of } P(X|Y) \text{ over } Y \\ &= \sum_Y P(X|Y)P(Y) \\ P(x) &= \sum_y P(x|y)P(y) \\ &= \mathbb{E}_y[P(x|y)] \end{aligned}$$