Exam Hints

Topics:

1. A\*
2. Q-Learning (Almost entirely on MDP)
3. Bayesian Networks (Slides 56-57)
4. CSPs (V,D,C)
5. Modelling SAT as CSP

Q1)

* Should only do in Prolog
* Search (probably A\*) - write in Prolog

Q2)

* MDP pseudocode or just use formulas
  + Memorise equations
  + Do a couple of iterations of MDP (connect with some Q-learning)
  + Possible some logic

Q3)

* Bayesian Networks
  + Simple equations
  + Probability question from slides 55-57
  + **Don’t look at** slide 59 onwards
  + Only part of a question

|  |  |
| --- | --- |
| **Could be on paper** | **Not on paper** |
| * The halting problem * Universal Turing machines * SAT CSP * Constraint Satisfaction (3 colour) * Q3 - Datalog * What is an interpretation? () * Horn clauses (False keyword stuff) * Abduction (interpretation / logical consequence) | * Marr * Fixed clauses |

Q-learning - If you don’t have the probability and reward, you learn them from the environment.

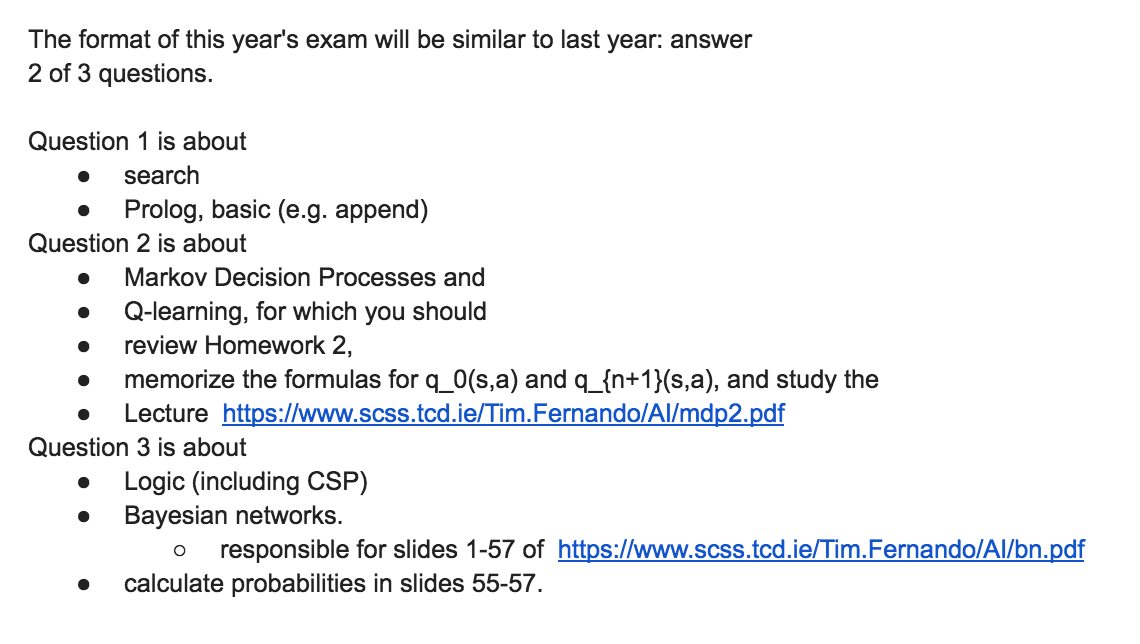
Constraint Satisfaction Problems (VDC)

CSP ➝ Specifically set

2017 Q3e) probably not as halting problem (program and input domain) in infinite.

CSP problems tend to be solvable although some are polynomial.

From Timbolic Programming himself:



DISCLAIMER!

FORMULAE MAY BE INCORRECT!

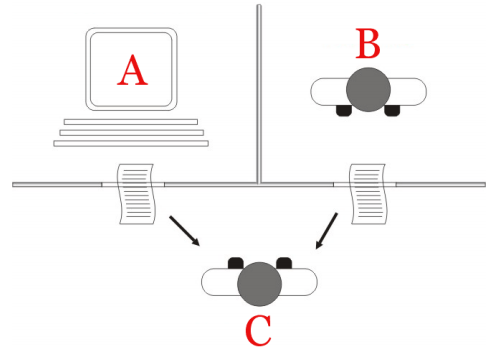
Please check the original notes at <https://www.scss.tcd.ie/Tim.Fernando/AI/> if something appears wrong, and send me (Ciarán Ingle) a message if it is.

Topic 1: Introduction

01 Introduction

**The Turing test**:

Can C tell A from B?



### ELIZA

Used **pattern-matching** and **substitution** to fake understanding.

**The ELIZA effect**:

Humans are inclined to see computers as humans.

**AI complete**:

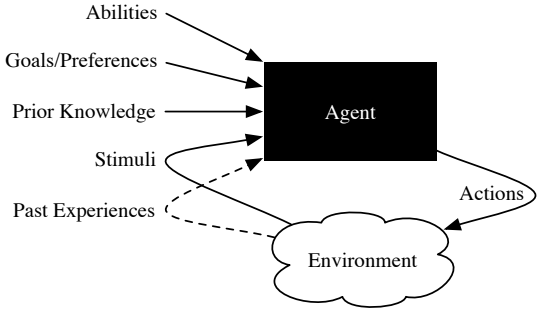
An AI problem is AI complete if any AI problem is mechanically reducible to it. (It is at least as hard as any other problem.)

The difficulty of these computational problems is equivalent to that of solving the central intelligence problem - making computers as intelligent as people.

AI complete problems are too difficult to be solved by the use of conventional algorithms.

*e.g. Natural Language Understanding*

### Locating Intelligence (Black Box)



**Intelligence**:

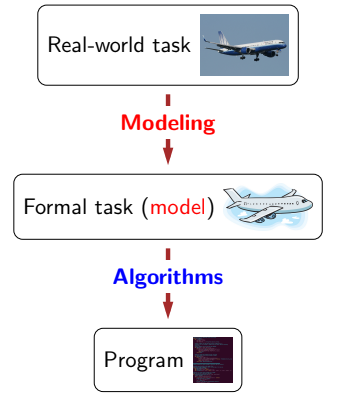
(Abilities, Goals, …, Experiences) ➝ action

**Turing Test**:

What to say ➝ What to do

### Agent & Environment

|  |  |
| --- | --- |
| **Agent** | **Environment** |
| Program  Cognitive revolution  Hard-wired  Rationalist  Nativist  Innate  Nature | data  Big Data  experienced  empiricist  behaviorist  tabula rasa  nurture |
| Turing machine & specialised automaton | Learning from environment:  Trial & error: “data as oil” |
| **Moving Target**:  Changing agent and environment (e.g. change in state). | |



Unstructured information ⟶ actionable knowledge

|  |  |
| --- | --- |
| **Traditional Approach** | **Machine Learning Approach** |
|  |  |
| **✘ Complexity becomes unwieldy.** |  |

### Machine Learning



02 Challenges to Logic

Limits on:

* **Truth**  
  Liar’s Paradox: “I am lying.”
* **Sets / membership**  
  Russel set
* **Countability**  
  Cantor:
* **Change**  
  Sorites: heap (minus one grain)
* **Computability**Turing: The Halting Problem

### Tolerance & Sorites Chains

A unary relation is **tolerant** up to if:

**Example 1**:

is   
 is

**Example 2**:

is   
 is

**Example 3**:

is ,   
 is

A **Sorites chain** is a sequence such that holds of **but not** , even though for .

# The Halting Problem

Given a program and data , return either 0 or 1, with 1 indicating that halts on input .

**Theorem (Turing)**:

No Turing machine (TM) computes the halting problem (HP).

### Proof of Uncomputability

Given a Turing machine that takes two arguments, we show that does not compute the halting problem by defining another Turing machine such that:

Let

and notice that

(From our definition of the )

(From our definition of )

### Semi-Solvability of the Halting Problem

There is a Turing machine that meets the positive part of the halting problem (looping when the HP asks for 0), in view of the existence of a **universal Turing machine**.

A **universal Turing machine** is a Turing machine that runs on :

for any given TM and data

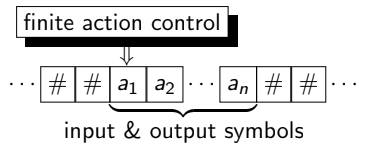
A Turing machine can be quite complex. Simple instances for which halting is no problem are **finite state machines**.

03 Finite State Machines

### Church-Turing Thesis

**Church-Turing thesis**:

Program ≈ Turing machine.



### Finite State Machines (FSMs)

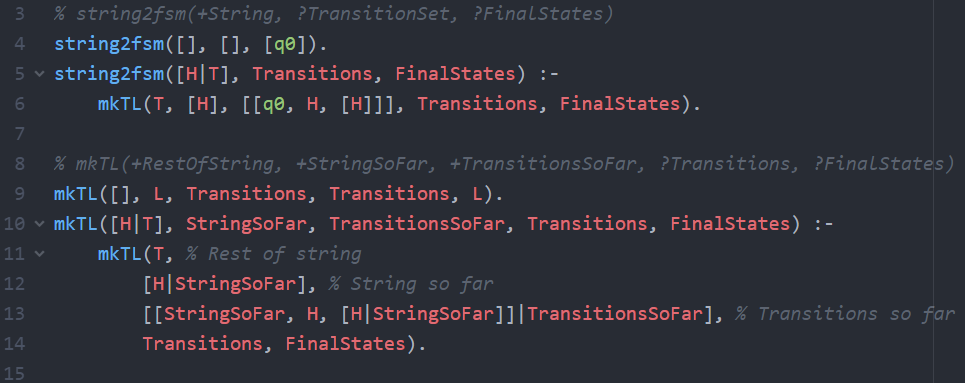
A **finite state machine**  is a triple where:

* is a list of triples such that may, at state and seeing symbol , transition to state .
* is a list of ’s final (accepting) states.
* is ’s initial state.

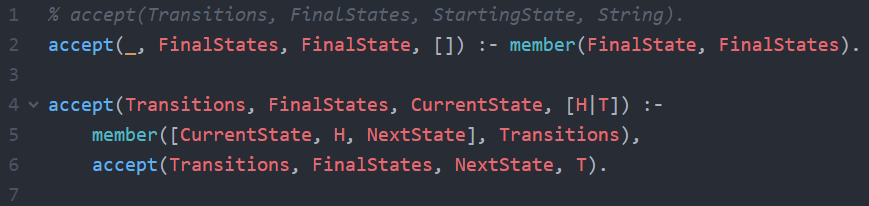
Example:

|  |
| --- |
| Trans = [[q0,a,q0], [q0,b,q1], [q1,b,q1]]  Final = [q1]  Q0 = q0 |

### Encoding Strings as Lists



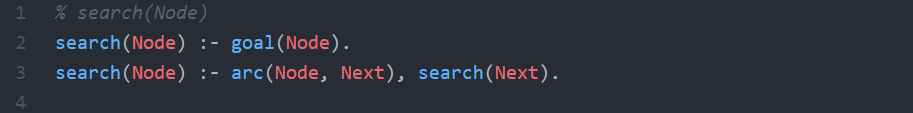
### Accepting Strings

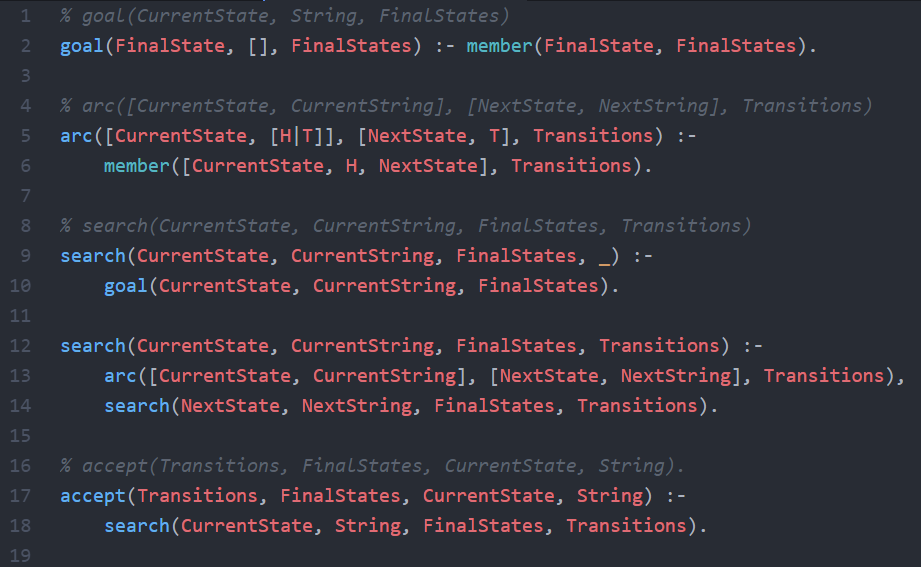


Topic 2: Search

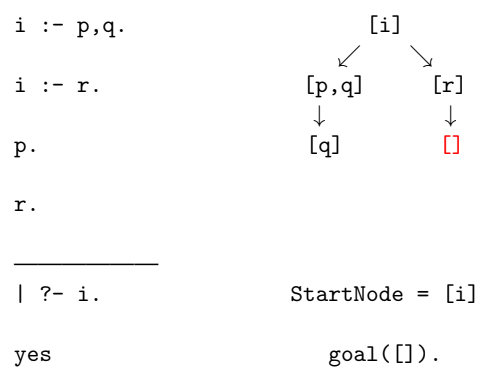
204 Search

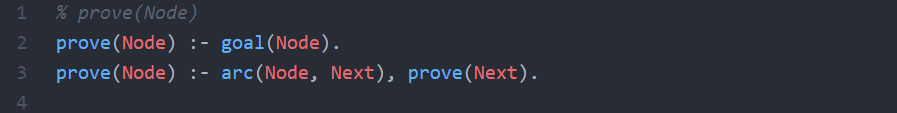
Given goal/1 and arc/2:



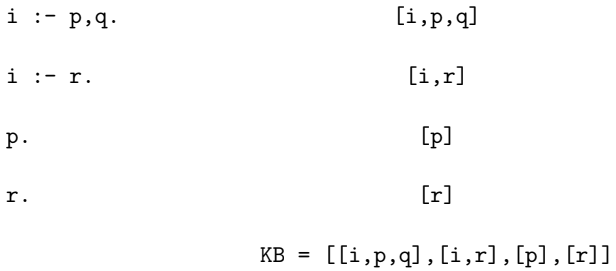


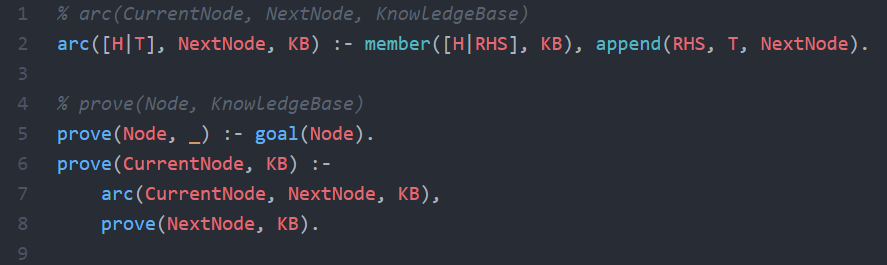
# Prolog as Search



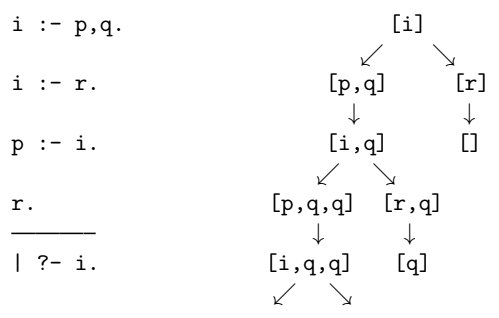


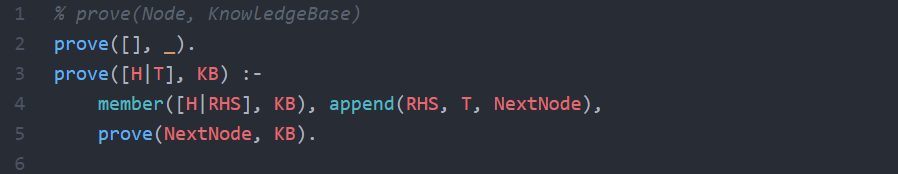
### Knowledge Bases (KBs) & arc/3





### Non-Termination





05 Frontier Search

### Determinisation (Eliminating Choice)

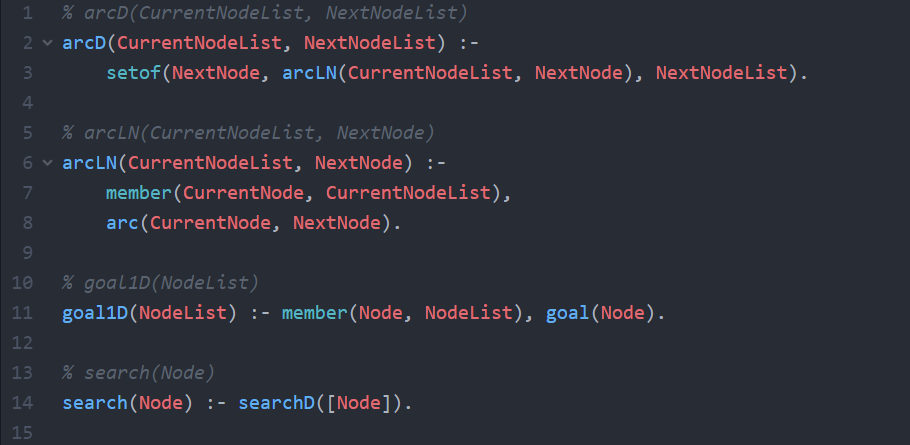
A FSM such that:

is a **deterministic finite automaton (DFA)**.

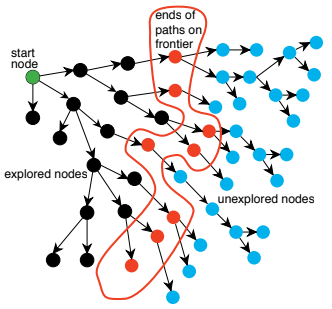
A **non-deterministic finite automaton (NFA)** is a FSM.

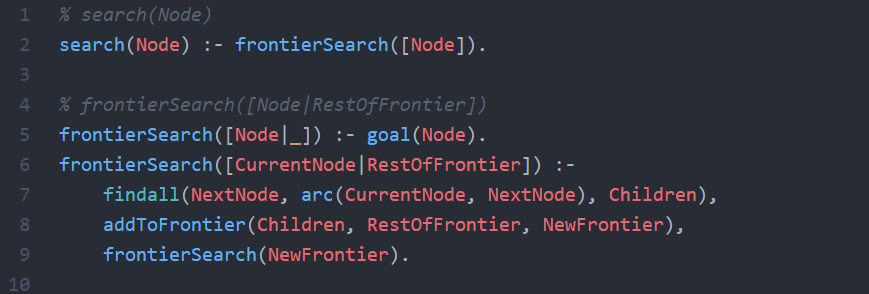
**Fact**: Every NFA has a DFA accepting the same language.

**Proof**: Subset (power set) construction.

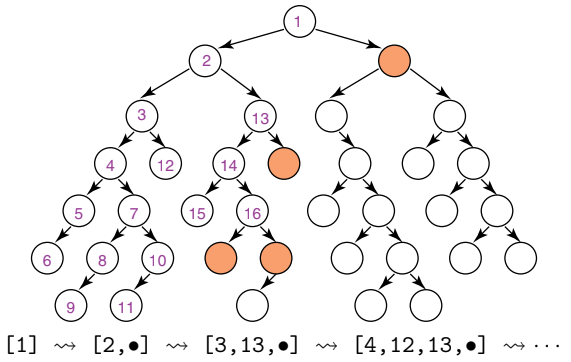


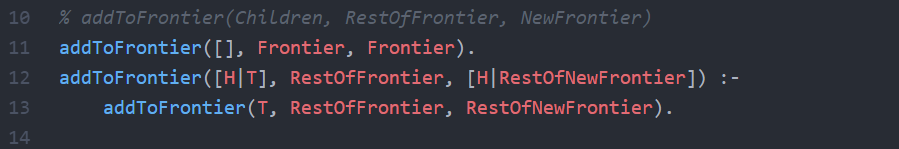
# Frontier Search



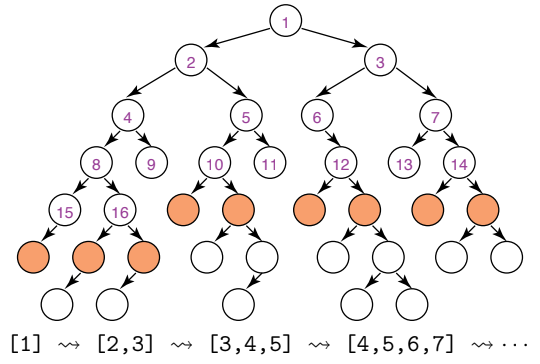


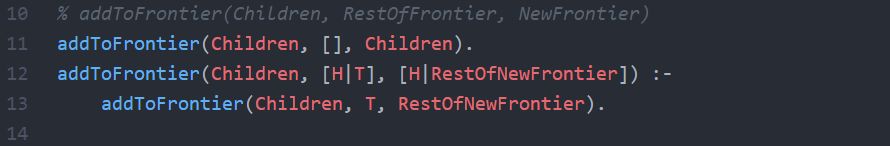
### Depth-First: Stack (LIFO)





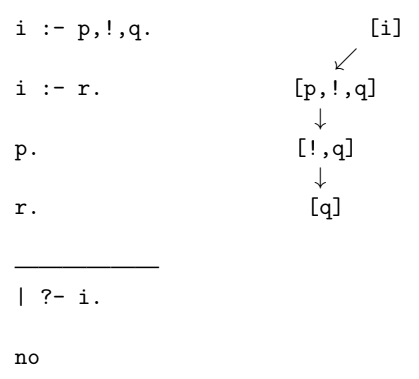
### Breadth-First: Queue (FIFO)



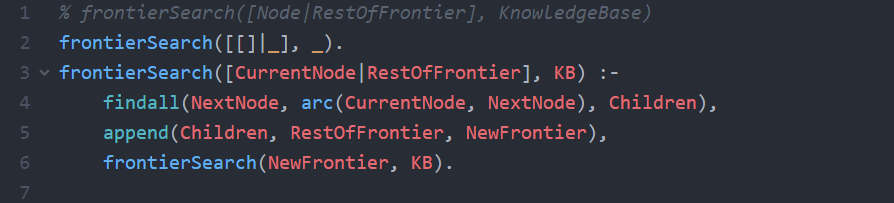


# Cut

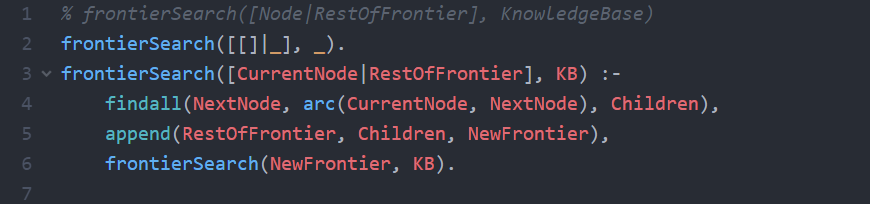
Cut (!) is true, but it destroys backtracking.



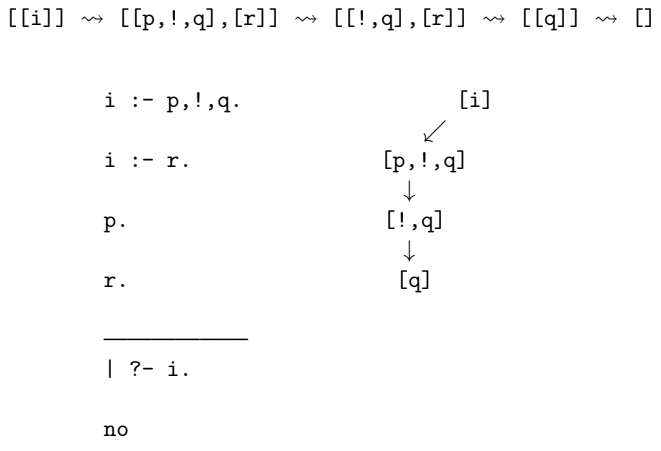
### Depth-First Frontier Search



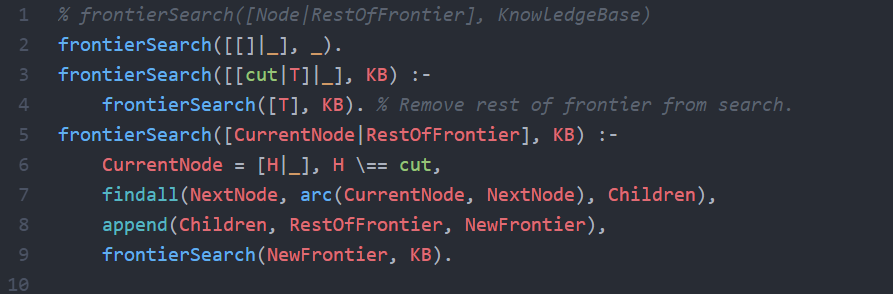
### Breadth-First Frontier Search



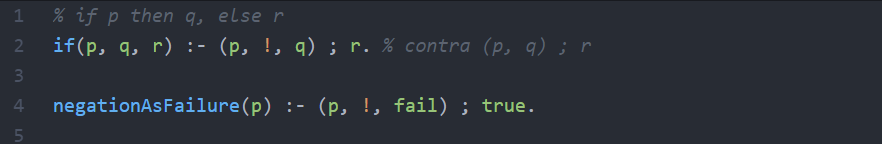
### Tracking the Frontier



### Cut via Frontier Depth-First Search

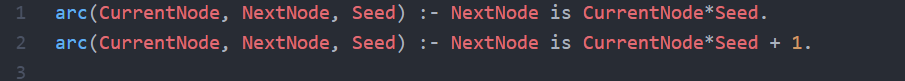


# If-then-else



06 Costs & Heuristics

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



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

Goal nodes are multiples of a positive integer Target.

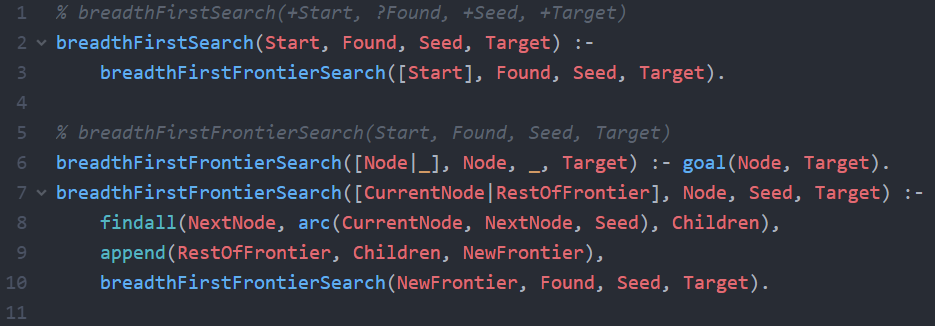


e.g. Target=13 gives goals 13,26,39, …

Modify frontier search to define predicates:



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



For depth-first search:

 BFS

becomes

 DFS

# Refining Frontier Search

For ,

we must ensure that / is no worse than any in :

1. **costs no more** than .
   * Minimum-cost search (Breadth-first if every arc costs 1).
2. is deemed **no further from a goal node** than .
   * Best-first search (Depth-first for heuristic ∝ depth-1).
3. Some mix of 1) and 2).
   * A-star search.

### Arc Costs

|  |  |
| --- | --- |
|  |  |

### Heuristics

Estimate the **minimum cost** of a path from to a goal node.

**Examples**:

FSM accept where , and every arc costs 1.

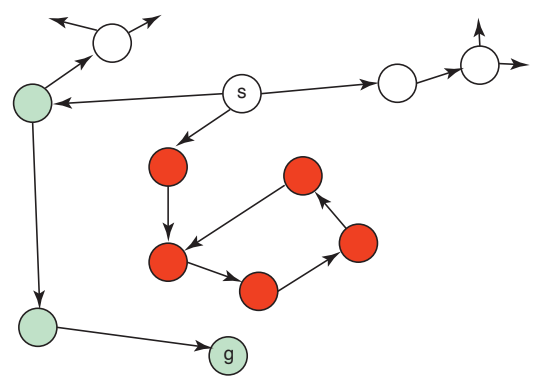
Prolog search where , and every arc costs 1.

, , .

### Best-First Search

Form

such that for every in .



07 A-Star

### Min Cost ≠ Breadth-First

|  |  |
| --- | --- |
|  |  |



for each in ?

*What about the proximity to the goal?*

is the **estimate** of the **minimum cost path** .

# A-Star Search

|  |  |
| --- | --- |
|  | where is explored. |

Ensure where has minimal .

* for every ⇒ **Min-cost**
* for every ⇒ **Best-first** (disregarding the past)

### Admissibility

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

There are three conditions sufficient for admissibility:

1. **Underestimate**:  
   For every solution , .
2. **Termination**:  
   For some , every arc costs .
3. **Finite Branching**:  
    is finite for each node .

Assuming the three conditions above, let be a solution.

**To show**: A\* returns a solution with minimum cost .

Let:

* i.e. the starting frontier.
* be A\*’s next frontier after ( if none).
* be the cost of the head of ( if ).

For every such that , has a **prefix** of .

for some such that the head of is a solution.

Topic 3: Q-Learning & MDP

08 Deterministic Case

is the **goal set**.

|  |  |
| --- | --- |
|  |  |

### Distance dG to Minimise

Refine

*1 if is a goal*

to reward from 1 to 0 (≈ distance from 0 to ∞).

halving the reward as we step back (starting at G).

### Reward rG to Maximise

### Rewards Looking Ahead

*1/0 + Half of the max of the child heuristics*

where:

For ,

where:

For:

### 

A foolproof heuristic for the shortest solution:

where

*What if the arcs have different costs?*

Modify to:

and modify to:

where:

is our **return**.

### Discounted Rewards

Immediate rewards at times give a -discounted value of:

where is the value from time step on

(Backward induction)

which is bound by bounds on

for each implies

since

### 

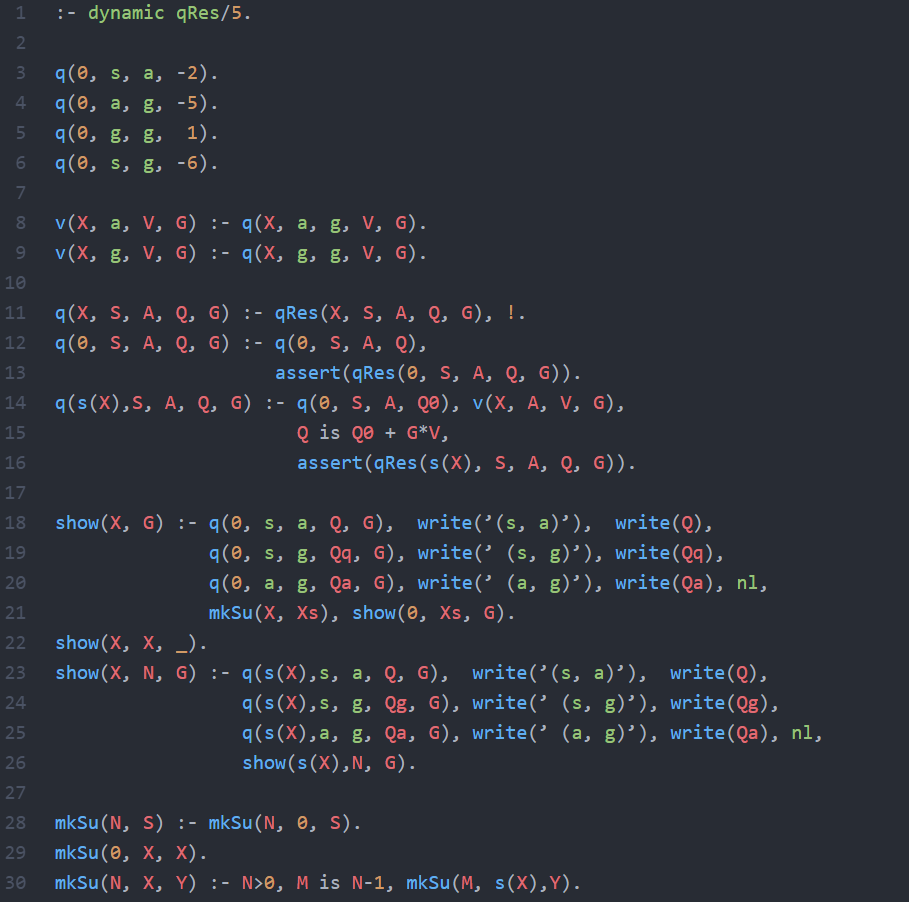
where:

|  |  |
| --- | --- |
|  | Solution not chosen.  Solution? |

|  |  |
| --- | --- |
|  | Costlier solution chosen. |

### Code for 3-Node Graph

This code applies to the first of the previous two diagrams.



### Upping the Reward

Adjust to:

for some reward high enough to **offset costs of reaching a goal**.

For solutions up to arcs with arcs costing :

* states
* is the max arc cost

Let

so for , and

### Recap

From node , find a path to the goal maximising:

with discount on future , contra

Trade min cost guarantee for cost-benefit analysis with chance if survival / doom.

10 Uncertainty

Given a specification of immediate rewards after particular actions, calculate the return of particular actions over time via:

(1)

(2)

|  |  |
| --- | --- |
|  | (1) is (2) with action resulting in deterministically for , with . |

# Markov Decision Process (MDP)

A 5-tuple consisting of:

1. A finite set of **states**
2. A finite set of **actions**
3. A function :   
    = How likely is after doing at .
4. A function : (where is the real numbers.)  
    = Immediate reward at after is done at .
5. A discount factor

Missing: A policy : (what to do at )

### Exercise

Sam is either fit or unfit:

and has to decide whether to exercise or relax:

and are -table entries for row , col :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **exercise** | **fit** | **unfit** | | **fit** | 0.99, 8 |  | | **unfit** | 0.2, 0 |  |   Depend on the resulting state. | |  |  |  | | --- | --- | --- | | **relax** | **fit** | **unfit** | | **fit** | 0.7, 10 |  | | **unfit** | 0, 5 |  |   Immediate rewards do not. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **exercise** | **fit** | **unfit** | | **fit** | 0.99, 8 | 0.01, 8 | | **unfit** | 0.2, 0 | 0.8, 0 | | |  |  |  | | --- | --- | --- | | **relax** | **fit** | **unfit** | | **fit** | 0.7, 10 | 0.3, 10 | | **unfit** | 0, 5 | 1, 5 | |

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

### Policy from an MDP

Given state , pick the action that maximises return:

where:

* is the different outcomes .
* is the immediate reward.
* is the discounted future reward.

For tied to via policy : :

e.g. The greedy -policy above:

for

### Value Iteration

Mutual recursion between and :

Value of an action / state vs. what to do at a state.

Focus on , approached in the limit:

From iterates

In case for some (necessarily unique), the iterates simplify to:

### Deterministic Actions & Absorbing States (Game Over)

Fix an MDP with min reward .

An action is **s-deterministic** if for some .

A state is **absorbing** if for every action , whence

where

A state is a **sink** if it is absorbing and

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

and

Let

so if

### Arcs and Goals as a Deterministic MDP

Given and goal set , let:

where for each :

Satisfy probability constraint via sink state , requiring of every and :

11 Learning

For the prior example:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **exercise** | **fit** | **unfit** | | **fit** | 0.99, 8 | 0.01, 8 | | **unfit** | 0.2, 0 | 0.8, 0 | | |  |  |  | | --- | --- | --- | | **relax** | **fit** | **unfit** | | **fit** | 0.7, 10 | 0.3, 10 | | **unfit** | 0, 5 | 1, 5 | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **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 Differences

A sequence of values averages at time to:

which learning updates to:

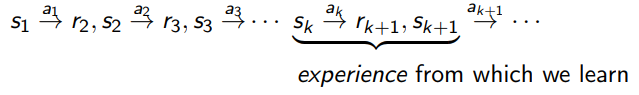
so if ,  
 where is new

where is old, is temporal difference

Temporal difference = new - old.

# Q-Learning

Assume is derived from , observed sequentially:



given:

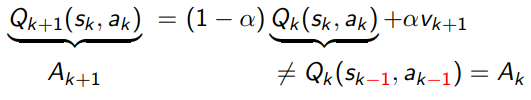
* :

with:

Similar to the previously defined:

for

### Averaging?



For a deterministic MDP: (i.e. )

Let as may look ahead further than for the experience (determined by ).

For , sample at frequency to average as a whole (not just at a particular ), converging to optimal under certain assumptions, including:

and (e.g. )

### MDP, One Experience at a Time

Update : via , for:

or pointwise via experiencefor:

To converge to MDP’s optimal Q-value, visit every state-action pair repeatedly

(for  with different , under , )

End **episode**



at an absorbing state with for every action .

### Exploration-Exploitation Tradeoff

 , are from environment, but ?

from functional policy : [e.g. ]

to : such that

e.g. for actions, having max

**(1)**

**(2)**

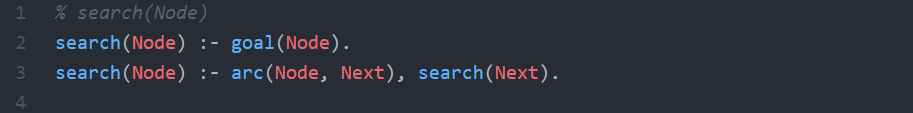
**(1)** says **exploit**: Use what we know.

**(2)** says **explore**: Try something new (for the future).

**SARSA**: Replace arg max by policy in use.

Topic 4: Constraint Satisfaction

12 SAT



More than one Next may satisfy arc(Node, Next) - **Non-determinism**.

Choose Next closest to goal (heuristic: best-first), keeping track of costs (min cost, A\*).

Available choices depend on arc/2 - Actions specified by Turing machine (graph).

Computation eliminates non-determinism (determinisation).

Bound number of calls to arc/2 (iterations of search).

# Feasibility & Non-Determinism: P vs. NP

### Cobham’s Thesis

“A problem is **feasibly solvable** iff some **deterministic Turing machine** (dTm) solves it in **polynomial time**.”

Clearly .

Whether is the most celebrated open mathematical problem in computer science.

* would mean that non-determinism **wrecks** feasibility.
* would mean that non-determinism makes **no difference** to feasibility.

### A Closer Look

Given:

* A set of strings
* A Tm

solves in time if there is a fixed integer such that for every string of size :

iff accepts within steps.

e.g. includes every regular language.

# Boolean Satisfiability (SAT)

**SAT**:

Given a Boolean expression with variables , can we make true by assigning true / false to ?

e.g.

Checking that a particular assignment makes is easy ().

Non-determinism (guessing the assignment) puts SAT in .

But is SAT in ? There are 2n assignments to try.

**Cook-Levin Theorem**:

SAT is in iff .

**CSAT**:

is a conjunction of clauses, where:

* A clause is an OR of literals
* A literal is a variable or negated variable

**k-SAT**:

Every clause has exactly literals.

**3-SAT** is as hard as SAT, 2-SAT is in .

**Horn-SAT**: Every clause has at most one positive literal - linear.

### Prolog & SAT

Prolog KB (definite clauses):

|  |  |
| --- | --- |
|  | [[x1,x2,x4], [x2,x3], x4] |

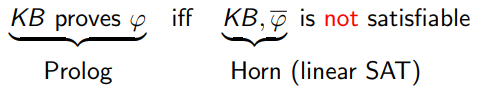
CSAT-input:

|  |  |
| --- | --- |
| . | [[1,-2,-4], [2,3], 4] |

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:



13 Setup

# Constraint Satisfaction Problem [Var, Dom, Con]

* A list of variables
* A list of finite sets of size
* A finite set of **constraints** that may or may not be satisfied by (a node) instantiating with a value in .  
  (Search space size )

e.g. SAT:

, for search space of size 2n.

**Problem**: Satisfy all constraints of , instantiating variables if necessary / convenient.





### Order-Independent Unification (Martelli-Montanari)

**Input**: Set of pairs

**Output**: Substitution unifying pairs in

Simplify non-deterministically until no longer possible

1. (allowing )  
   ⇒ Replace by pairs
2. where or   
   ⇒ Halt with failure
3. ⇒ Delete
4. where is not a var  
   ⇒ Replace by
5. where and occurs elsewhere  
   ⇒ Apply to all other pairs
6. where and   
   ⇒ Halt with failure

**N.B.** Prolog omits the *occurs* check in 5. and 6. for speed-up.

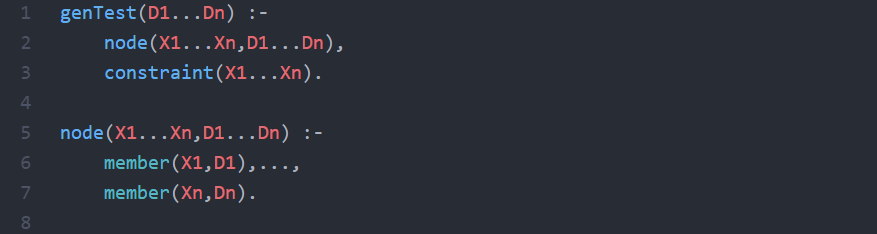
### Instantiate Before Negation (As Failure)

|  |  |
| --- | --- |
|  | Contra: |

### Generate-and-test

**Brute force**:

Instantiate all variables before testing constraints.



For each of the :



(With Di of size si), assume:



can be checked within a polynomial of X1…XN.

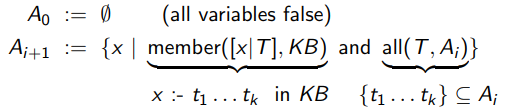
Nodes are g enerated in lexicographic order without regard to constraints.

### Inferring Changes

Horn-SAT by minimal changes to (all variables = 0 / false).

|  |  |  |
| --- | --- | --- |
| **CSAT** | **Definite Clause** | **List Encoding** |
|  | x :- u, z. | [x,u,z] |
|  | false :- u, v. | [false,u,v] |

For each stage , collect the variables set at stage to 1 / true in :



Check:

No minimal set for non-Horn (or xor).

### Instantiate One Variable at a Time

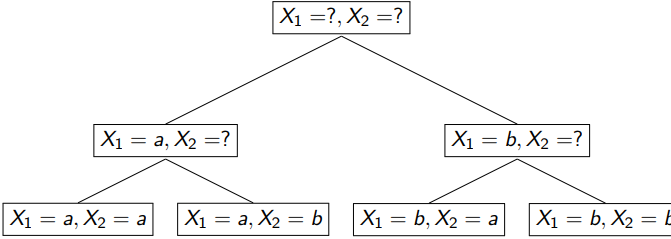
Allow node to map to , raising search space size from

to from adding to .

**Payoff**: Search tree of:

* Depth
* Branching factor
* Start node instantiating no variable
* An arc instantiating least uninstantiated variable

e.g. ,



### Interleave Generation with Testing & Backtracking

Whenever :

1. instantiates one more variable than
2. satisfies every constraint on instantiated variables

Reduce the domains of uninstantiated variables via constraints.

**Constraint graph**:

(e.g. 3-color)

**Arc consistency**:

For and ,

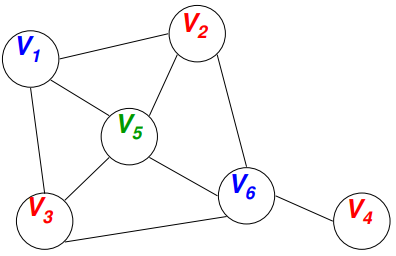
removing from when no such exists.

Optimising the backtracking search:

* **MRV (Minimum Remaining Values)**:  
  Instantiate variable with MRV to minimise branching / cases.
* **LCV (Value that is Least Constraining)**:  
  Assign LCV for greatest chance of success.

14 More Examples

### Canonical Example: Graph Colouring



Consider nodes in a graph.

* Assign values to each of the nodes.
* The values are taken in .
* **Constraints**:  
  If there is an edge between and , then must be different to

### CSP Definition (Again)

:

* , the set of variables:  
  e.g. The values of the nodes in the graph.
* , the set of values that each variable can take:  
  e.g.
* , the set of constraints.  
  Each constraint consists of:  
  **1)** A tuple of variables.  
  **2)** A list of values that the tuple is allowed to take for this problem.  
  e.g.

Constraints are usually **defined implicitly**.

A function is defined to test if a tuple of variables satisfies the constraint.

e.g. for every edge .

### Binary CSP

Variable and are **connected** if they appear in a constraint.

**Neighbours** of are variables that are **connected** to .

The **domain** of , , is the **set of candidate values** for variable .

The constraint graph for binary CSP problem:

* **Nodes** are variables.
* **Links** represent constraints.
* The same as our canonical graph-colouring problem.

### N-Queens

|  |  |
| --- | --- |
| **Variables**:  **Domains**:  **Constraints**:   * (Two queens cannot be in the same row) * (Or same diagonal)   Valid values for are |  |

### Search Space

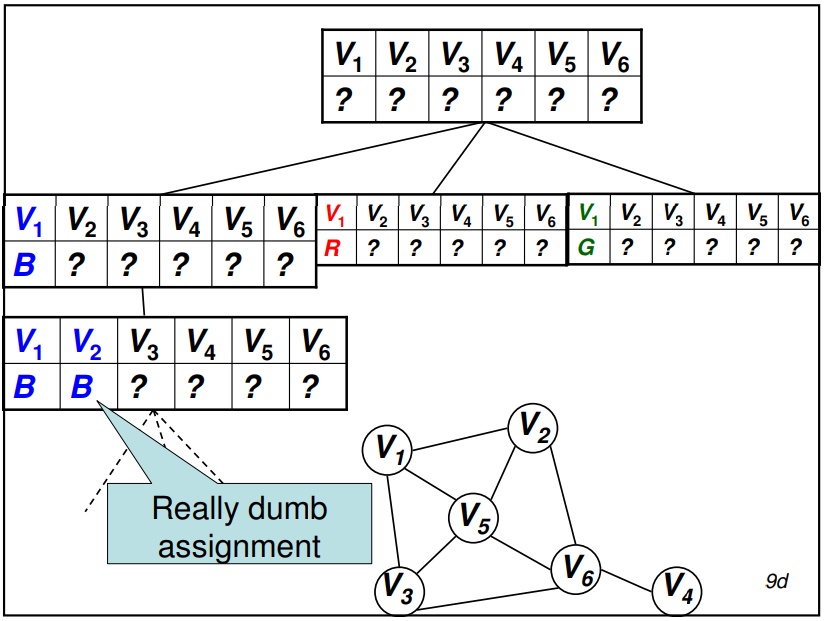
|  |  |
| --- | --- |
|  | **Example state**: |

* **State**:  
  Assignment of variables with unassigned.
* **Successor**:  
  The successor of a state is obtained by assigning a value to variable , keeping the others unchanged.
* **Start State**:
* **Goal State**:  
  All **variables assignment** with **constraints satisfied**.

No concept of **cost on transition**.

We just want to find a solution, without worrying how we get there.

### Depth-First Search



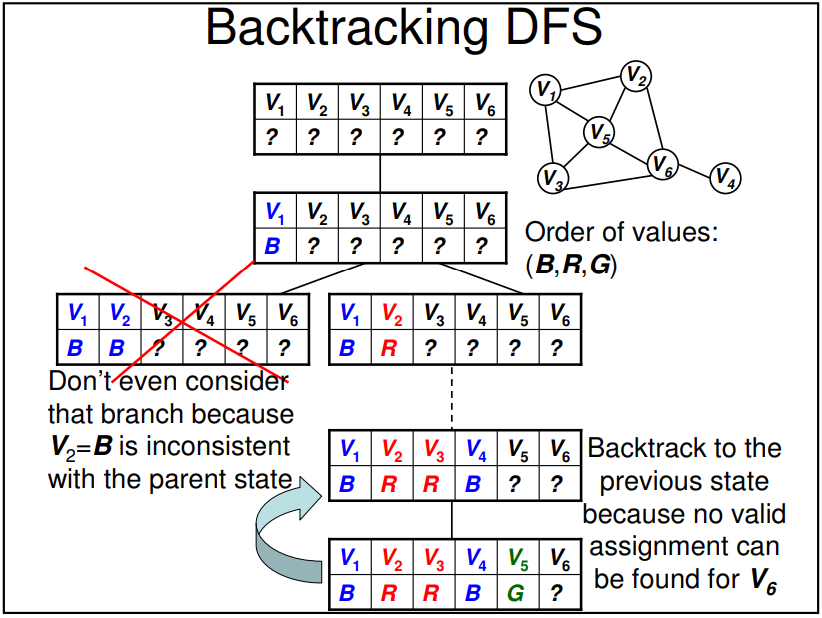
Recursively, for every possible value in :

* Set the next unassigned variable in the successor to that value.
* Evaluate the successor of the current state with this variable assignment.
* Stop as soon as a **solution is found**.

**Improvements**:

* Evaluate only value assignments that do not violate any constraints with the current assignments.
* Don’t search branches that obviously cannot lead to a solution.
* Predict valid assignments ahead.
* Control order of variables and values.

### Backtracking DFS



For every possible value in :

* If assigning to the next unassigned variable does not violate any constraint with the already-assigned variables:
  + Set the variable to .
  + Evaluate the successors of the current state with this variable assignment.
* If no valid assignment is found, backtrack to the previous state.
* Stop as soon as the solution is found.

**Additional computation**:

At each step, we need to evaluate the constraints associated with the current candidate assignment .

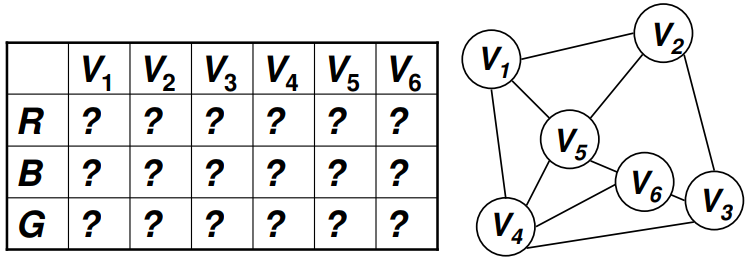
**Uninformed search**: We can improve by **predicting**:

* What is the effect of assigning a variable on all of the other variables?
* What variable should be assigned next, and in which order should values be evaluated?
* When a branch fails, how can we avoid repeating the same mistake?

### Forward Checking

Keep track of **remaining legal values** for unassigned variables.

**Backtrack** when any variable has **no legal values**.



**Warning**: Different example with order .

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

Forward-checking does **not** detect **all** the inconsistencies, but only those that can be detected by looking at the constraints which contain the current variable.

Can we look ahead any further?



### Constraint Propagation

is the variable being assigned at the current level of the search.

Set to a value in .

For every variable connected to :

1. Remove the values in that are **inconsistent** with the assigned variables.
2. Remove every variable **connected** to .
   1. Remove the values in that are **no longer possible candidates**.
   2. Do this again with the variables connected to until **no more values can be discarded**.

Forward-checking as before.

**New**: Constraint propagation.

Topic 5: Logic

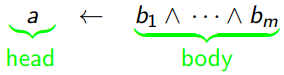
15 Interpretations & Logical Consequence

# Datalog

Datalog is based on the following assumptions:

1. An agent’s knowledge can be usefully described in terms of:
   1. **Individuals**
   2. **Relations** among individuals
2. An agent’s knowledge base consists of:
   1. **Definite** statements
   2. **Positive** statements
3. The environment is **static**.
4. There are only a **finite number of individuals** of interest in the domain.
5. An individual can be named.

### Syntax

* A **variable** starts with an **uppercase** letter.
* A **constant** starts with a **lowercase** letter or is a sequence of digits.
* A **predicate symbol** starts with a lowercase letter.
* A **term** is either a variable or a constant.
* An **atomic symbols** (atom) is of the form or , where:
  + is a predicate symbol
  + are terms
* A **definite clause** is either an atomic symbol (a fact) or is of the form:  
    
  where and are atomic symbols.
* A **query** is of the form .
* A **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 the world.
  + **Constants** denote **individuals**.
  + **Predicate symbols** denote **relations**.

# Formal Semantics - Interpretations

An **interpretation** is a triple , where:

* , the **domain**, is a **non-empty** set.  
  Elements of are **individuals**.
* is a **mapping** that assigns to **each constant** an element of .  
  Constant denotes individuals .
* is a mapping that assigns to each n-ary predicate symbol a relation:  
  A function from into .

### Example Interpretation

**Constants**:

**Predicate Symbol**:

The domain can contain **real objects** (e.g. a person).

cannot necessarily be stored on a computer.

specifies whether the relation denoted by the n-ary predicate symbol is true or false for each -tuple of individuals.

If the predicate symbol has **no arguments**, then is either or .

### Truth in an Interpretation

A constant denotes in the individual .

A **ground atom (variable-free atom)**  is:

* **True in interpretation**  if in interpretation .
* **False** otherwise.

A **ground clause**  is

1. **False in interpretation** if:
   1. is false in and...
   2. Each is true in
2. **True in interpretation**  otherwise.

### Example Truths

In an interpretation given before, which of the following are true?

|  |  |
| --- | --- |
|  |  |

### Models & Logical Consequences

A knowledge base is true in interpretation iff **every clause** in is true in .

A **model** of a set of clauses is an interpretation in which **all the clauses are true**.

If is a set of clauses, and is a conjunction of atoms, then is a **logical consequence** of (written ) if is true in **every model** of .

### 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 - **axiomatising the domain**.
5. Ask questions about the intended interpretation.
6. If , then must be true for the intended interpretation.

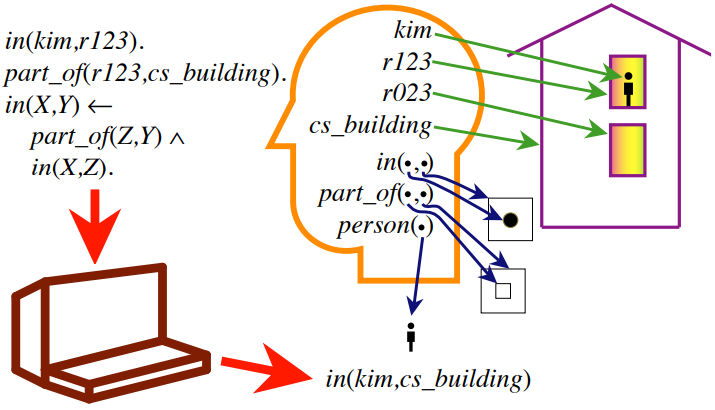
### Computer’s View of Semantics

The computer doesn’t have access to the intended interpretation, all it knows is the **knowledge base** .

The computer cannot determine if a formula is a **logical consequence** of .

If , then must be true for the intended interpretation.

If , then there is a model of in which is false. This could be the intended interpretation.



### Soundness & Completeness

Recall is a logical consequence of (), precisely if is true in **all models** of .

Let proof be a **mechanical procedure** for deriving a formula from a knowledge base , written :

* is **sound** if whenever .
* is **complete** if whenever .

Two extreme examples:

1. for **no** - **Sound**
2. for **all** - **Complete**

### Propositional KBs

Recall:

|  |  |
| --- | --- |
|  |  |

Let

**Theorem**:

1. is **sound** (Proved by induction).
2. is **not complete** (why?).

# Logical Consequences Bottom Up

where is the number of clauses in

|  |  |
| --- | --- |
|  |  |

### Substitutions & Instances

A 0-ary predicate is interpreted by as:

Let be a **set of constants**.

A **-substitution** is a function from a finite set of variables to :

i.e. a set of and distinct variables .

The **application**  of a -substitution to a clause is with each replaced by :

e.g. .

A **-instance** of is for some -substitution .

Given a set of clauses and a -substitution , let:

.

### Bottom-Up with Substitutions

If has constants from some non-empty finite set , let:

e.g. for and ,

### Soundness & Completeness via Herbrand

The **Herbrand interpretation** of a set of clauses with constraints from a non-empty set is the triple , where:

* The domain is the set of constants.
* is the identity function on (each constant in refers to itself).
* For each -ary and -tuple from ,

**Fact**: is a model of , and every clause true in is true in every model of (interpreting constants in ).

**Corollary**: The bottom-up procedure with substitutions is sound and complete (for Datalog).

16 Non-Monotonicity, Defaults & Interference

# Horn Clauses

An **integrity constraint** is a clause of the form:



where:

* Each ai is an atom
* false is a special atom that is false in all interpretations

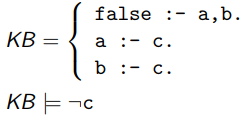
A **Horn clause** is either a:

* **Definite clause** or...
* **Integrity constraint**

### Negations

The **negation** of a formula , written , is a formula that is true in an interpretation iff is false in .

Example:



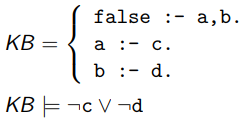
### Disjunctions

Every **set of definite clauses** is **satisfiable**.

This is **not so** with Horn clauses.

The **disjunction** of and is a formula that is true in an interpretation iff **at least one** of or is true in .

Example:



**Horn-SAT** is **feasible**, whereas **3-SAT** is likely **not**.

### Non-Monotonicity

Logical consequence is **monotonic**:

Adding clauses does **not** invalidate a previous conclusion.

implies

Negation-as-failure leads to **non-monotonicity**:

A conclusion **can** be invalidated by adding more clauses.



Sometimes assume that a database of facts is complete.

Any fact **not listed** is **false**.

Example: Assume that a database of video segments is complete.

# Rules

Encode :



to allow for exceptions.

**Default rule ()**: (R. Reiter)

In general:

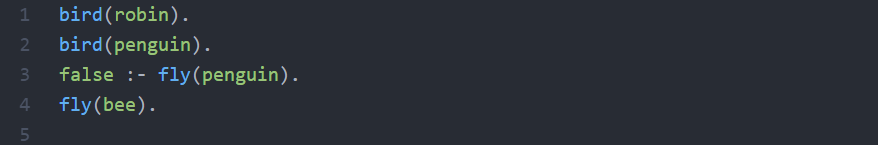
Applied to says:

Conclude if and is -consistent ().

is true in some model of .

### Birds and Bees

Let be:



**Conclude**:

* by default rule ()
* but not

An explanation of using () is:

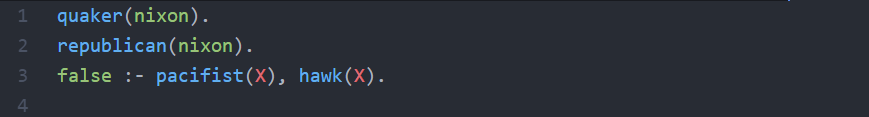
which we can **block** by adding to the rule:



### Non-Determinism

Conflicting defaults:

Let be:



Applying **one default** to Nixon makes the other **inapplicable**.

has two **incompatible extensions**, breaking least fixed point (provability model) for Horn clauses.

### Normal Default Rules

A default rule is **normal** if its justification is its conclusion.

Infer if it is **consistent** and is **provable**.

**Closed-world assumption**:

Any **unprovable** atom is **false**.

**Negation-as-failure**:

is false if attempting to prove fails **infinitely**.



**N.B.** Checking **finite failure** can be as hard as the Halting Problem.

### 3 Modes of Inference

|  |  |  |
| --- | --- | --- |
| **Deduction** | Deduce | Modus ponens ≅ function app . |
| **Abduction** | Explain | Choose input from **assumables**. |
| **Induction** | Generalise / Program | Choose rule / function . |

From as inclusion :

to weighing alternatives via probabilities given :

conditional probability of given

17 Abduction Lab

Maybe later :/

Feel free to send me solutions xo

Topic 6: Bayesian Networks

18 Conditional Probabilities & Independence

From a Constraint Satisfaction Problem to random variables with probabilities constrained by a graph:

1. The **domain** (range) of a variable , written . is the set of **possible values** that can take.
2. A **proposition**  is an equation between a variable and a value , or a Boolean combination of such.

A proposition is **assigned a probability** through:

1. A **notion**  of a possible world satisfying , and...
2. A **measure**  for weighing a set of possible worlds.

### Satisfaction, Measure & Probability

Fix a set of possible worlds that assign a value to each random variable, and interpret a proposition via .

assigns the value

and

or

For finite , a **probability measure** is a function:

such that and for any subset of :

Given , a proposition has probability:

### Tuples, Distributions & The Sum Rule

A **tuple**  of random variables is a random variable with domain:

A **probability distribution** on a random variable is a function:

such that

is often written as , and as .

The **Sum Rule**:

### Conditional Probability

To incorporate a proposition into the background assumptions, we restrict the set of possible worlds to:

and assuming , map a subset to:

for a probability measure on .

The **conditional probability** of given is:

### The Product Rule & Bayes’ Theorem

The **Product Rule**:

for

As conjunction is commutative ():

and so the product rule yields **Bayes’ Theorem**:

The **prior probability** of :

is updated by to the **posterior probability** given :

### Why is Bayes’ Theorem Interesting?

Form a hypothesis given evidence with via Bayes’:

We often have **causal knowledge**:

but want to do some **evidential reasoning**: ()

### Tuples & The Chain Rule

Recall that a tuple of random variables is a random variable.

Let us write:

and apply the product rule repeatedly for:

(Chain rule)

with as the empty tuple and .

### Simplifying the Chain Rule via Conditional Independence

Choose a sub-tuple of such that:

is **independent** of given , written :

i.e. ,

Knowing ’s value says nothing about ’s value, given ’s value.

Note:

# Belief Networks

Totally order the variables of interest:

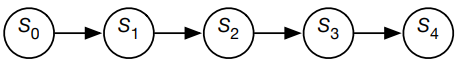
and for each from to , choose from such that:

A **belief network** consists of:

1. A **directed acyclic graph** with nodes = random variables, and an arc from the parents of each node into that node.
2. A **domain** for each random variable.
3. **Conditional probability tables** for each variable given its parents respecting ().

### Markov Chains

A **Markov chain** is a special sort of belief network:



What probabilities need to be specified?

* specifies **initial conditions**.
* specifies the **dynamics**.

What independent assumptions are made?

represents the **state** at time , capturing **everything about the past** () that can affect the future ().

The future is **independent** of the past, given the present.

### Two Elaborations

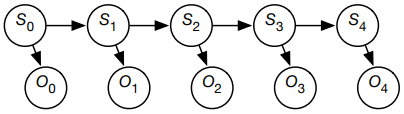
In a **stationary Markov chain**:

and

so it is simple enough to specify and :

* A **simple model** that is **easy to specify**.
* The hidden network can be **extended indefinitely**.

A **Hidden Markov Model (HMM)** is a belief network of the form:



where:

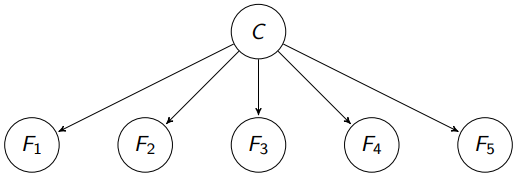
* specifies **initial conditions**.
* specifies the **dynamics**.
* specifies the **sensor model**.

### Naive Bayes Classifier

**Problem**:

Classify on the basis of features :

Assume s are **independent** of each other given :



Assume the values of features are **predictable** given a class.

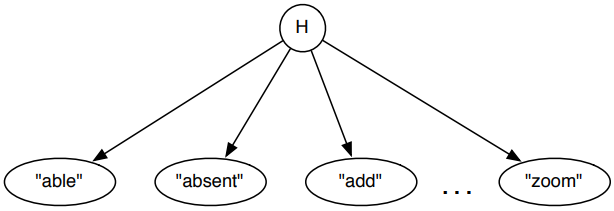
Requires and for each .

### Learning Probabilities

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **F1** | **F2** | **F3** | **F4** | **C** | **Count** |
| ... | ... | ... | ... | ... | ... |
| t | f | t | t | 1 | 40 |
| t | f | t | t | 2 | 10 |
| t | f | t | t | 3 | 50 |
| ... | ... | ... | ... | ... | ... |

With pseudo-counts (Cromwell’s rule).

### Help System



The domain of is the set of all help pages.

The **observations** are 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:

1. What are the relevant **variables**?
   * What will you observe?
   * What would you like to find out (query)?
   * What other features make the model simpler?
2. What **values** should the variables take?
3. What is the **relationship** between them?
   * Express this in terms of a **directed graph**, representing how each variable is generated from its predecessors .
   * The parents of are variables on which directly depends.  
      is dependent of its non-descendants given its parents.
4. How does the value of each variable **depend on its parents**?
   * Expressed in terms of the conditional probabilities.

### Example: Fire Alarm Belief Network

Variables:

1. **Fire**: There is a fire in the building.
2. **Tampering**: Someone has been tampering with the alarm.
3. **Smoke**: What appears to be smoke coming from an upstairs window.
4. **Alarm**: The fire alarm going off.
5. **Leaving**: People are leaving the building en-masse.
6. **Report**: A colleague says that people are leaving the building en-masse  
    (A noisy sensor for leaving).

### Head-to-Tail: Chain

|  |  |
| --- | --- |
|  | and are **dependent**.  and are **independent given** .  Intuitively, the only way that affects is by affecting .  *(cond. prob)*  *(net)*  *(product)*  *(for ㅛ)* |

### Tail-to-Tail: Common Ancestors

|  |  |
| --- | --- |
|  | and are **dependent**.  and are **independent given** .  Intuitively, can explain and :  Learning one can affect the other by **changing your belief** in fire.  *(net)*  *(for ㅛ)* |

### Head-to-Head: Common Descendants

|  |  |
| --- | --- |
|  | and are **independent**.  and are **dependent given** .  Intuitively, can explain away .  for and : |

### 

*(Bayes’ rule)*

*(Sum)*

*(Product)*

*(Net)*

*(Sum)*

*(Net)*

### 

*(Net)*

*(Bayes’)*

*(Sum)*

*(Product)*

*(Net)*

# Conditional Independence via D-Separation

Given disjoint sets of nodes (variables),

when are the variables of **independent** of those in given ?

When is d-separated from by

i.e. All paths from to are -blocked.

A path from to is-blocked if it has a node such that:

1. is in , and the arrows on the path meet head-to-tail or tail-to-tail at or…
2. Neither nor any of its descendants are in , and the arrows on the path meet head-to-head at .

**Fact**:  
If is d-separated from by , then the variables in are independent of those in given (for all network probabilities).

We can make the net **undirected** with disconnected = d-separate.

### Understanding Conditional Independence

From non-implications:

(head-to-head)

(head-to-head, tail-to-tail)

to

**Graphoid axioms**:

as “ intercepts all paths from to ”.

1. implies
2. implies
3. implies
4. and implies
5. and implies