Constraint Satisfaction Problem

Constraint satisfaction problems (CSPs)

- Standard search problem:
  state is a "black box"—any old data structure
     that supports goal test, eval, successor

- CSP:
  state is defined by variables X_i with values from domain D_i

  goal test is a set of constraints specifying
     allowable combinations of values for subsets of variables

Simple example of a formal representation language

Allows useful general-purpose algorithms with more power than standard search algorithms

Example: Map-Coloring
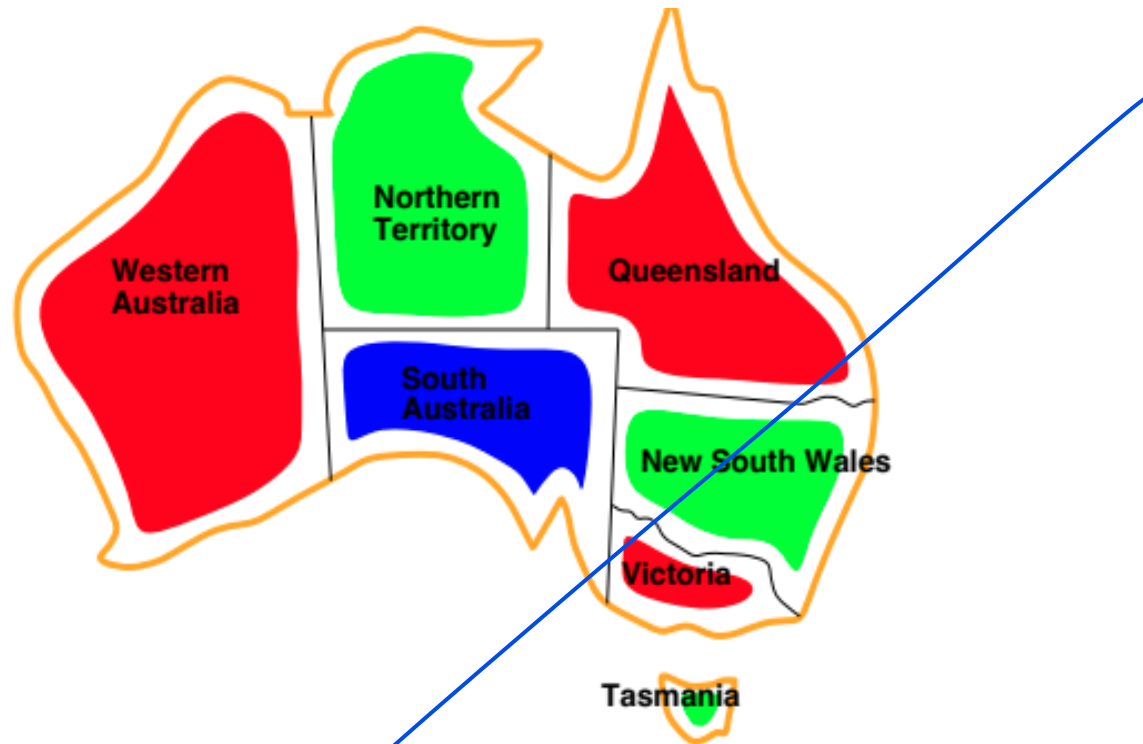

Variables WA, NT, Q, NSW, V, SA, T
Domains $D_i = \{red, green, blue\}$
Constraints: adjacent regions must have different colors
  e.g., $WA \neq NT$ (if the language allows this), or
  $(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$
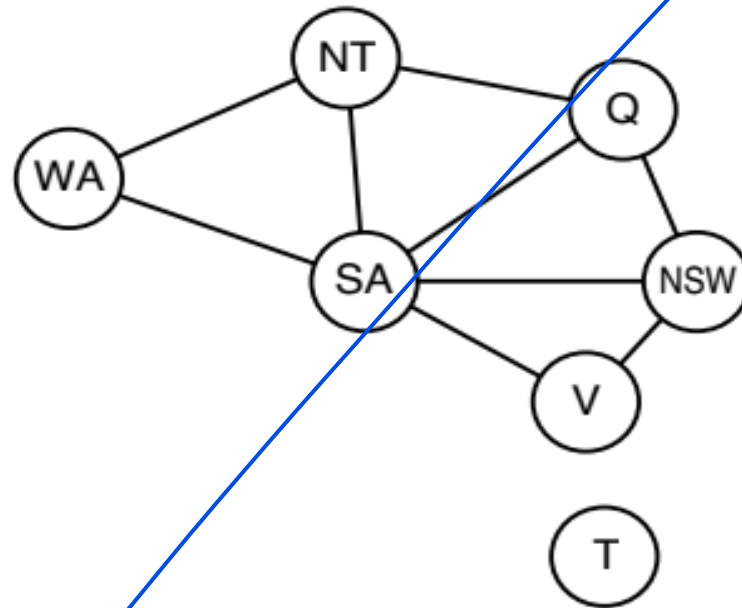
Example: Map-Coloring

Solutions are assignments satisfying all constraints, e.g.,
$$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$$

Constraint graph

Binary CSP: each constraint relates at most two variables

Constraint graph: nodes are variables, arcs show constraints

General-purpose CSP algorithms use the graph structure
to speed up search. E.g., Tasmania is an independent subproblem!

Varieties of CSPs

VARIABLE TYPES

Discrete variables

    finite domains; size $d \Rightarrow O(d^n)$ complete assignments

        $\diamond$ e.g., Boolean CSPs, incl. Boolean satisfiability (NP-complete)

    infinite domains (integers, strings, etc.)

        $\diamond$ e.g., job scheduling, variables are start/end days for each job

        $\diamond$ need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$

        $\diamond$ linear constraints solvable, nonlinear undecidable

Continuous variables

        $\diamond$ e.g., start/end times for Hubble Telescope observations

        $\diamond$ linear constraints solvable in poly time by LP methods

Varieties of constraints
CONSTRAINT TYPE

Unary constraints involve a single variable,
e.g., $SA \neq green$

Binary constraints involve pairs of variables,
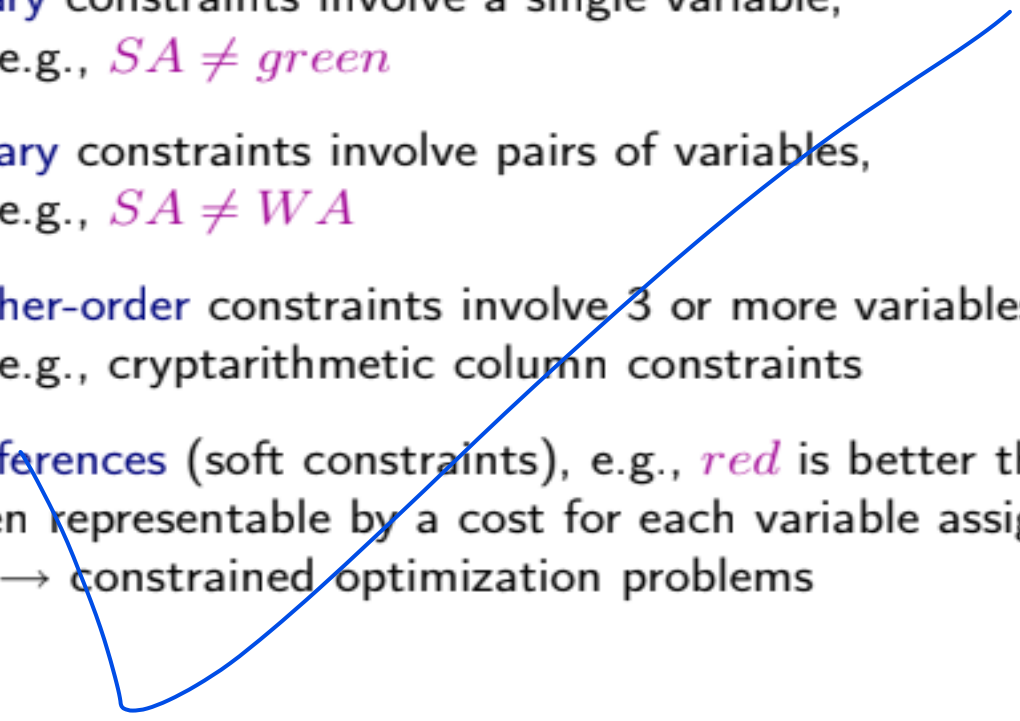e.g., $SA \neq WA$

Higher-order constraints involve 3 or more variables,
e.g., cryptarithmetic column constraints

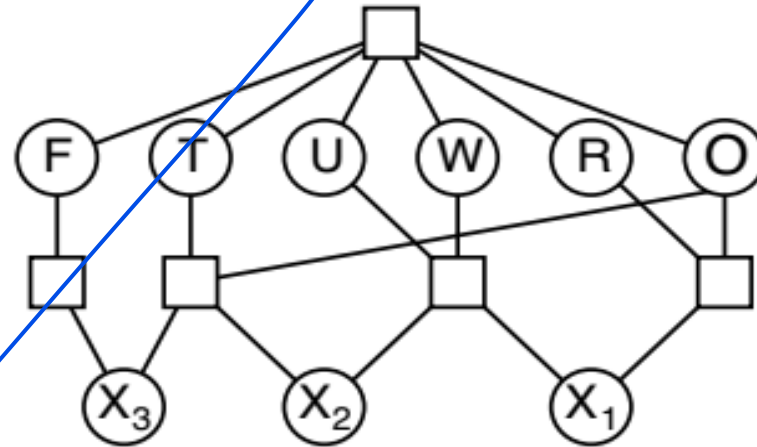Preferences (soft constraints), e.g., red is better than $green$
often representable by a cost for each variable assignment
$\rightarrow$ constrained optimization problems

Example: Crypt arithmetic

```
   T W O
 + T W O
 -------
 F O U R
```

Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Constraints
  $\textit{alldiff}(F, T, U, W, R, O)$
  $O + O = R + 10 \cdot X_1$, etc.

Real-world CSPs

- Assignment problems

  e.g., who teaches what class
- Timetabling problems

  e.g., which class is offered when and where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Floorplanning

Notice that many real-world problems involve real-valued variables

Standard search formulation (incremental)

Let's start with the straightforward, dumb approach, then fix it

States are defined by the values assigned so far

◇ Initial state: the empty assignment, { }

◇ Successor function: assign a value to an unassigned variable
that does not conflict with current assignment.
⇒   fail if no legal assignments (not fixable!)

◇ Goal test: the current assignment is complete

1) This is the same for all CSPs! 😈
2) Every solution appears at depth n with n variables
⇒   use depth-first search
3) Path is irrelevant, so can also use complete-state formulation
4) $b = (n - \ell)d$ at depth ℓ, hence $n!d^n$ leaves!!!! 😧

Backtracking search

Variable assignments are commutative, i.e.,

$$[WA = red \text{ then } NT = green] \text{ same as } [NT = green \text{ then } WA = red]$$

Only need to consider assignments to a single variable at each node

$$\Rightarrow \quad b = d \text{ and there are } d^n \text{ leaves}$$

Depth-first search for CSPs with single-variable assignments
is called backtracking search

Backtracking search is the basic uninformed algorithm for CSPs

Can solve n-queens for $n \approx 25$
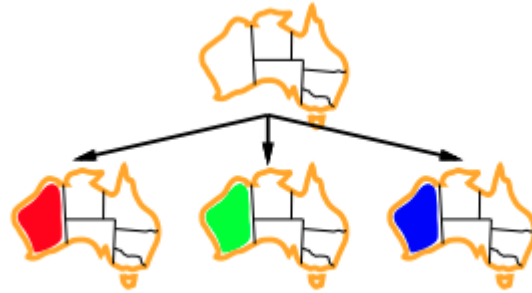
Backtracking search

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```
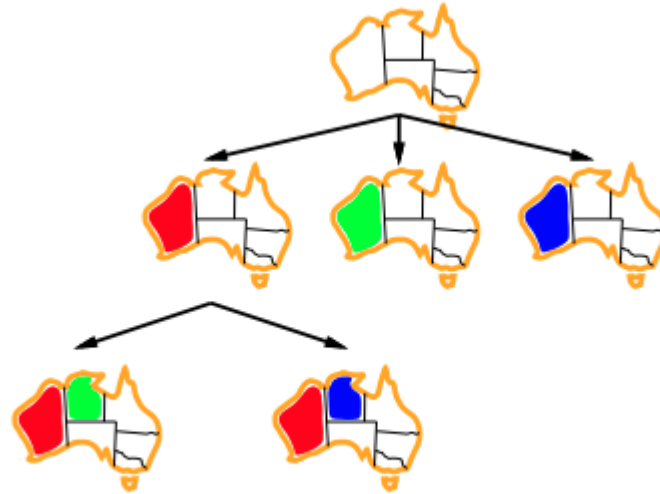
Backtracking example

Backtracking example

Backtracking example

Backtracking example
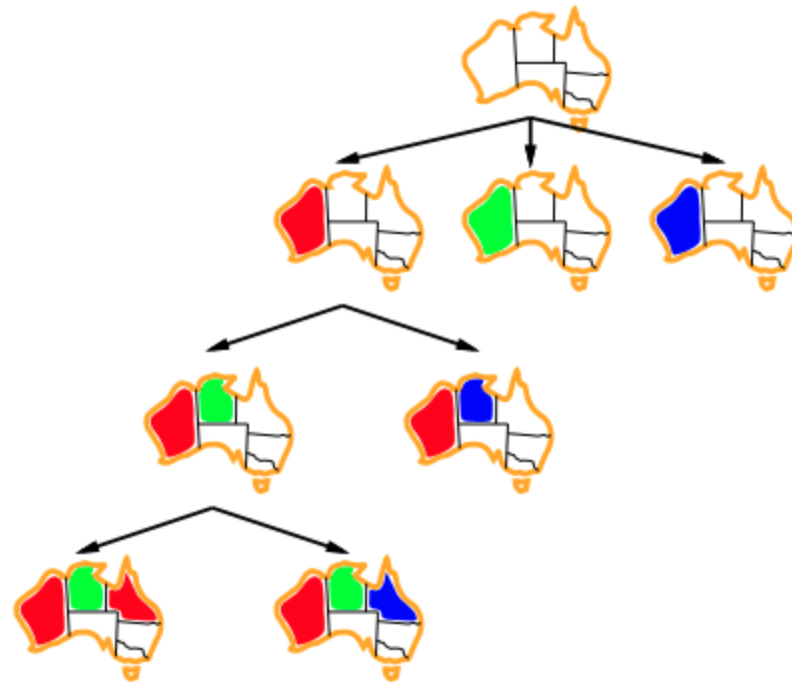
Improving backtracking efficiency

General-purpose methods can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. Can we take advantage of problem structure?

Minimum remaining values

Minimum remaining values (MRV):
   choose the variable with the fewest legal values

Degree heuristic

Tie-breaker among MRV variables

Degree heuristic:
   choose the variable with the most constraints on remaining variables
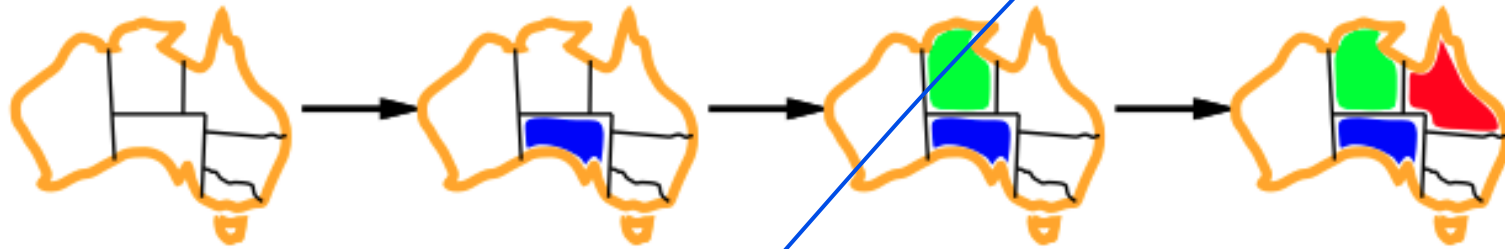
Least constraining value

Given a variable, choose the least constraining value:
  the one that rules out the fewest values in the remaining variables

Allows 1 value for SA

Allows 0 values for SA
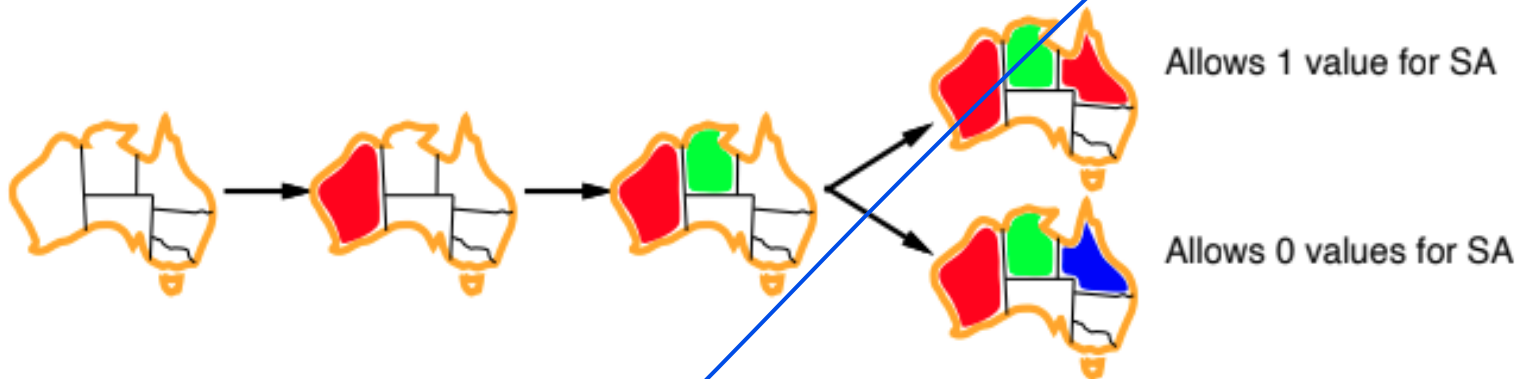
Combining these heuristics makes 1000 queens feasible

Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

WA	NT	Q	NSW	V	SA	T

Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

	WA	NT	Q	NSW	V	SA	T
	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
	🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦

Forward checking

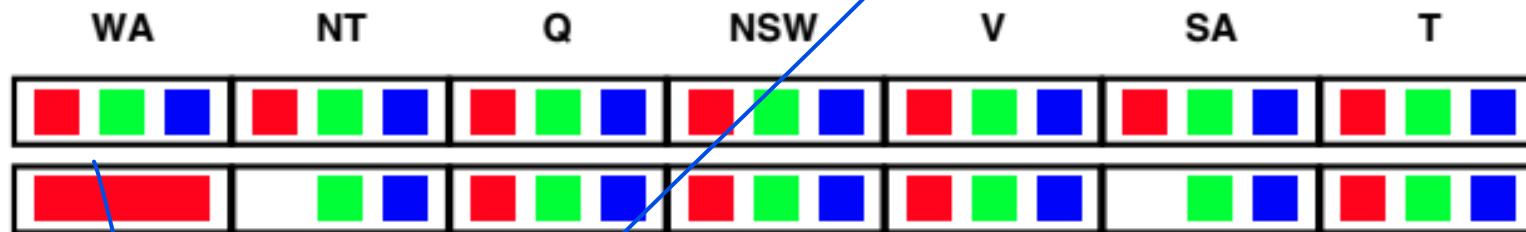Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values

WA	NT	Q	NSW	V	SA	T
red green blue	red green blue	red green blue	red green blue	red green blue	red green blue	red green blue
red	green blue	red green blue	red green blue	red green blue	green blue	red green blue
red	blue	green	red	red green blue	blue	red green blue
red	blue	green	red	blue		red green blue
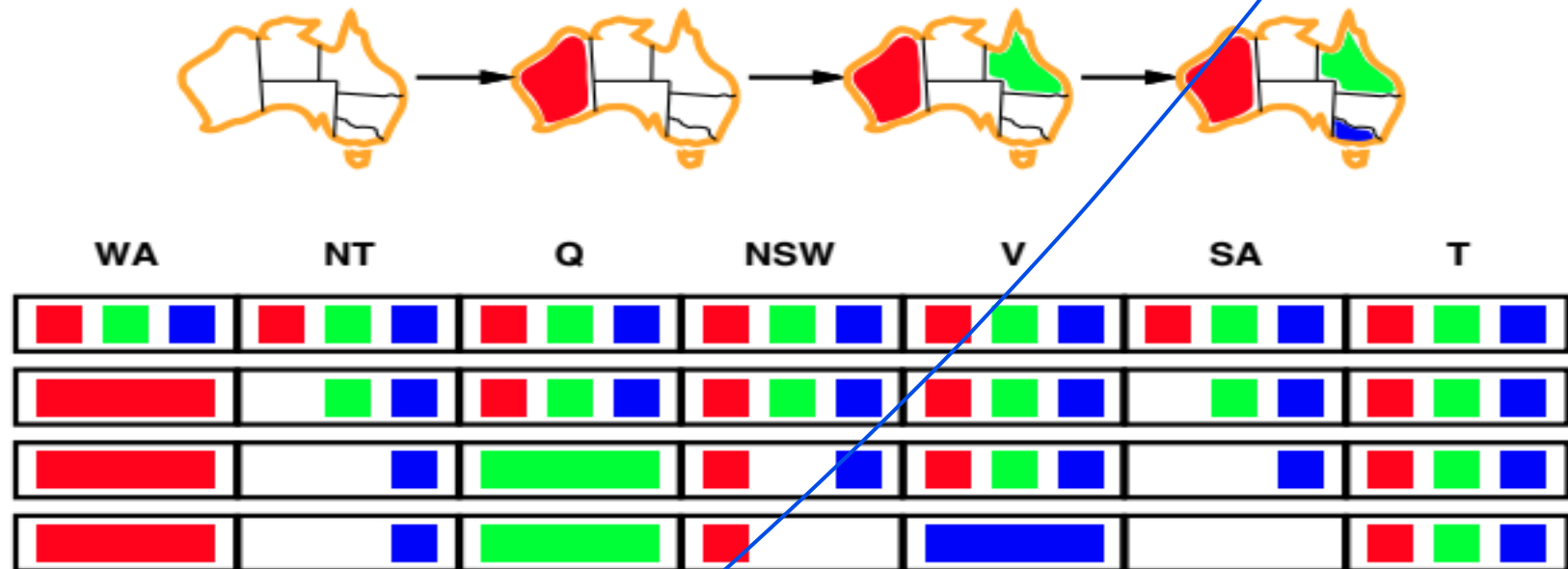
Constraint propagation

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

WA	NT	Q	NSW	V	SA	T
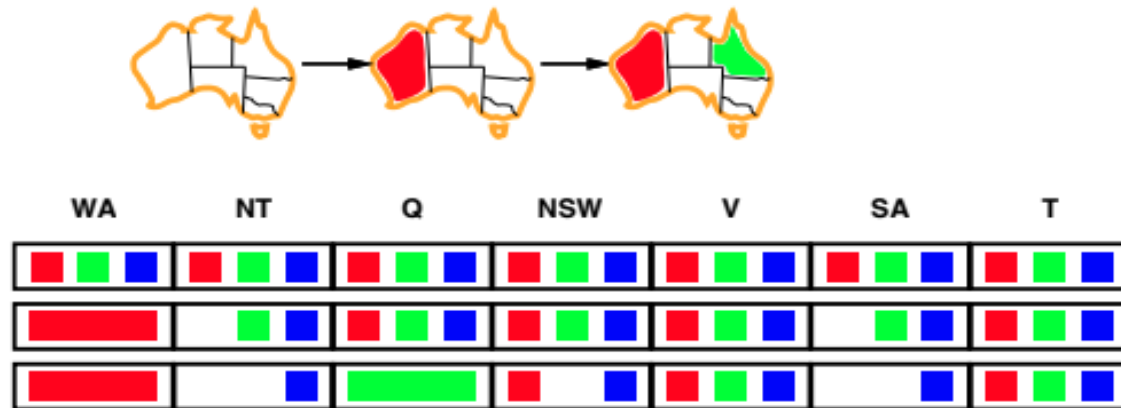
NT and SA cannot both be blue!

Constraint propagation repeatedly enforces constraints locally

Arc consistency

Simplest form of propagation makes each arc **consistent**

$X \rightarrow Y$ is consistent iff
for **every** value x of X there is **some** allowed y

Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y
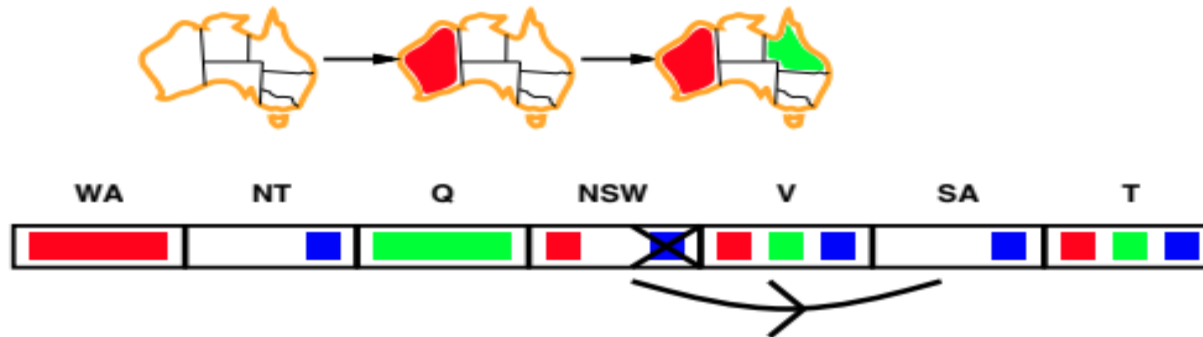
Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff
for **every** value x of X there is **some** allowed y

If X loses a value, neighbors of X need to be rechecked

Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff
    for **every** value x of X there is **some** allowed y

If X loses a value, neighbors of X need to be rechecked

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment

Arc consistency algorithm

function AC-3(*csp*) returns the CSP, possibly with reduced domains
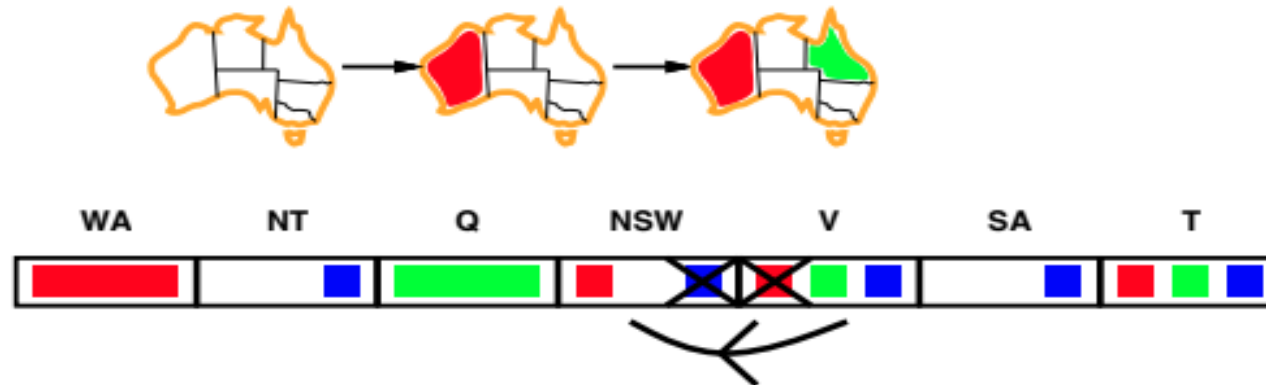   inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
    local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

    while *queue* is not empty do
        $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
        if REMOVE-INCONSISTENT-VALUES(X_i, X_j) then
            for each X_k in NEIGHBORS[X_i] do
                add (X_k, X_i) to *queue*

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) returns true iff succeeds
    removed ← *false*
    for each x in DOMAIN[X_i] do
        if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraint $X_i \leftrightarrow X_j$
            then delete x from DOMAIN[X_i]; *removed* ← *true*
    return *removed*

$O(n^2 d^3)$, can be reduced to $O(n^2 d^2)$ (but detecting **all** is NP-hard)

Problem structure
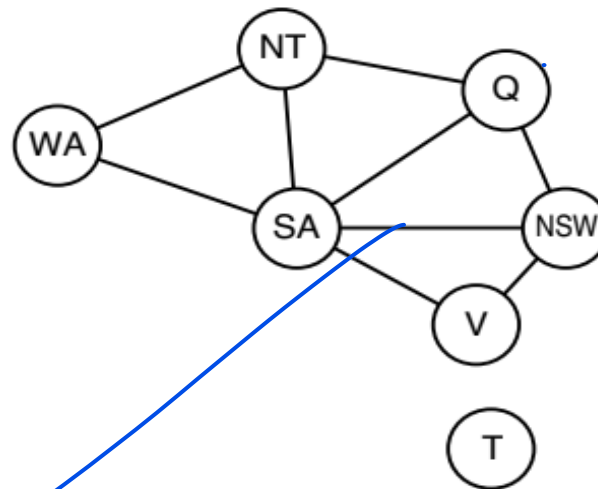
Tasmania and mainland are independent subproblems

Identifiable as connected components of constraint graph

Problem structure contd.

Suppose each subproblem has c variables out of n total

Worst-case solution cost is $n/c \cdot d^c$, **linear** in n

E.g., $n = 80$, $d = 2$, $c = 20$

    2^{80} = 4 billion years at 10 million nodes/sec

    $4 \cdot 2^{20}$ = 0.4 seconds at 10 million nodes/sec

Tree-structured CSPs
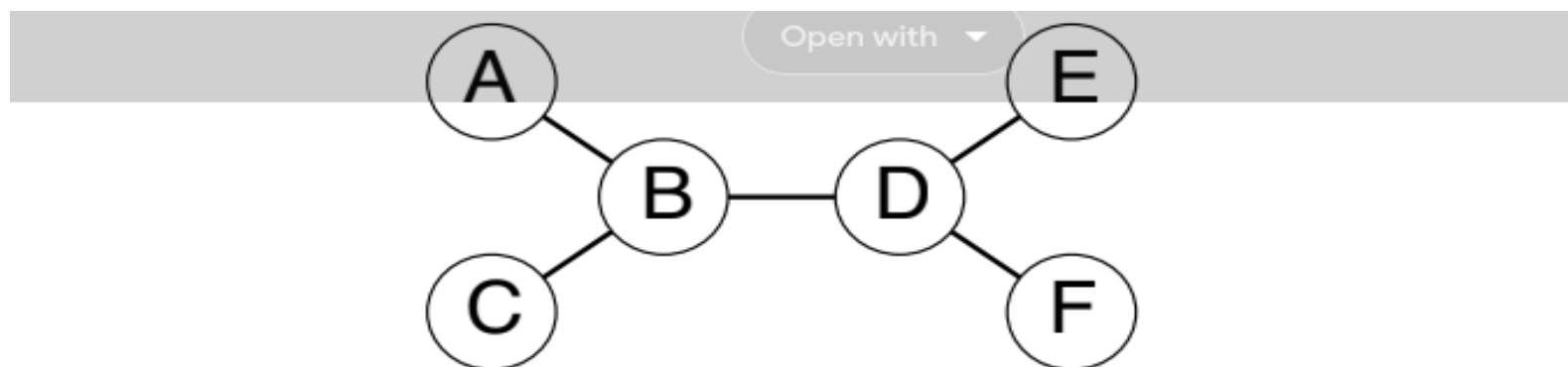
Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\,d^2)$ time

Compare to general CSPs, where worst-case time is $O(d^n)$

This property also applies to logical and probabilistic reasoning:
an important example of the relation between syntactic restrictions
and the complexity of reasoning.

Algorithm for tree-structured CSPs

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering
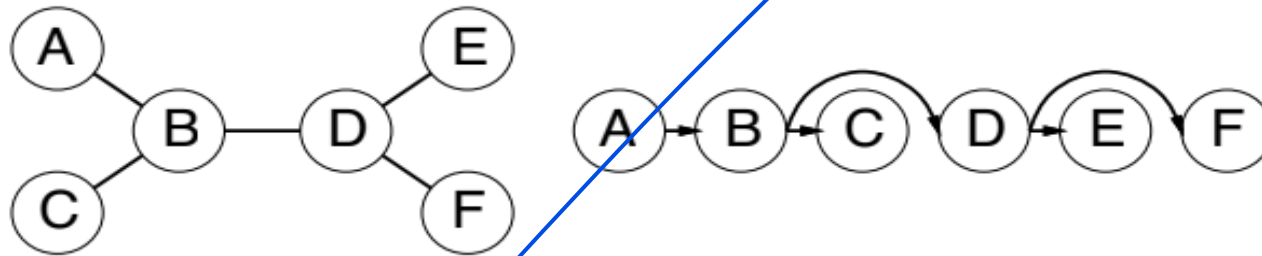
2. For j from n down to 2, apply REMOVEINCONSISTENT($Parent(X_j), X_j$)

3. For j from 1 to n, assign X_j consistently with $Parent(X_j)$

Nearly tree-structured CSPs

Conditioning: instantiate a variable, prune its neighbors' domains
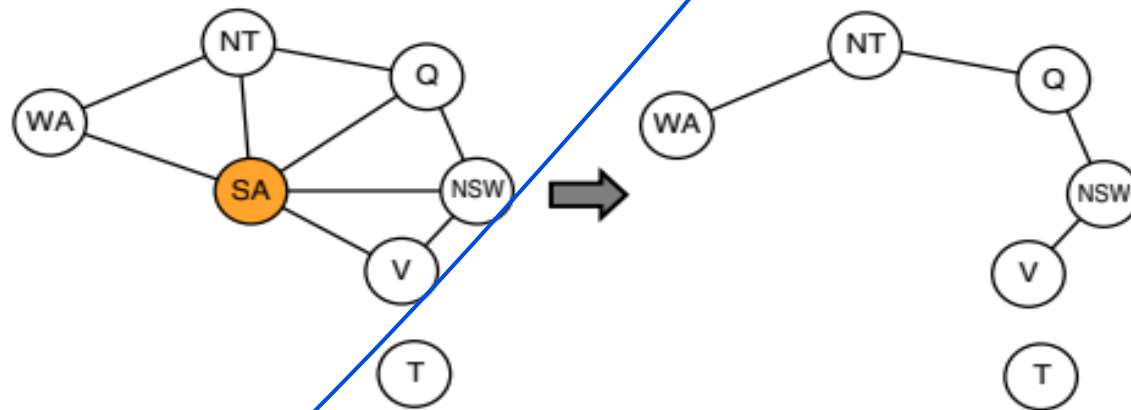
Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

Cutset size c $\Rightarrow$ runtime $O(d^c \cdot (n - c)d^2)$, very fast for small c

Iterative algorithms for CSPs

- Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned
- To apply to CSPs:

    allow states with unsatisfied constraints

    operators reassign variable values
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic:

    choose value that violates the fewest constraints

    i.e., hillclimb with h(n) = total number of violated constraints

Example: 4-Queens

States: 4 queens in 4 columns ($4^4 = 256$ states)

Operators: move queen in column

Goal test: no attacks

Evaluation: $h(n) =$ number of attacks

h = 5   h = 2   h = 0
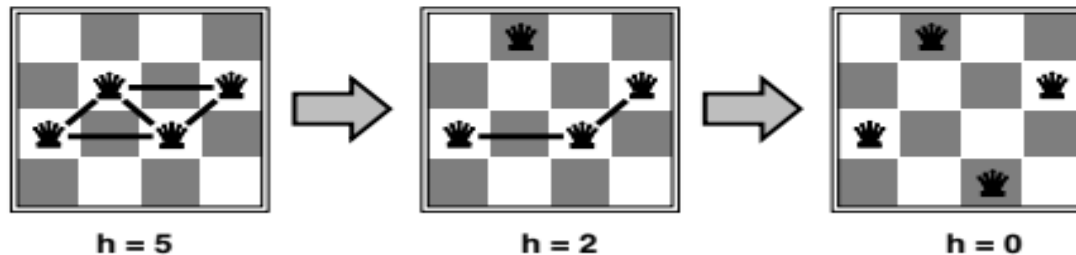
Performance of min-conflicts

Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., $n = 10,000,000$)

The same appears to be true for any randomly-generated CSP **except** in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$
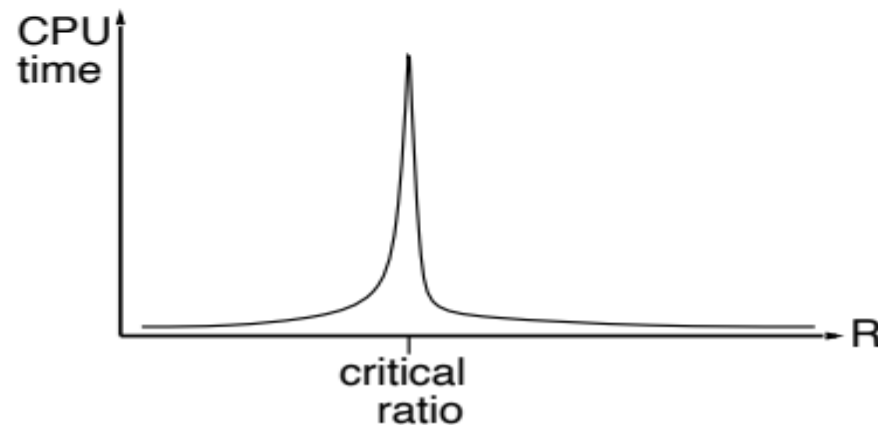
Summary

CSPs are a special kind of problem:
   states defined by values of a fixed set of variables
   goal test defined by constraints on variable values

Backtracking = depth-first search with one variable assigned per node

Variable ordering and value selection heuristics help significantly

Forward checking prevents assignments that guarantee later failure

Constraint propagation (e.g., arc consistency) does additional work
to constrain values and detect inconsistencies

The CSP representation allows analysis of problem structure

Tree-structured CSPs can be solved in linear time

Iterative min-conflicts is usually effective in practice

Natural Language Processing

Introduction

- Language meant for communicating with the world.
- Also, By studying language, we can come to understand more about the world.
- If we can succeed at building computational mode of language, we will have a powerful

  tool for communicating with the world.
- Also, We look at how we can exploit knowledge about the world, in combination with

  linguistic facts, to build computational natural language systems.

➔ Natural Language Processing (NLP) problem can divide into two tasks:

1. Processing written text, using lexical, syntactic and semantic knowledge of the language

   as well as the required real-world information.

2. Processing spoken language, using all the information needed above plus additional knowledge about phonology as well as enough added information to handle the further ambiguities that arise in speech.

Steps in Natural Language Processing
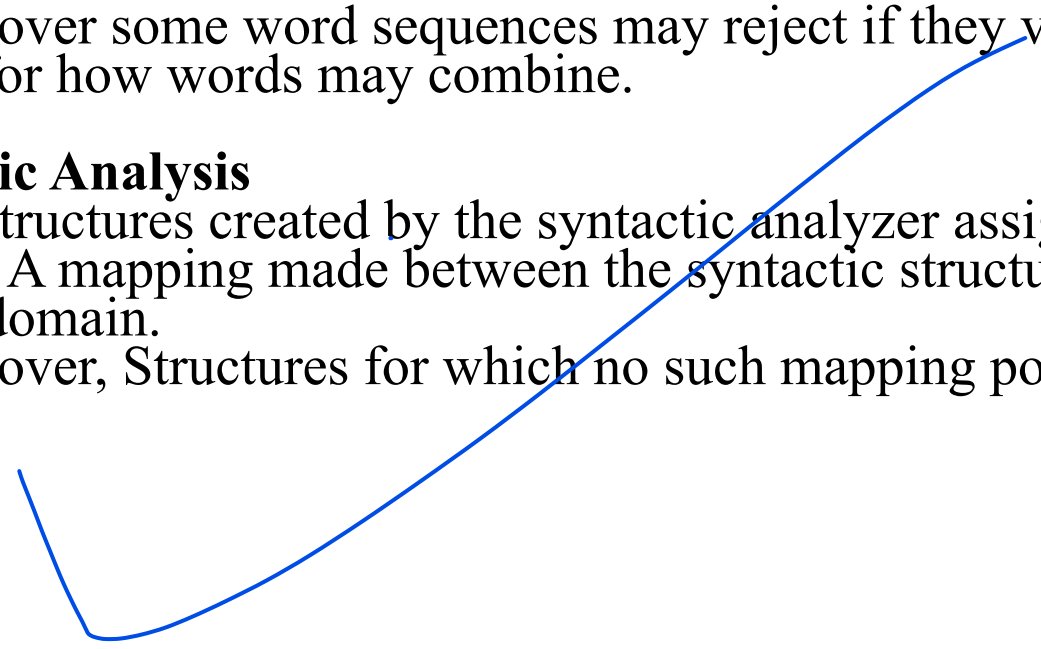
Morphological Analysis
- Individual words analyzed into their components and non-word tokens such as punctuation separated from the words.

Syntactic Analysis
- Linear sequences of words transformed into structures that show how the words relate to each other.
- Moreover some word sequences may reject if they violate the language's rule for how words may combine.

Semantic Analysis
- The structures created by the syntactic analyzer assigned meanings.
- Also, A mapping made between the syntactic structures and objects in the task domain.
- Moreover, Structures for which no such mapping possible may reject.

Steps in Natural Language Processing

Discourse integration
- The meaning of an individual sentence may depend on the sentences that precede it and also, may influence the meanings of the sentences that follow it.

Pragmatic Analysis
- Moreover, The structure representing what said reinterpreted to determine what was actually meant.

Steps in Natural Language Processing

Morphological Analysis

- Suppose we have an English interface to an operating system and the following sentence typed: I want to print Bill's .init file.

➔ The morphological analysis must do the following things:

- Pull apart the word Bill's´ into proper noun Bill´ and the possessive suffix.

- Recognize the sequence ".init " as a file extension that is functioning as an adjective in the sentence.

- This process will usually assign syntactic categories to all the words in the sentence.
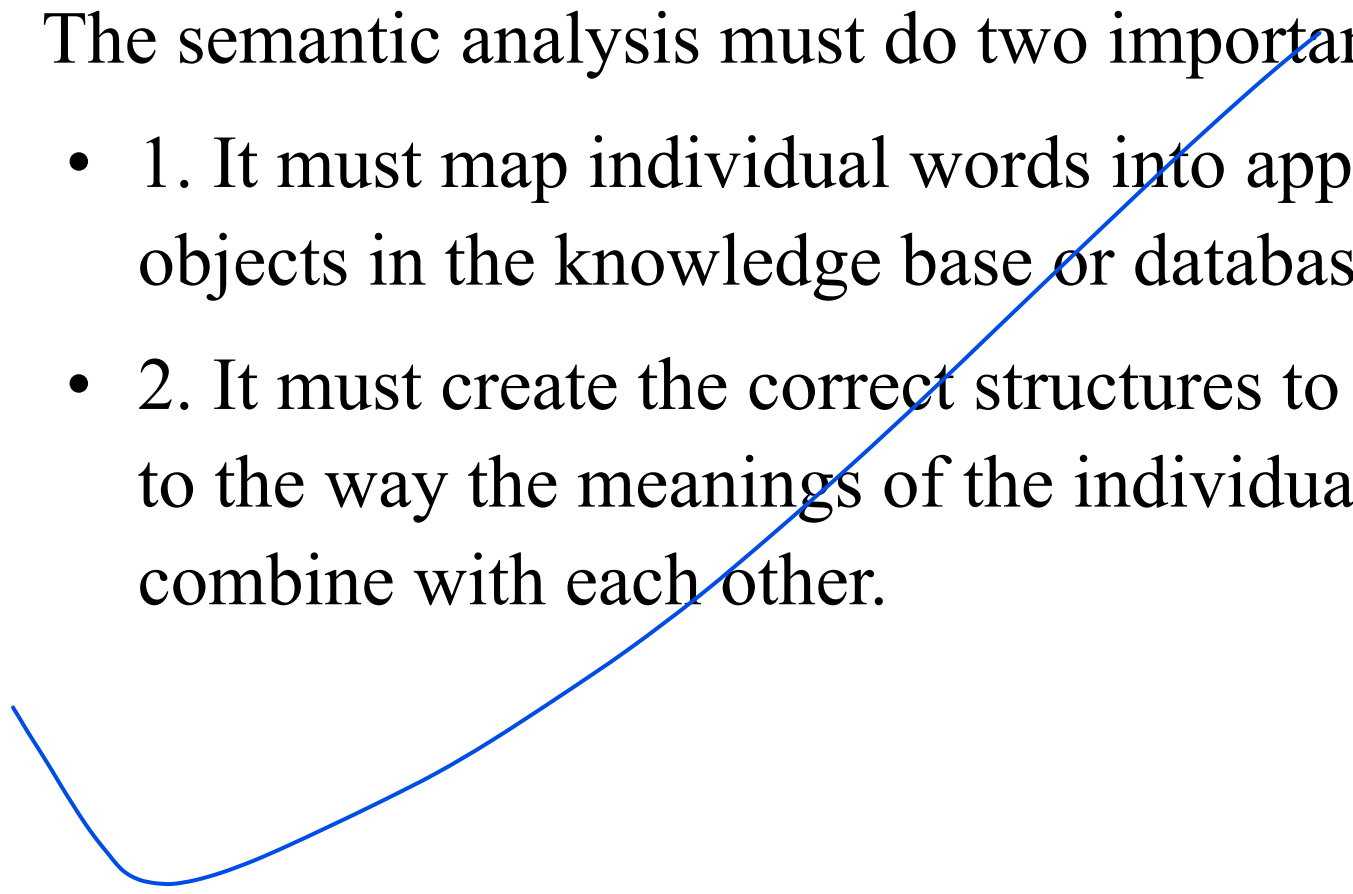
Syntactic Analysis

- A syntactic analysis must exploit the results of the morphological analysis to build a structural description of the sentence.
- The goal of this process, called parsing, is to convert the flat list of words that form the sentence into a structure that defines the units that represented by that flat list.
- The important thing here is that a flat sentence has been converted into a hierarchical structure. And that the structure corresponds to meaning units when a semantic analysis performed.
- Reference markers (set of entities) shown in the parenthesis in the parse tree.
- Each one corresponds to some entity that has mentioned in the sentence.
- These reference markers are useful later since they provide a place in which to accumulate information about the entities as we get it.

Syntactic Analysis

I want to print Bill's .init file.

S
(RM1)

NP   VP

PRO   V   S
        Want (RM3)

I   NP   VP
(RM2)
        PRO   V   NP
                  print (RM4)

        to   ADJS   NP
        (RM2)
                        Bill's   ADJS   N
                        (RM5)   .init   file
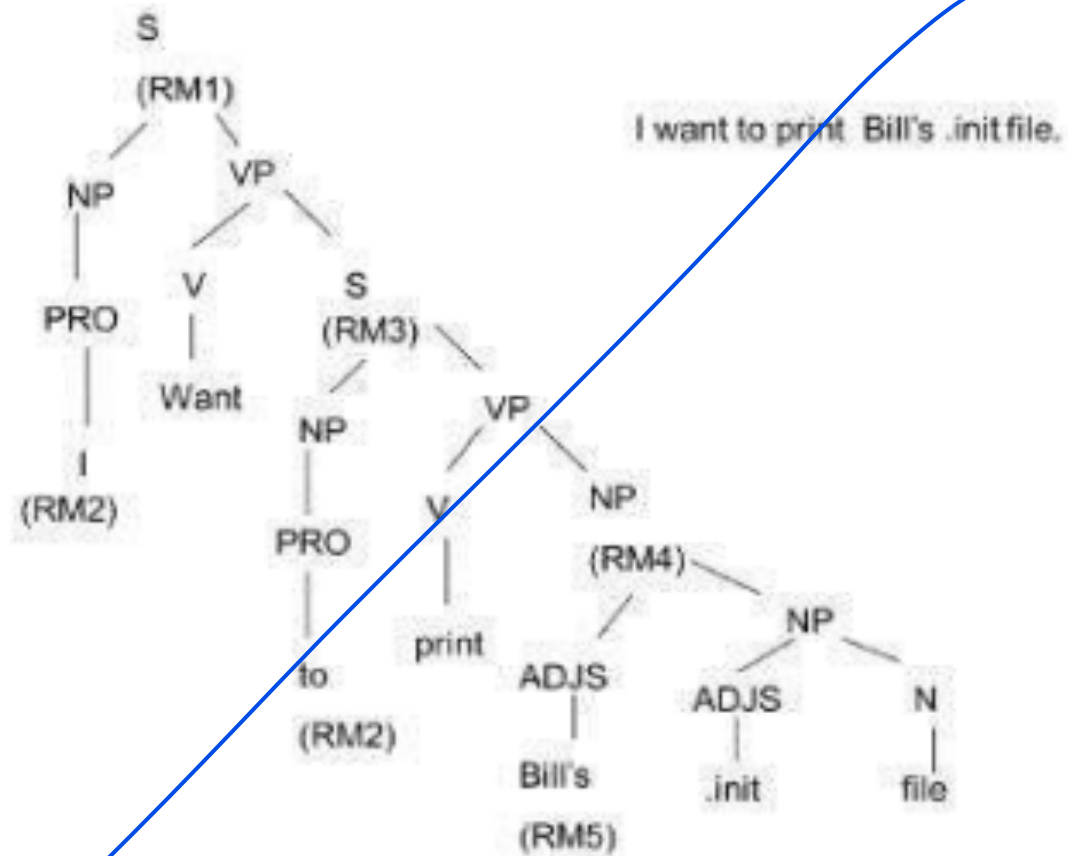
Semantic Analysis

- The semantic analysis must do two important things:

  - 1. It must map individual words into appropriate objects in the knowledge base or database.

  - 2. It must create the correct structures to correspond to the way the meanings of the individual words combine with each other.

Discourse Integration

- Specifically, we do not know whom the pronoun "Bill" or refers to.
- To pin down these references requires an appeal to a model of the current discourse context, from which we can learn that the current user is USER068 and that the only person named "Bill " about whom we could be talking is USER073.
- Once the correct referent for Bill known, we can also determine exactly which file referred to.

Pragmatic Analysis

- The final step toward effective understanding is to decide what to do as a result.
- One possible thing to do to record what was said as a fact and done with it.
- For some sentences, a whose intended effect is clearly declarative, that is the precisely correct thing to do.
- But for other sentences, including this one, the intended effect is different.
- We can discover this intended effect by applying a set of rules that characterize cooperative dialogues.
- The final step in pragmatic processing to translate, from the knowledge-based representation to a command to be executed by the system.

Syntactic Processing

- Syntactic Processing is the step in which a flat input sentence converted into a hierarchical structure that corresponds to the units of meaning in the sentence. This process called parsing.
- It plays an important role in natural language understanding systems for two reasons:

  1. Semantic processing must operate on sentence constituents. If there is no syntactic parsing step, then the semantics system must decide on its own constituents. If parsing is done, on the other hand, it constrains the number of constituents that semantics can consider.

  2. Syntactic parsing is computationally less expensive than is semantic processing. Thus it can play a significant role in reducing overall system complexity.

Syntactic Processing

- Although it is often possible to extract the meaning of a sentence without using grammatical facts, it is not always possible to do so.
- Almost all the systems that are actually used have two main components:

  - 1. A declarative representation, called a grammar, of the syntactic facts about the language.

  - 2. A procedure, called parser that compares the grammar against input sentences to produce parsed structures.

Grammars and Parsers

- The most common way to represent grammars is a set of production rules.
- The first rule can read as "A sentence composed of a noun phrase followed by Verb Phrase" ; the Vertical bar is OR; İ represents the empty string.
- Symbols that further expanded by rules called non-terminal symbols.
- Symbols that correspond directly to strings that must found in an input sentence called terminal symbols.
- Grammar formalism such as this one underlies many linguistic theories, which in turn provide the basis for many natural language understanding systems.
- Pure context-free grammars are not effective for describing natural languages.

Grammars and Parsers

- NLPs have in common with computer language processing systems such as compilers.
- Parsing process takes the rules of the grammar and compares them against the input sentence.
- The simplest structure to build is a Parse Tree, which simply records the rules and how they matched.
- Every node of the parse tree corresponds either to an input word or to a non-terminal in our grammar.
- Each level in the parse tree corresponds to the application of one grammar rule.

Syntactic Processing Augmented Transition Network

- Syntactic Processing is the step in which a flat input sentence is converted into a hierarchical structure that corresponds to the units of meaning in the sentence. This process called parsing.
- It plays an important role in natural language understanding systems for two reasons:

1. Semantic processing must operate on sentence constituents. If there is no syntactic parsing step, then the semantics system must decide on its own constituents. If parsing is done, on the other hand, it constrains the number of constituents that semantics can consider.

2. Syntactic parsing is computationally less expensive than is semantic processing. Thus it can play a significant role in reducing overall system complexity.

Syntactic Processing Augmented Transition Network

Example: A Parse tree for a sentence: Bill Printed the file

```
              S
          /        \
       NP          VP
        |         /    \
       PN        V      NP
        |        |     /    \
       Bill   printed the   NP1
                           /    \
                        ADJS     N
                          |       |
                          ε      file
```
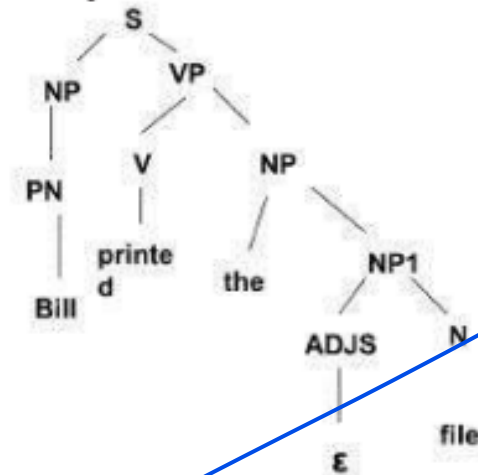
The grammar specifies two things about a language:
1. Its weak generative capacity, by which we mean the set of sentences that contained within the language. This set made up of precisely those sentences that can completely match by a series of rules in the grammar.
2. Its strong generative capacity, by which we mean the structure to assign to each grammatical sentence of the language.
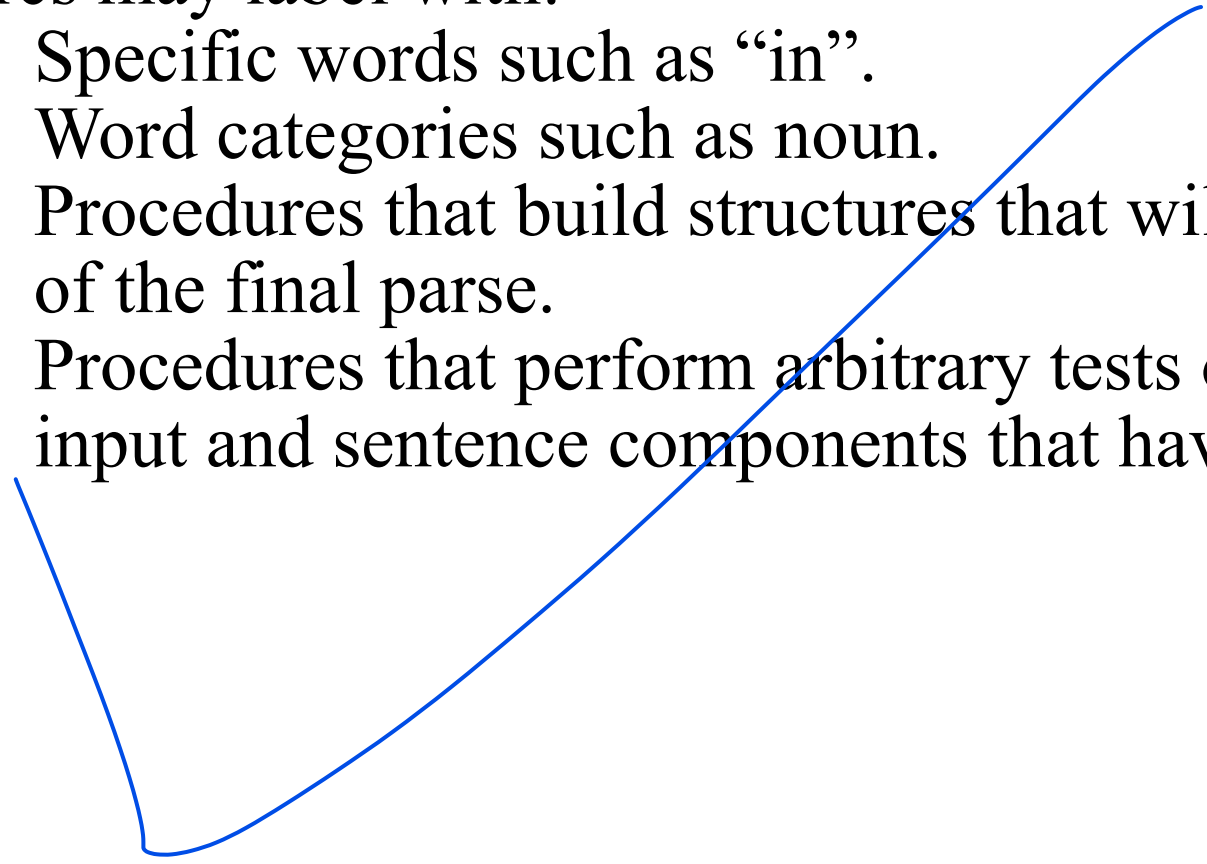
Augmented Transition Network (ATN)

- An augmented transition network is a top-down parsing procedure that allows various kinds of knowledge to incorporated into the parsing system so it can operate efficiently.
- ATNs build on the idea of using finite state machines (Markov model) to parse sentences.
- Instead of building an automaton for a particular sentence, a collection of transition graphs built.
- A grammatically correct sentence parsed by reaching a final state in any state graph.
- Transitions between these graphs simply subroutine calls from one state to any initial state on any graph in the network.
- A sentence determined to be grammatically correct if a final state reached by the last word in the sentence.
- The ATN is similar to a finite state machine in which the class of labels that can attach to the arcs that define the transition between states has augmented.

Augmented Transition Network (ATN)

- Arcs may label with:
    - Specific words such as "in".
    - Word categories such as noun.
    - Procedures that build structures that will form part of the final parse.
    - Procedures that perform arbitrary tests on current input and sentence components that have identified.

Semantic Analysis

- The structures created by the syntactic analyser assigned meanings.
- A mapping made between the syntactic structures and objects in the task domain.
- Structures for which no such mapping is possible may rejected.
- The semantic analysis must do two important things:
  - It must map individual words into appropriate objects in the knowledge base or database.
  - It must create the correct structures to correspond to the way the meanings of the individual words combine with each other.

- Producing a syntactic parse of a sentence is the first step toward understanding it.
- We must produce a representation of the meaning of the sentence.
- Because understanding is a mapping process, we must first define the language into which we are trying to map.
- There is no single definitive language in which all sentence meaning can describe.
- The choice of a target language for any particular natural language understanding program must depend on what is to do with the meanings once they constructed.

Semantic Analysis

- Choice of the target language in Semantic Analysis AI
  - There are two broad families of target languages that used in NL systems, depending on the role that the natural language system playing in a larger system:
  - When natural language considered as a phenomenon on its own, as for example when one builds a program whose goal is to read the text and then answer questions about it. A target language can design specifically to support language processing.
  - When natural language used as an interface language to another program, then the target language must legal input to that other program.
  - Thus the design of the target language driven by the backend program.

Discourse and Pragmatic Processing

- To understand a single sentence, it is necessary to consider the discourse and pragmatic context in which the sentence was uttered.
- There are a number of important relationships that may hold between phrases and parts of their discourse contexts, including:

1. Identical entities. Consider the text:

- Bill had a red balloon. o John wanted it.
- The word "it" should identify as referring to the red balloon. These types of references called anaphora.

2. Parts of entities. Consider the text:

- Sue opened the book she just bought.
- The title page was torn.
- The phrase title page´ should be recognized as part of the book that was just bought.

Discourse and Pragmatic Processing

3. Parts of actions. Consider the text:

- John went on a business trip to New York.
- He left on an early morning flight.
- Taking a flight should recognize as part of going on a trip.

4. Entities involved in actions. Consider the text:

- My house was broken into last week.
- Moreover, They took the TV and the stereo.
- The pronoun "they" should recognize as referring to the burglars who broke into the house.

5. Elements of sets. Consider the text:

- The decals we have in stock are stars, the moon, item and a flag.
- I'll take two moons.
- Moons mean moon decals.

6. Names of individuals:

- Dev went to the movies.

Discourse and Pragmatic Processing

7. Causal chains

- There was a big snow storm yesterday.
- So, The schools closed today.

8. Planning sequences:

- Sally wanted a new car
- She decided to get a job.

9. Implicit presuppositions:

- Did Joe fail CS101?
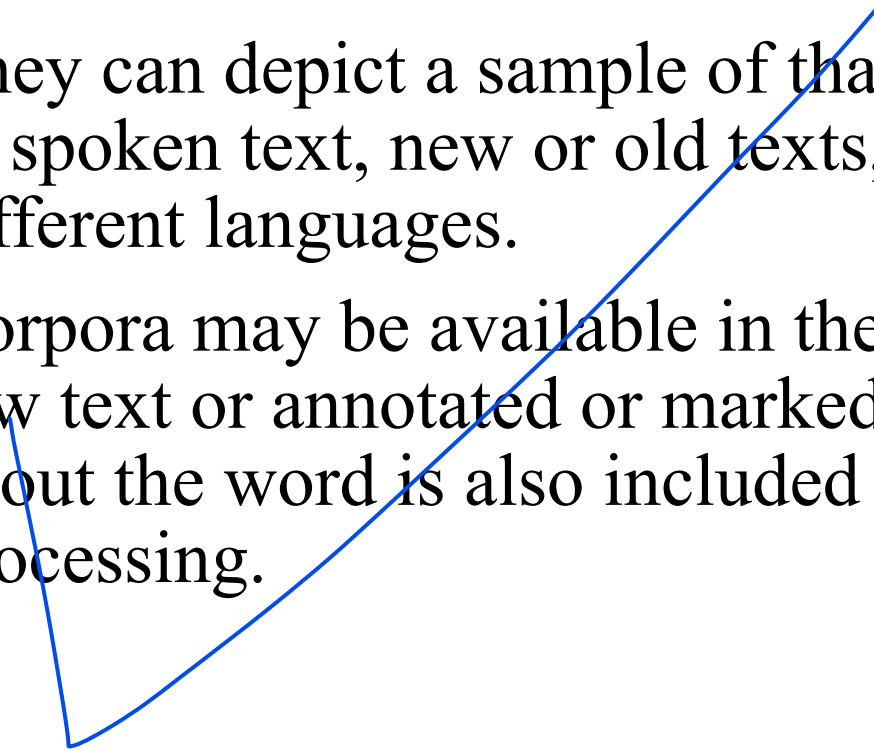
The major focus is on using following kinds of knowledge:

- The current focus of the dialogue.
- A model of each participant's current beliefs.
- The goal-driven character of dialogue.
- The rules of conversation shared by all participants.

STATISTICAL NATURAL LANGUAGE PROCESSING

- Long sentences rise ambiguities when conventional grammars are used to process them.
- The processing of such sentences may require a large number of analyses.
- The statistical information extracted from a large corpus of the concerned language can aid in disambiguation.
- Some issues as below are highlighted :
    - Corpora
    - Counting the elements in a Corpus
    - N-Grams
    - Smoothing

Corpora

- Corpora is a collection of spoken words of a language.
- These are selected and ordered based on some linguistic criteria.
- They can depict a sample of that language such as written or spoken text, new or old texts, texts from either one or different languages.
- Corpora may be available in the form of a collection of raw text or annotated or marked up where information about the word is also included to ease the language processing.

Counting the elements in a Corpus

- Counting the number of words in a corpus can yield valuable information regarding the probability of the occurrence of a word.

- These probabilities can be used to predict a word that will follow it.

- How to use the punctuation marks (, ; ! ?) should be decided as they allows us to understand what information is being conveyed.

- How to use case sensitisation (In, in), singular and plural (book, books) as distinct ones should be decided.

- We arrive at two terms called Types (the number of distinct words in the corpus) and Tokens (the total number of words in the corpus).

N-Grams

- N-grams are basically sequences of N words or strings, where N could assume the value 1,2,3, and so on.
- N=1, is called unigram (one word). N=2 makes a bigram (a sequence of two words), N=3 is trigrams, N=4 is tetragrams and so on.
- Different words in a corpus have their own frequency (i.e. the number of times they occur) in a given corpus.
- Some words have a high frequency like the article "the".
- As an example let us take the number of occurrences of words in the novel - The Scarlet Pimpernel by Baroness Orczy

N-Grams

- Frequencies and probabilities of some single words in a corpus of a novel The Scarlet Pimpernel

Word	Word Frequency	Probability = (Word Frequency)/Total number of words
the	4508	0.051
of	2474	0.028
and	2353	0.027
to	2267	0.026
a	1559	0.018
her	1137	0.013
had	1077	0.012
she	935	0.0107
It	444	0.005
said	399	0.0046
man	174	0.002
Scarlet	98	0.0011
woman	66	0.00076
beautiful	39	0.00045
fool	18	0.00002
However	19	0.00002

N-Grams

- What is the probability of a word being followed by another?
- *A very wealthy...*
- Many times the frequency word "*the*" will not follow in the above sentence.
- The next word is dependent on the previous one.
- Finding probabilities of all such words in the corpus that follow the word wealthy and then choosing the best of them may lead to the construction of a more appropriate sentence.
- Thus we could look at a two-word window and predict the second word using probabilities of finding the second word, given the first word.
- To find that word X which appears after wealthy and has the maximum probability of occurrence in this two-word sequence among all other such words in the corpus is written as *max(P(X | wealthy))*

N-Grams
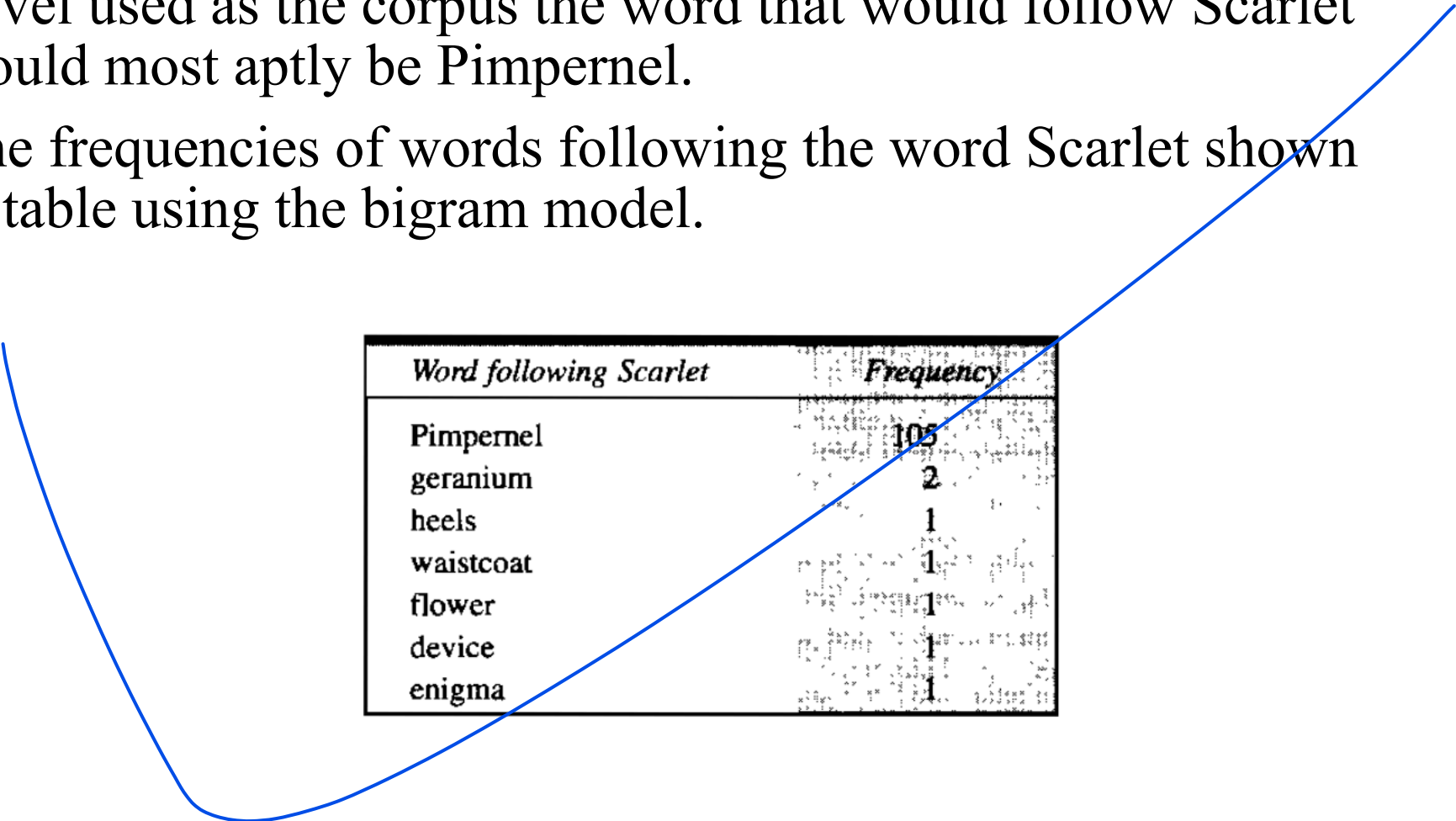
- If there are n words in sentence and assuming the occurrence of each word at their appropriate places to be independent events, the probability $P(w_1,..., w_n,)$ can be expressed using the chain rule as

$$P(W) = P(w_1) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid (w_1, w_2))... P(w_n \mid (w_1, w_2... w_{n-1}))$$

- Computing this probability is very complex as we move to rightwards. The last term would be the most complex to compute.

- A more practical approach to such chaining of probabilities could be to look for only the previous word to compute the probabilities.

- Since we look at only word pairs (viz. a single word previous to the one we are searching) this model is called the bigram model.

N-Grams

- If we use bigram model to seek the next word using the novel used as the corpus the word that would follow Scarlet would most aptly be Pimpernel.

- The frequencies of words following the word Scarlet shown in table using the bigram model.

Word following Scarlet	Frequency
Pimpernel	105
geranium	2
heels	1
waistcoat	1
flower	1
device	1
enigma	1

N-Grams

- When we refer to a previous word and find the probability of the next word, a more apt sentence is created.

- This is called a ***Markov assumption***.

- For a bigram model, the probability can be approximated to the product of all $P(w_i \mid w_{i-1})$ for i varying from 1 to n (n is the number of words in the sentence).

$$P(w_1^n) \approx \prod_{i=1}^{n} P(w_i \mid w_{i-1})$$

- This approximation could be extended to trigrams and further to tetragrams. The approximate probability of finding the next word in case of N-grams is given by

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-N+1}^{n-1})$$

w_1^{n-1} includes the words from w_1 to w_{n-1}. Similarly, w_{n-N+1}^{n-1} means words w_{n-N+1} to w_{n-1}.

N-Grams

- "*He never told her and she had never cared to ask*" can be found using the bigram probability model.
- Each probability term is calculated by finding the number of occurrences of the specific bigram and dividing it by the frequency of the previous word.
- P(she\and) = (Number of occurrence of the bigram ***and she***)(Number of occurrences of the word ***and***)

$$= P(He|<nil>).P(never|He).P(told|never).P(her|told).P(and|her).P(she|and).P(had|she).$$
$$P(never|had)\ P(cared\ |never).P(to|cared).P(ask|to).$$
$$= (207/67675).(3/512).(1/60).(10/31).(6/1137).(34/2353).(168/935).(6/1077).(1/60).(3/11).$$
$$(3/2267)$$

	He	never	told	her	and	she	had	never	cared	to	ask
He [207]	0(0)	3(0.014)	0(0)	0(0)	4(0.019)	0(0)	114(0.55)	3(0.0144)	0(0)	2(0.01)	0(0)
never [60]	0(0)	0(0)	1(0.02)	0(0)	0(0)	0(0)	3(0.05)	0(0)	1(0.02)	0(0)	0(0)
told [31]	0(0)	0(0)	0(0)	10(0.323)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)
her [1137]	1(0.00008)	0(0)	0(0)	0(0)	6(0.005)	0(0)	1(0.00008)	0(0)	0(0)	19(0.0167)	0(0)
and [2353]	34(0.0144)	1(0.00004)	0(0)	26(0.011)	0(0)	34(0.014)	23(0.01)	1(0.00004)	0(0)	38(0.016)	0(0)
she [935]	0(0)	1(0.001)	1(0.001)	0(0)	3(0.003)	0(0)	168(0.18)	1(0.001)	2(0.002)	0(0)	0(0)
had [1077]	8(0.007)	6(0.006)	3(0.003)	0(0)	0(0)	9(0.008)	12(0.111)	6(0.006)	1(0.0001)	8(0.007)	0(0)
never [60]	0(0)	0(0)	1(0.02)	0(0)	0(0)	0(0)	3(0.05)	0(0)	1(0.02)	0(0)	0(0)
cared [11]	0(0)	0(0)	0(0)	0(0)	1(0.090)	0(0)	0(0)	0(0)	0(0)	3(0.273)	0(0)
to [2267]	0(0)	0(0)	0(0)	99(0.044)	1(0.00004)	0(0)	0(0)	0(0)	0(0)	0(0)	2(0.00008)
ask [12]	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)

Smoothing

- N-grams is a good way of predicting the next word but it relies heavily on the corpus which forms the basic training data.

- The corpus is finite and there may be many N-grams missing within it.

- There may be numerous N-grams which should have had non zero probability but are assigned a zero value instead.

- if we assign some non zero probability to avoid the problem to some extent.

- This process is known as smoothing.

Smoothing

- ***Add-One Smoothing :***
- Let P(w_i) be the normal unigram probability without smoothing and N be the total number of words in the corpus then,

$$P(w_i) = c(w_i)/\Sigma c(w_i) = c(w_i)/N$$

- Before calculating the probabilities, the count c of each distinct word within the corpus is incremented by 1.
- The total number of the counts of all words has now increased by D the number of distinct types of words in the language (vocabulary).
- The new probability of a word after add one smoothing can now be computed as-

-

$$P_{add1}(w_i) = \{c(w_i)+1\}/(N+D)$$

Smoothing

- ***Witten-Bell Discounting*** :

- The probability of unseen N-grams is seen as things we saw once (for the first time).

- As we go through the corpus we encounter new N-grams and finally are in a position to ascertain the number of unique N-grams.

- The number of unique N-grams seen in the data is the same as the count H, of the N-grams observed (so far) for the first time.

- Thus (N+H) would mean the sum of the words or tokens seen so far and the unique N-grams types in the corpus.
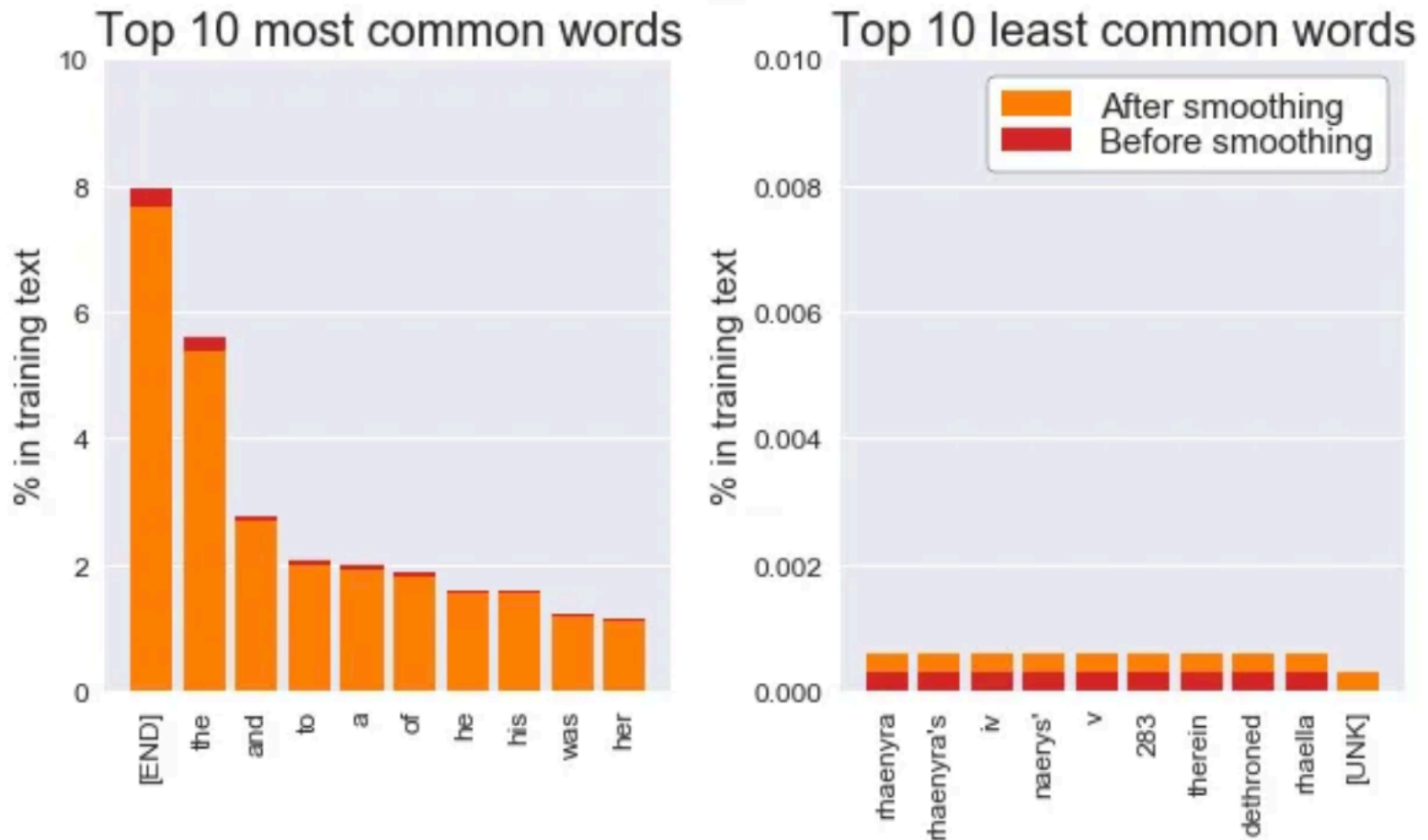
Smoothing

- The total probability mass of all such N-grams occurring for the first time is estimated by computing H/(N+H).
- If I is the total number of N-grams that have never occurred so far, dividing the probability of unseen N-grams by I would distribute it equally among them.
- Thus the probability of an unseen N-gram could be written as-

$$P^u = H/I(N+H)$$

- Since the total probability has to be 1, this extra probability distributed amongst unseen N-grams has to be scooped or discounted from other regions in the probability distribution.
- The probability of the seen N-grams is therefore discounted to aid the generation of the extra probability requirement for the unseen ones as:

$$P^s_k = c_k/(N+H) \text{ where } c_k \text{ is the (non-zero positive) count of } k^{th} \text{ N-gram.}$$

Smoothing

Percentages of top 10 most and least common words (before and after add-one smoothing). Left and right bar charts are not plotted at the same scale.

SPELL CHECKING

- A Spell Checker is a tool used in word processing, character or text recognition systems, speech recognition and generation.

- It is one of the pre-processing formalities that involves identifying words and non words and also suggesting the possible alternatives for its correction.

- Spelling Errors : Typographic errors, Orthographic errors, Phonetic errors.

- Spell Checking Techniques : Non-Word Error Detection, Isolated-Word Error Correction, Context dependent Error detection and correction

SPELL CHECKING

- ***Soundex Algorithm*** :

- The Soundex algorithm is a simple phonetic based spellchecker based on rules found using the phonetics of the language.

- It uses a code to check for the closest word. The code for a word consists of its first letter followed by three numbers that encode the remaining consonants.

- Those consonants that generate the same sound have the same number: Thus the labials B, F, P and V imply the same number viz. 1.

SPELL CHECKING

- ***Soundex algorithm :***

- Remove all punctuation marks and capitalise the letters in the word.

- Retain the first letter of the word.

- Remove any occurrence of the letters - A, E, I, O, U, H, W, Y apart from the very first letter.

- Replace the letters (other than the first) by the numbers shown in Table.

- If two or more adjacent letters, not separated by vowels, have the same numeric value, retain only one of them.

- Return the first four characters; pad with zeroes if there are less than four.

SPELL CHECKING

- Substitutions for generating the Soundex code for some words as shown in table below.

- This algorithm can be used to measure the similarity of two words.

- This measure can then be used to find possible good candidates for correction to be effected for a misspelled word such as rorn (viz. torn , worn, horn).

Letter(s)	Substitute with Integer
B,F,P,V	1
C,G,J,K,S,X,Z	2
D,T	3
L	4
M,N	5
R	6

Word	Soundex Code
Grate, great	G630
Network, network	N362
Henry, Henary	H560
Torn	T650
Worn	W650
Horn	H650

Time for walk....

Expert System

Expert System

- Expert system = knowledge + problem-solving methods.
- A knowledge base that captures the domain-specific knowledge and an inference engine that consists of algorithms for manipulating the knowledge represented in the knowledge base to solve a problem presented to the system.
- Expert systems (ES) are one of the prominent research domains of AI. It is introduced by the researchers at Stanford University, Computer Science Department.

➔ What are Expert Systems?
- The expert systems are the computer applications developed to solve complex problems in a particular domain, at the level of extra-ordinary human intelligence and expertise.

➔ Characteristics of Expert Systems
- High performance
- Understandable
- Reliable
- Highly responsive

Capabilities of Expert Systems

➜ The expert systems are capable of
- Advising
- Instructing and assisting human in decision making
- Demonstrating
- Deriving a solution
- Diagnosing
- Explaining
- Interpreting input
- Predicting results
- Justifying the conclusion
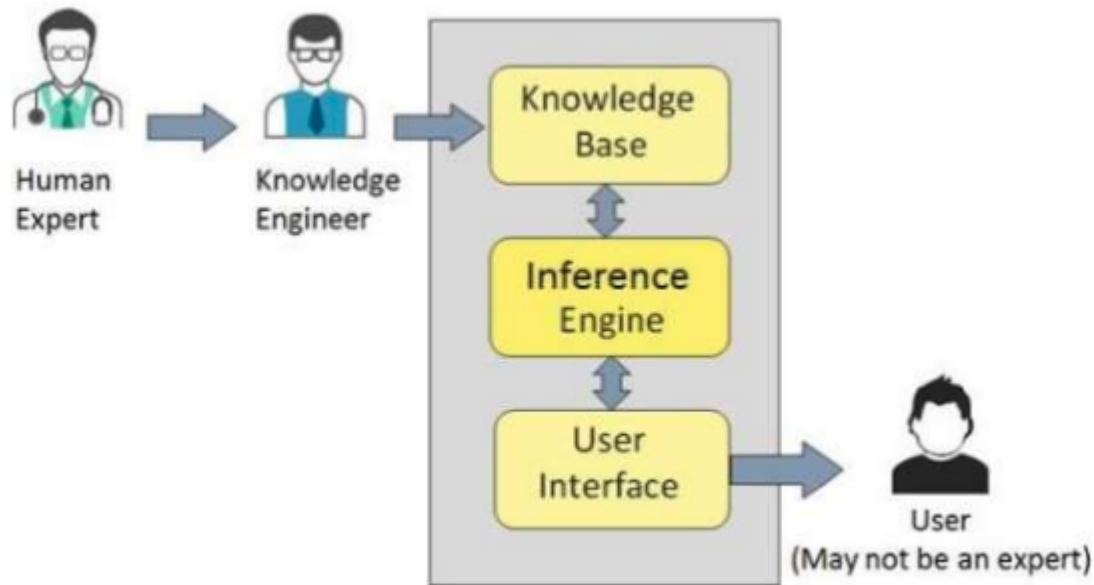- Suggesting alternative options to a problem

➜ They are incapable of
- Substituting human decision makers
- Possessing human capabilities
- Producing accurate output for inadequate knowledge base
- Refining their own knowledge

Components of Expert Systems

➔ The components of ES include
● Knowledge Base
● Inference Engine
● User Interface

Knowledge Base

- It contains domain-specific and high-quality knowledge. Knowledge is required to exhibit intelligence. The success of any ES majorly depends upon the collection of highly accurate and precise knowledge.
➔ **What is Knowledge?**
- The data is collection of facts. The information is organized as data and facts about the task domain. Data, information, and past experience combined together are termed as knowledge.
➔ **Components of Knowledge Base**
- The knowledge base of an ES is a store of both, factual and heuristic knowledge.
  - **Factual Knowledge:** It is the information widely accepted by the Knowledge Engineers and scholars in the task domain.
  - **Heuristic Knowledge :** It is about practice, accurate judgement, one's ability of evaluation, and guessing.

Knowledge Base

➔ **Knowledge representation**

● It is the method used to organise and formalise the knowledge in the knowledge base. It is in the form of IF-THEN-ELSE rules.

➔ **Knowledge Acquisition**

● The success of any expert system majorly depends on the quality, completeness, and accuracy of the information stored in the knowledge base.

● The knowledge base is formed by readings from various experts, scholars, and the Knowledge Engineers. The knowledge engineer is a person with the qualities of empathy, quick learning, and case analyzing skills.

● He acquires information from subject expert by recording, interviewing, and observing him at work, etc. He then categorizes and organizes the information in a meaningful way, in the form of IF-THEN-ELSE rules, to be used by interference machine. The knowledge engineer also monitors the development of the ES.

Inference Engine

- Use of efficient procedures and rules by the Inference Engine is essential in deducting a correct, flawless solution.
- In case of knowledge-based ES, the Inference Engine acquires and manipulates the knowledge from the knowledge base to arrive at a particular solution.

In case of rule based ES, it:

- Applies rules repeatedly to the facts, which are obtained from earlier rule application.
- Adds new knowledge into the knowledge base if required.
- Resolves rules conflict when multiple rules are applicable to a particular case.

To recommend a solution, the Inference Engine uses the following strategies

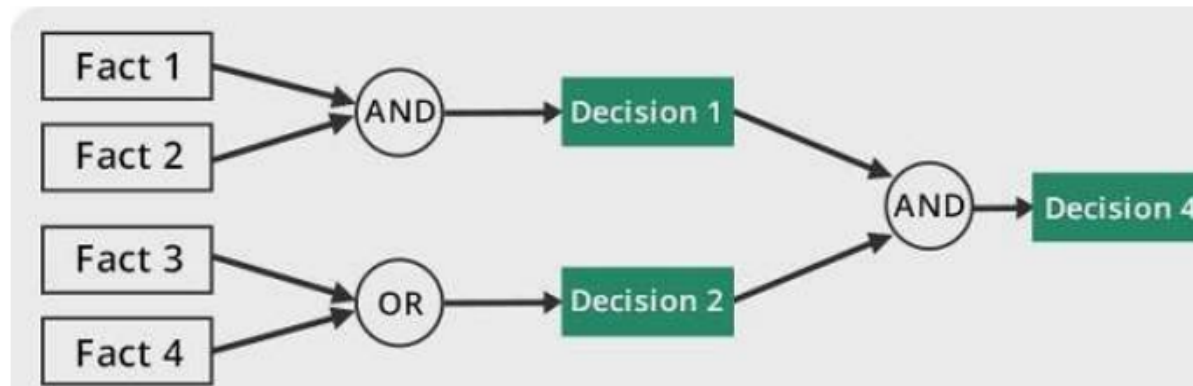- Forward Chaining
- Backward Chaining

Forward chaining

- It is a strategy of an expert system to answer the question, "What can happen next?"

- Here, the Inference Engine follows the chain of conditions and derivations and finally deduces the outcome.

- It considers all the facts and rules, and sorts them before concluding to a solution.

- This strategy is followed for working on conclusion, result, or effect. For example, prediction of share market status as an effect of changes in interest rates.

Forward chaining

- Forward chaining starts with the data available and uses the inference rules to extract more data until a desired goal is reached.

- An inference engine using forward chaining searches the inference rules until it finds one in which the if clause is known to be true.

- It then concludes the then clause and adds this information to its data.

- It continues to do this until a goal is reached. Because the data available determines which inference rules are used, this method is also classified as data driven.

Backward chaining

- On the basis of what has already happened, the Inference Engine tries to find out which conditions could have happened in the past for this result.

- This strategy is followed for finding out cause or reason. For example, diagnosis of blood cancer in humans
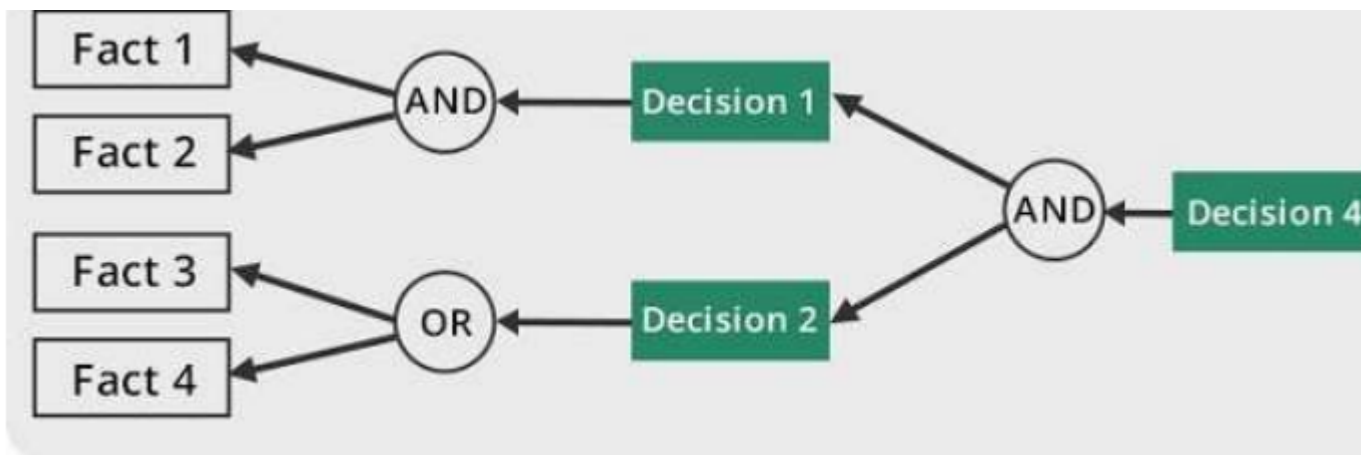
Backward chaining

- Backward chaining starts with a list of goals and works backwards to see if there is data which will allow it to conclude any of these goals.

- An inference engine using backward chaining would search the inference rules until it finds one which has a then clause that matches a desired goal.

- If the if clause of that inference rule is not known to be true, then it is added to the list of goals.

User Interface

- User interface provides interaction between user of the ES and the ES itself.
- It is generally Natural Language Processing so as to be used by the user who is well-versed in the task domain.
- The user of the ES need not be necessarily an expert in Artificial Intelligence. It explains how the ES has arrived at a particular recommendation.
- The explanation may appear in the following forms:
  - Natural language displayed on screen.
  - Verbal narrations in natural language.
  - Listing of rule numbers displayed on the screen.

User Interface

- The user interface makes it easy to trace the credibility of the deductions.

- Requirements of Efficient ES User Interface.
  - It should help users to accomplish their goals in shortest possible way.
  - It should be designed to work for user' existing or desired work practices.
  - Its technology should be adaptable to user's requirements; not the other way round.
  - It should make efficient use of user input.

Expert System Limitations

- No technology can offer easy and complete solution. Large systems are costly, require significant development time, and computer resources.

- ESs have their limitations which include
  - Limitations of the technology
  - Difficult knowledge acquisition
  - ES are difficult to maintain
  - High development costs

Applications of Expert System

Application	Description
Design Domain	Camera lens design, automobile design.
Medical Domain	Diagnosis Systems to deduce cause of disease from observed data, conduction medical operations on humans.
Monitoring Systems	Comparing data continuously with observed system or with prescribed behavior such as leakage monitoring in long petroleum pipeline.
Process Control Systems	Controlling a physical process based on monitoring.
Knowledge Domain	Finding out faults in vehicles, computers.
Finance/Commerce	Detection of possible fraud, suspicious transactions, stock market trading, Airline scheduling, cargo scheduling.

Expert System Technology

- There are several levels of ES technologies available.
- Expert systems technologies include
- Expert System Development Environment :The ES development environment includes hardware and tools. They are
  - Workstations, minicomputers, mainframes.
  - High level Symbolic Programming Languages such as LISt Programming (LISP) and PROgrammation en LOGique (PROLOG).
  - Large databases.

- Tools :- They reduce the effort and cost involved in developing an expert system to large extent.
  - Powerful editors and debugging tools with multi-windows.
  - They provide rapid prototyping
  - Have Inbuilt definitions of model, knowledge representation, and inference design.

- A shell is nothing but an expert system without knowledge base.
- A shell provides the developers with knowledge acquisition, inference engine, user interface, and explanation facility.
- For example, few shells are given below
  - Java Expert System Shell (JESS) that provides fully developed Java API for creating an expert system.
  - Vidwan, a shell developed at the National Centre for Software Technology, Mumbai in 1993. It enables knowledge encoding in the form of IF-THEN rules

General Steps for Development of Expert Systems

- Identify Problem Domain
  - The problem must be suitable for an expert system to solve it.
  - Find the experts in task domain for the ES project.
  - Establish cost-effectiveness of the system.

- Design the System
  - Identify the ES Technology
  - Know and establish the degree of integration with the other systems and databases.
  - Realize how the concepts can represent the domain knowledge best.

- Develop the Prototype
- From Knowledge Base: The knowledge engineer works to
  - Acquire domain knowledge from the expert.
  - Represent it in the form of If-THEN-ELSE rules.

Development of Expert Systems: General Steps

- Test and Refine the Prototype
  - The knowledge engineer uses sample cases to test the prototype for any deficiencies in performance.
  - End users test the prototypes of the ES.

- Develop and Complete the ES
  - Test and ensure the interaction of the ES with all elements of its environment, including end users, databases, and other information systems.
  - Document the ES project well.
  - Train the user to use ES.

- Maintain the ES
  - Keep the knowledge base up-to-date by regular review and update.
  - Cater for new interfaces with other information systems, as those systems evolve.

Benefits of Expert Systems

- Availability:- They are easily available due to mass production of software.
- Less Production Cost: Production cost is reasonable. This makes them affordable.
- Speed: They offer great speed. They reduce the amount of work an individual puts in.
- Less Error Rate: Error rate is low as compared to human errors.
- Reducing Risk: They can work in the environment dangerous to humans.
- Steady response : They work steadily without getting emotional, tensed or fatigued.

SW Architecture.

The following general points about expert systems and their architecture have been outlined:

1. The sequence of steps taken to reach a conclusion is dynamically synthesised with each new case. The sequence is not explicitly programmed at the time that the system is built.

2. Expert systems can process multiple values for any problem parameter. This permits more than one line of reasoning to be pursued and the results of incomplete (not fully determined) reasoning to be presented.

3. Problem solving is accomplished by applying specific knowledge rather than specific technique. This is a key idea in expert systems technology. It reflects the belief that human experts do not process their knowledge differently from others, but they do possess different knowledge. With this philosophy, when one finds that their expert system does not produce the desired results, work begins to expand the knowledge base, not to re-program the procedures.

End User

- There are two styles of user-interface design followed by expert systems. In the original style of user interaction, the software takes the end-user through an interactive dialog.

- In the following example, a backward-chaining system seeks to determine a set of restaurants to recommend:

Q. Do you know which restaurant you want to go to?

A. No

Q. Is there any kind of food you would particularly like? A. No

Q. Do you like spicy food?

A. No

Q. Do you usually drink wine with meals?

A. Yes

Q. When you drink wine, is it French wine? A. Yes

Participant

- There are generally three individuals having an interaction in an expert system. Primary among these is the end-user, the individual who uses the system for its problem solving assistance.
- In the construction and maintenance of the system there are two other roles: the problem domain expert who builds the system and supplies the knowledge base, and a knowledge engineer who assists the experts in determining the representation of their knowledge, enters this knowledge into an explanation module and who defines the inference technique required to solve the problem.
- Usually the knowledge engineer will represent the problem solving activity in the form of rules.
- When these rules are created from domain expertise, the knowledge base stores the rules of the expert system.

Inference Rule

- An understanding of the "inference rule" concept is <mark>important to understand expert systems</mark>.

- An inference rule is a conditional statement with two parts: an if clause and a then clause. This rule is what gives expert systems the ability to find solutions to diagnostic and prescriptive problems.

- An example of an inference rule is: If the restaurant choice includes French and the occasion is romantic, Then the restaurant choice is definitely Paul Bocuse

Procedure node interface

- The function of the procedure node interface is to receive information from the procedures coordinator and create the appropriate procedure call.
- The ability to call a procedure and receive information from that procedure can be viewed as simply a generalisation of input from the external world.
- In some earlier expert systems external information could only be obtained in a predetermined manner, which only allowed certain information to be acquired.
- Through the knowledge base, this expert system disclosed in the cross-referenced application can invoke any procedure allowed on its host system.
- This makes the expert system useful in a much wider class of knowledge domains than if it had no external access or only limited external access.

Procedure node interface

- The knowledge that is represented in the system appears in the rule base.
- In the rule base described in the cross-referenced applications, there are basically four different types of objects, with the associated information:

1. Classes: Questions asked to the user.

2. Parameters: Place holders for character strings which may be variables that can be inserted into a class question at the point in the question where the parameter is positioned.

3. Procedures: Definitions of calls to external procedures.

4. Rule nodes: Inferences in the system are made by a tree structure which indicates the rules or logic mimicking human reasoning. The nodes of these trees are called rule nodes. There are several different types of rule nodes.

Expert Systems/Shells

- The E.S shell simplifies the process of creating a knowledge base.

- It is the shell that actually processes the information entered by a user relates it to the concepts contained in the knowledge base and provides an assessment or solution for a particular problem.