Tip: Use graphics to set the tone of the speech.

Knowledge Representation

Chapter 3

Representational Systems

- The function of any representation scheme is to capture - often called *abstract out* – the critical features of a problem domain and make that information accessible to a problem solving procedure.
- A representation should provide a *natural* framework for expressing problem-solving knowledge; it should make that knowledge available to the computer and assist the programmer in its organization.

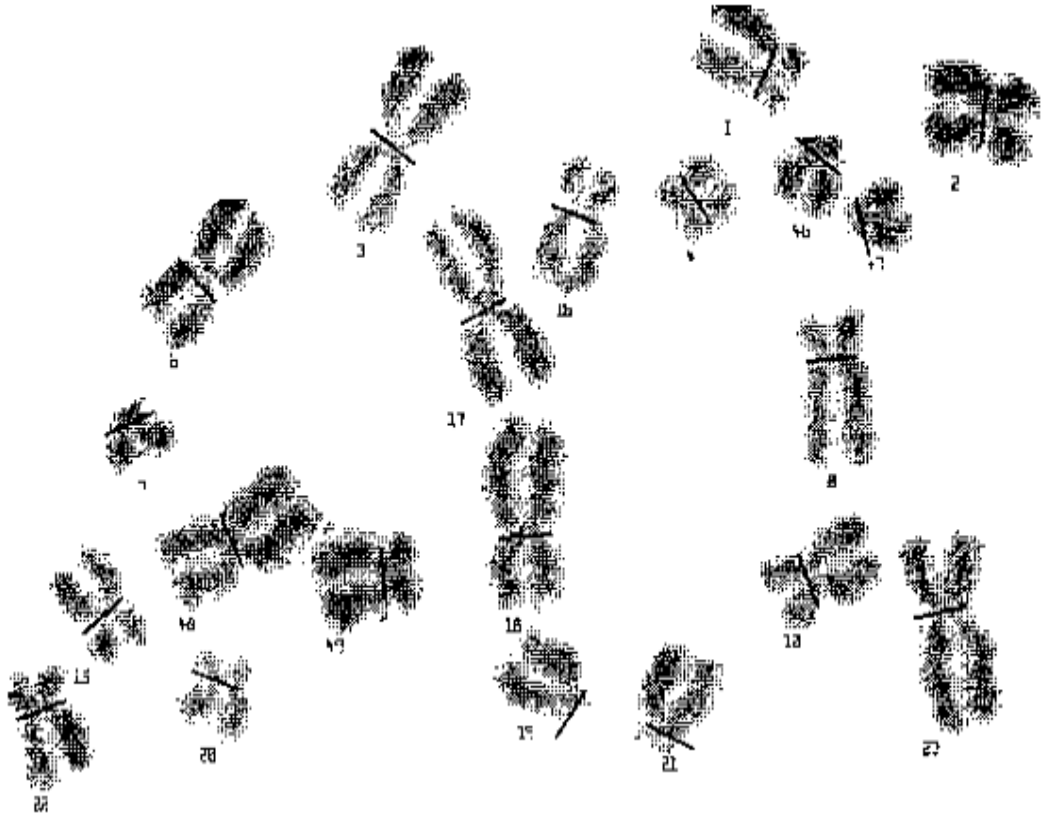
Example: An Array

- For many problems, an array is more natural and efficient than the memory architecture implemented in computer hardware.
- This gain in naturalness and efficiency involves compromises in expressiveness, as illustrated by the following example from image processing.

Example: An Array

Figure is a digitized image of human chromosomes in a stage called *metaphase*.

The image is processed to determine the number and structure of the chromosomes, looking for breaks, missing pieces, and other abnormalities.



Example: An Array

- The visual scene is made up of a number of pixels, has both a location and a number value representing its intensity or *gray level*.

Example: An Array

- The visual scene is made up of a number of pixels, has both a location and a number value representing its intensity or *gray level*.
- It is natural, then, to collect the entire scene into a 2D array where the row and column address gives the location of a pixel (X and Y coordinates) and content of the array element is the gray level at that point.

Example: An Array

- The visual scene is made up of a number of pixels, has both a location and a number value representing its intensity or *gray level*.
- It is natural, then, to collect the entire scene into a 2D array where the row and column address gives the location of a pixel (X and Y coordinates) and content of the array element is the gray level at that point.
- Algorithms are designed to perform operations like looking for isolated points to remove noise from the image, finding threshold levels for discerning objects and edges, summing contiguous elements to determine size or density, etc. transforming the picture point data.

Example: An Array

- The visual scene is made up of a number of pixels, has both a location and a number value representing its intensity or *gray level*.
- It is natural, then, to collect the entire scene into a 2D array where the row and column address gives the location of a pixel (X and Y coordinates) and content of the array element is the gray level at that point.
- Algorithms are designed to perform operations like looking for isolated points to remove noise from the image, finding threshold levels for discerning objects and edges, summing contiguous elements to determine size or density, etc. transforming the picture point data.
- This task would be quite cumbersome using other representations such as the predicate calculus, records, or assembly code, because these do not have a natural fit with the material being represented.

Example: An Array

- When we represent the picture as an array of pixel points, we sacrifice fineness of resolution (compare a photo in a newspaper to the original print of the same picture).
- In addition, pixel arrays cannot express the deeper semantic organization of the image.
  - For example, a pixel array cannot represent the organization of chromosomes in a single cell nucleus, their genetic function, or the role of metaphase in cell division.
- This knowledge is more easily captured using a representation such as predicate calculus or semantic networks.

Summary

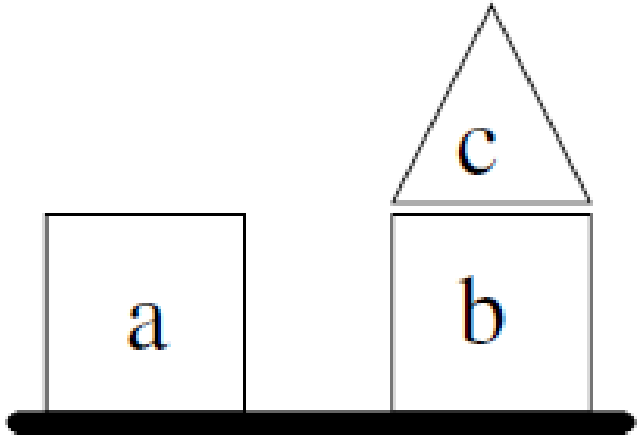
- In summary, a representational scheme should be adequate to express all of the necessary information, support efficient execution of the resulting code, and provide a natural scheme for expressing the required knowledge.

Knowledge Representation

- In general, the problems AI attempts to solve do not lend themselves to the representations offered by more traditional formalisms such as arrays.
- Artificial intelligence is concerned with qualitative rather than quantitative problem solving, with reasoning rather than numeric calculation, and with organizing large and varied amounts of knowledge rather than implementing a single, well-defined algorithm.

Another Example

- For example, consider this figure, the arrangement of blocks on a table.
- Suppose we wish to capture the properties and relations required to control a robot arm.
- We must be determining which blocks are stacked on other blocks and which blocks have clear tops so that they can be picked up.
- The *predicate calculus* offers a medium to capture this descriptive information.



Another Example (continued)

- The first word of each expression (on, ontable, etc.) is a *predicate* denoting some property or relationship among its *arguments* (appearing in the parentheses).
- The arguments are symbols denoting objects (blocks) in the domain. The collection of logical clauses describes the important properties and relationships of this *blocks world*:
  - clear(c)
  - clear(a)
  - ontable(a)
  - ontable(b)
  - on(c, b)
  - cube(b)
  - cube(a)
  - pyramid(c)

Another Example (continued)

- We want to define a test to determine whether a block is clear, that is, has nothing stacked on top of it.

Another Example (continued)

- We want to define a test to determine whether a block is clear, that is, has nothing stacked on top of it.
- This is important if the robot hand is to pick it up or stack another block on top of it. We can define a general rule:
  - $\forall X \neg \exists Y \; on(Y,X) \Rightarrow clear(X)$
  - This is read “for all X, X is clear if there does not exist a Y such that Y is on X.”

Another Example (continued)

- We want to define a test to determine whether a block is clear, that is, has nothing stacked on top of it.
- This is important if the robot hand is to pick it up or stack another block on top of it. We can define a general rule:
  - $\forall X \neg \exists Y \; on(Y,X) \Rightarrow clear(X)$
  - This is read “for all X, X is clear if there does not exist a Y such that Y is on X.”
- This general rule can be applied to a variety of situations by substituting different block names, a, b, c, etc., for X and Y.

Another Example (continued)

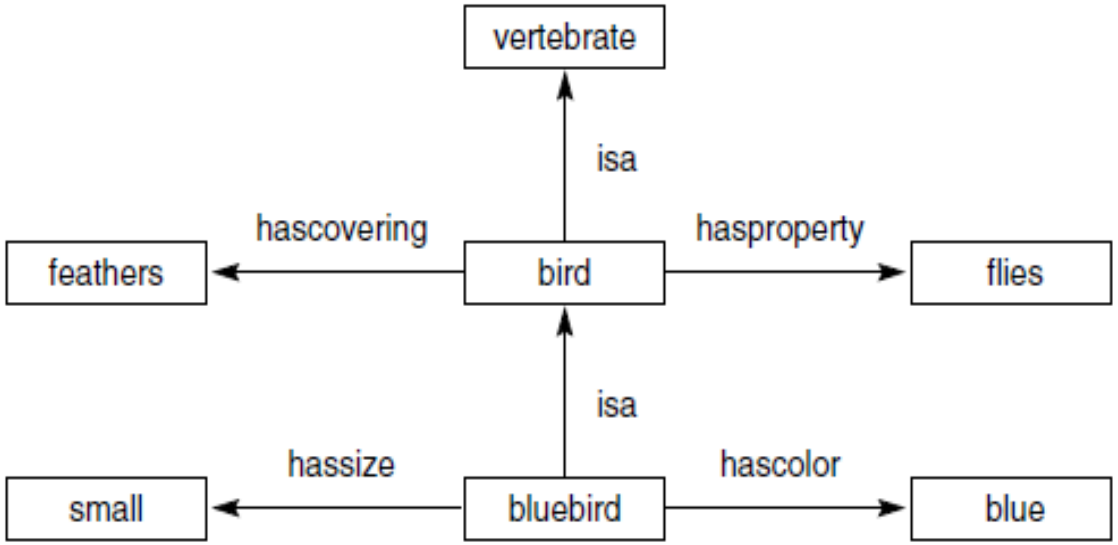
- We want to define a test to determine whether a block is clear, that is, has nothing stacked on top of it.
- This is important if the robot hand is to pick it up or stack another block on top of it. We can define a general rule:
  - $\forall X \neg \exists Y \; on(Y,X) \Rightarrow clear(X)$
  - This is read “for all X, X is clear if there does not exist a Y such that Y is on X.”
- This general rule can be applied to a variety of situations by substituting different block names, a, b, c, etc., for X and Y.
- By supporting general inference rules, predicate calculus allows economy of representation, as well as the possibility of designing systems that are flexible and general enough to respond intelligently to a range of situations.

Other Examples

- It is often not sufficient, for example, to describe a car by simply listing its component parts; we may want to describe the ways in which those parts are combined, interactions between them.
- This view of structure is essential to a range of situations including taxonomic information, such as the classification of plants by genus and species, or a description of complex objects such as a diesel engine or a human body in terms of their component parts.
- Simple description of a bluebird can be “a bluebird is a small blue-colored bird and a bird is a feathered flying vertebrate”, which may be represented as the set of logical predicates:
- hassize(bluebird,small)
- hascolor(bluebird,blue)
- isa(bluebird,bird)
- hascovering(bird,feathers)
- hasproperty(bird,flies)
- isa(bird,vertebrate)

Bluebird Representation: A Semantic Network

In the bluebird illustration, the program needs only follow one link to see that a blubird flies and two links to determine that a bluebird is a vertebrate.



Semantic Networks

- Perhaps the most important application for semantic networks is to represent meanings for language understanding programs.
- When it is necessary to understand a child's story, the details of a journal article, or the contents of a web page, semantic networks may be used to encode the information and relationships that reflect the knowledge in that application.

The Propositional Calculus

- The propositional calculus and the predicate calculus are first of all languages.
- Using their words, phrases, and sentences, we can represent and reason about properties and relationships in the world.

Symbols and Sentences

- The first step in describing a language is to introduce the pieces that make it up: its set of symbols.

Propositional Calculus Symbols

- The *symbols* of propositional calculus are the propositional symbols:
  - P, Q, R, S, …
  - truth symbols: true, false
  - and connectives:
    - ∧,
    - ∨,
    - ¬,
    - →,
    - ≡
- Propositional symbols denote *propositions*, or statements about the world that may be either true or false, such as “the car is red” or “water is wet.”
- Propositions are denoted by uppercase letters near the end of the English alphabet.

Propositional Calculus Sentences

- Sentences in the propositional calculus are formed from these atomic symbols according to the following rules:
  - Every propositional symbol and truth symbol is a sentence.
    - For example: true, P, Q, and R are sentences.
  - The *negation* of a sentence is a sentence.
    - For example: ¬ P and ¬ false are sentences.
  - The *conjunction*, or *and*, of two sentences is a sentence.
    - For example: P ∧ ¬ P is a sentence.

Propositional Calculus Sentences

- The *disjunction*, or *or*, of two sentences is a sentence.
  - For example: P ∨ ¬ P is a sentence.
- The *implication* of one sentence from another is a sentence.
  - For example: P → Q is a sentence.
- The *equivalence* of two sentences is a sentence.
  - For example: P ∨ Q ≡ R is a sentence.

Propositional Calculus Sentences

- Legal sentences are also called *well-formed formulas* or *WFFs*.
- In expressions of the form P ∧ Q, P and Q are called the *conjuncts*.
- In P ∨ Q, P and Q are referred to as *disjuncts*.
- In an implication, P → Q, P is the *premise* or *antecedent* and Q, the *conclusion* or *consequent*.

Propositional Calculus Sentences

- In propositional calculus sentences, the symbols ( ) and [ ] are used to group symbols into subexpressions and so to control their order of evaluation and meaning.
- For example,
  - (P ∨ Q) ≡ R is quite different from P ∨ (Q ≡ R), as can be demonstrated using truth tables.

Expression

- An expression is a sentence, or well-formed formula, of the propositional calculus if and only if it can be formed of legal symbols through some sequence of these rules.
- For example,
  - $((P \wedge Q) \rightarrow R) \equiv \neg P \vee \neg Q \vee R$
- is a well-formed sentence in the propositional calculus because:
  - P, Q, and R are propositions and thus sentences.
  - $P \wedge Q$, the conjunction of two sentences, is a sentence.
  - $(P \wedge Q) \rightarrow R$, the implication of a sentence for another, is a sentence.
  - $\neg P$ and $\neg Q$, the negations of sentences, are sentences.
  - $\neg P \vee \neg Q$, the disjunction of two sentences, is a sentence.
  - $\neg P \vee \neg Q \vee R$, the disjunction of two sentences, is a sentence.
  - $((P \wedge Q) \rightarrow R) \equiv \neg P \vee \neg Q \vee R$, the equivalence of two sentences, is a sentence.

The Semantics of the Propositional Calculus

- Because AI programs must reason with their representational structures, it is important to demonstrate that the truth of their conclusions depends only on the truth of their initial knowledge or premises, i.e., that logical errors are not introduced by the inference procedures.
- A precise treatment of semantics is essential to this goal.

The Semantics of the Propositional Calculus

- A proposition symbol corresponds to a statement about the world.
- For example, P may denote the statement “it is raining” or Q, the statement “I live in a brown house.”
- A proposition must be either true or false, given some state of the world.
- The truth value assignment to propositional sentences is called an *interpretation*, an assertion about their truth in some *possible world*.

The Semantics of the Propositional Calculus

- Formally, an interpretation is a mapping from the propositional symbols into the set {T, F}.
- As mentioned in the previous section, the symbols true and false are part of the set of well-formed sentences of the propositional calculus; i.e., they are distinct from the truth value assigned to a sentence.
- To enforce this distinction, the symbols T and F are used for truth value assignment.

The Semantics of the Propositional Calculus

- Each possible mapping of truth values onto propositions corresponds to a possible world of interpretation.
- For example, if P denotes the proposition “it is raining” and Q denotes “I am at work,” then the set of propositions {P, Q} has four different functional mappings into the truth values {T, F}. These mappings correspond to four different interpretations.

Propositional Calculus Semantics

- An *interpretation* of a set of propositions is the assignment of a truth value, either T or F, to each propositional symbol.
- The symbol true is always assigned T, and the symbol false is assigned F.

Propositional Calculus Semantics

- The interpretation or truth value for sentences is determined by:
  - The truth assignment of *negation*, ¬ P, where P is any propositional symbol, is F if the assignment to P is T, and T if the assignment to P is F.

Propositional Calculus Semantics

- The interpretation or truth value for sentences is determined by:
  - The truth assignment of *negation*, ¬ P, where P is any propositional symbol, is F if the assignment to P is T, and T if the assignment to P is F.
  - The truth assignment of *conjunction*, ∧, is T only when both conjuncts have truth value T; otherwise it is F.

Propositional Calculus Semantics

- The interpretation or truth value for sentences is determined by:
  - The truth assignment of *negation*, ¬ P, where P is any propositional symbol, is F if the assignment to P is T, and T if the assignment to P is F.
  - The truth assignment of *conjunction*, ∧, is T only when both conjuncts have truth value T; otherwise it is F.
  - The truth assignment of *disjunction*, ∨, is F only when both disjuncts have truth value F; otherwise it is T.

Propositional Calculus Semantics

- The interpretation or truth value for sentences is determined by:
  - The truth assignment of *negation*, ¬ P, where P is any propositional symbol, is F if the assignment to P is T, and T if the assignment to P is F.
  - The truth assignment of *conjunction*, ∧, is T only when both conjuncts have truth value T; otherwise it is F.
  - The truth assignment of *disjunction*, ∨, is F only when both disjuncts have truth value F; otherwise it is T.
  - The truth assignment of *implication*, →, is F only when the premise or symbol before the implication is T and the truth value of the consequent or symbol after the implication is F; otherwise it is T.

Propositional Calculus Semantics

- The interpretation or truth value for sentences is determined by:
  - The truth assignment of *negation*, ¬ P, where P is any propositional symbol, is F if the assignment to P is T, and T if the assignment to P is F.
  - The truth assignment of *conjunction*, ∧, is T only when both conjuncts have truth value T; otherwise it is F.
  - The truth assignment of *disjunction*, ∨, is F only when both disjuncts have truth value F; otherwise it is T.
  - The truth assignment of *implication*, →, is F only when the premise or symbol before the implication is T and the truth value of the consequent or symbol after the implication is F; otherwise it is T.
  - The truth assignment of *equivalence*, ≡, is T only when both expressions have the same truth assignment for all possible interpretations; otherwise it is F.

Truth Tables

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Propositional Calculus Laws

- ¬ (¬ P) ≡ P

Propositional Calculus Laws

- ¬ (¬ P) ≡ P
- (P ∨ Q) ≡ (¬ P → Q)

Propositional Calculus Laws

- ¬ (¬ P) ≡ P
- (P ∨ Q) ≡ (¬ P → Q)
- the contrapositive law: (P → Q) ≡ (¬ Q→¬P)

Propositional Calculus Laws

- ¬ (¬ P) ≡ P
- (P ∨ Q) ≡ (¬ P → Q)
- the contrapositive law: (P → Q) ≡ (¬ Q→¬P)
- de Morgan's law: ¬ (P ∨ Q) ≡ (¬ P ∧ ¬ Q) and ¬ (P ∧ Q) ≡ (¬ P ∨¬ Q)

Propositional Calculus Laws

- ¬ (¬ P) ≡ P
- (P ∨ Q) ≡ (¬ P → Q)
- the contrapositive law: (P → Q) ≡ (¬ Q→¬P)
- de Morgan's law: ¬ (P ∨ Q) ≡ (¬ P ∧ ¬ Q) and ¬ (P ∧ Q) ≡ (¬ P ∨¬ Q)
- the commutative laws: (P ∧ Q) ≡ (Q ∧ P) and (P ∨ Q) ≡ (Q ∨ P)

Propositional Calculus Laws

- ¬ (¬ P) ≡ P
- (P ∨ Q) ≡ (¬ P → Q)
- the contrapositive law: (P → Q) ≡ (¬ Q→¬P)
- de Morgan's law: ¬ (P ∨ Q) ≡ (¬ P ∧ ¬ Q) and ¬ (P ∧ Q) ≡ (¬ P ∨¬ Q)
- the commutative laws: (P ∧ Q) ≡ (Q ∧ P) and (P ∨ Q) ≡ (Q ∨ P)
- the associative law: ((P ∧ Q) ∧ R) ≡ (P ∧ (Q ∧ R))

Propositional Calculus Laws

- ¬ (¬ P) ≡ P
- (P ∨ Q) ≡ (¬ P → Q)
- the contrapositive law: (P → Q) ≡ (¬ Q→¬P)
- de Morgan's law: ¬ (P ∨ Q) ≡ (¬ P ∧ ¬ Q) and ¬ (P ∧ Q) ≡ (¬ P ∨¬ Q)
- the commutative laws: (P ∧ Q) ≡ (Q ∧ P) and (P ∨ Q) ≡ (Q ∨ P)
- the associative law: ((P ∧ Q) ∧ R) ≡ (P ∧ (Q ∧ R))
- the associative law: ((P ∨ Q) ∨ R) ≡ (P ∨ (Q ∨ R))

Propositional Calculus Laws

- ¬ (¬ P) ≡ P
- (P ∨ Q) ≡ (¬ P → Q)
- the contrapositive law: (P → Q) ≡ (¬ Q→¬P)
- de Morgan's law: ¬ (P ∨ Q) ≡ (¬ P ∧ ¬ Q) and ¬ (P ∧ Q) ≡ (¬ P ∨¬ Q)
- the commutative laws: (P ∧ Q) ≡ (Q ∧ P) and (P ∨ Q) ≡ (Q ∨ P)
- the associative law: ((P ∧ Q) ∧ R) ≡ (P ∧ (Q ∧ R))
- the associative law: ((P ∨ Q) ∨ R) ≡ (P ∨ (Q ∨ R))
- the distributive law: P ∨ (Q ∧ R) ≡ (P ∨ Q) ∧ (P ∨ R)

Propositional Calculus Laws

- ¬ (¬ P) ≡ P
- (P ∨ Q) ≡ (¬ P → Q)
- the contrapositive law: (P → Q) ≡ (¬ Q→¬P)
- de Morgan's law: ¬ (P ∨ Q) ≡ (¬ P ∧ ¬ Q) and ¬ (P ∧ Q) ≡ (¬ P ∨¬ Q)
- the commutative laws: (P ∧ Q) ≡ (Q ∧ P) and (P ∨ Q) ≡ (Q ∨ P)
- the associative law: ((P ∧ Q) ∧ R) ≡ (P ∧ (Q ∧ R))
- the associative law: ((P ∨ Q) ∨ R) ≡ (P ∨ (Q ∨ R))
- the distributive law: P ∨ (Q ∧ R) ≡ (P ∨ Q) ∧ (P ∨ R)
- the distributive law: P ∧ (Q ∨ R) ≡ (P ∧ Q) ∨ (P ∧ R)