VU Einführung in Artificial Intelligence

SS 2024

Hans Tompits

Institut für Logic and Computation
Forschungsbereich Wissensbasierte Systeme

www.kr.tuwien.ac.at

Constraint Satisfaction Problems

Constraint Satisfaction Problems (CSPs)

- Standard search problem:
  - From the point of view of a search algorithm, a state is a "black box" with no discernible internal structure.
  - It is represented by a suitable data structure that can be accessed only by the *problem specific* routines:
    - the successor function,
    - the heuristic function,
    - and the goal test.
- *Constraint satisfaction problem* (CSP):
  - The states and the goal test conform to a standard, structured, and simple representation.
  - Search algorithms can be defined that take advantage of the structure of states and use *general-purpose* rather than *problem-specific* heuristics.

Constraint Satisfaction Problems (ctd.)

- In a constraint satisfaction problem
  - a *state* is defined by *variables* with *values* from an associated *domain*, and
  - the goal test is a set of *constraints* specifying allowable combinations of values for subsets of variables.
- Example of a simple *formal representation language*
  - allows useful general-purpose algorithms with more power than standard search algorithms.

CSP: Formal Definition

A *constraint satisfaction problem* (CSP) consists of the following components:

- a finite set $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$ of variables;
- each variable $V_i \in \mathcal{V}$ has an associated non-empty domain D_i of possible values;
- a finite set $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ of constraints.
  - A constraint $C \in \mathcal{C}$ between variables $V_{i_1}, \ldots, V_{i_j}$ is a subset of the Cartesian product

$$D_{i_1} \times \cdots \times D_{i_j} = \{(d_1, \ldots, d_j) \mid d_l \in D_{i_l}, 1 \leq l \leq j\}.$$

CSP: Formal Definition (ctd.)

- Each constraint limits the values that variables can take, e.g., $V_1 \neq V_2$.
- There are constraints of different arities:
  - *n*-ary constraints restrict the possible assignment of n variables, i.e., n-ary constraints are *n-ary relations*.
  - In particular:
    - Unary constraints restrict the domain D_i of one variable V_i. E.g., $C(V_i) = \{1, 3, 5, 7, 8\}$.
    - Binary constraints restrict the domains $D_i \times D_j$ of a pair of variables V_i, V_j. E.g., $C(V_i, V_j) = \{(1, 2), (3, 5), (7, 3), (8, 2)\}$.
    - Ternary constraints,...

CSP: Further notions

- A state of a CSP is defined by an *assignment* of values to some or all of the variables.
- An assignment that does not violate any constraints is *consistent* or *legal*.
- An assignment is *complete* iff it mentions every variable.
- A *solution* to a CSP is a complete consistent assignment, i.e., a function which assigns
  1. each variable a value of its associated domain and
  2. such that all constraints are satisfied.
- Some CSPs also require a solution that maximises an objective function
  - ➥ these are called *constrained optimisation problems*.

Example: Map-colouring

Consider the task of colouring a map of Australia with the colours red, green, and blue such that no neighbouring region have the same colour.

Example: Map-colouring (ctd.)

We can formulate this problem as the following CSP:

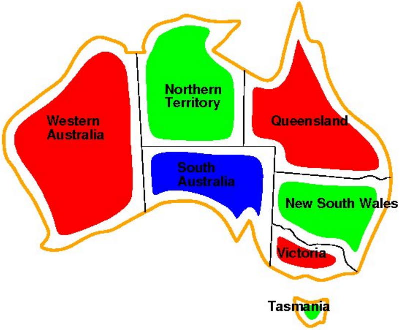
- *Variables:* $\mathcal{V} = \{WA, NT, Q, NSW, V, SA, T\}$
- *Domains:* $D_i = \{red, green, blue\}$, $i \in \mathcal{V}$
- *Constraints:* adjacent regions must have different colors
  - e.g., the allowable combinations of WA and NT are

$$C(WA, NT) = \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\},$$

  - or simply written as $WA \neq NT$ (if the language allows this).

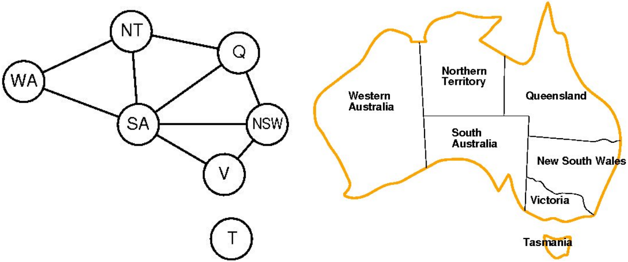
Example: Map-colouring (ctd.)

There are many possible solutions, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$



Constraint graph

- For a *binary CSP* (in which all constraints are binary), it is helpful to visualise the problem as a constraint graph:
  - the nodes are the variables,
  - the edges correspond to the constraints, i.e., there is an edge between two variables if there is a constraint involving them.
- E.g., our map-colouring problem has the following constraint graph:



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

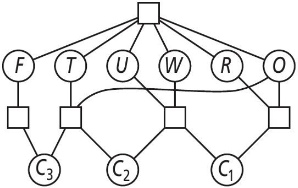
Constraint graph (ctd.)

- Higher-order constraints can be represented by a *constraint hypergraph*.
  - Reminder: a hypergraph is a pair (X, E), where X is a set of nodes and E is a set of non-empty subsets of X, the hyperedges.
- *Cryptarithmetic puzzles* are examples of involving higher-order constraints.
  - Usually, one assumes that each letter in a cryptarithmetic puzzle represents a different digit.

Constraint graph (ctd.)

Example:

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



- This is formulated as the following CSP:
  - Variables: F, T, U, W, R, O, C_1, C_2, C_3
  - Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - Constraints:
    - $Alldiff(F, T, U, W, R, O)$;
    - addition constraints:
      $O + O = R + 10 \cdot C_1$,
      $C_1 + W + W = U + 10 \cdot C_2$,
      $C_2 + T + T = O + 10 \cdot C_3$,
      $C_3 = F$.
- A solution for this CSP is, e.g., $938 + 938 = 1876$.

Varieties of CSPs

- The simplest kind of CSPs involves variables that are *discrete* and have *finite domains*.
  - E.g., map-colouring problems are of this kind.

- If the maximum domain size of any variable in a CSP is d, and there are n variables, then the number of possible complete assignments is $O(d^n)$
  - ➥ exponential in the number of variables!

Varieties of CSPs (ctd.)

- Finite domain CSPs whose variables can be either true or false are called *Boolean CSPs*.
- E.g., 3SAT can be expressed as a Boolean CSP
  - a clause like $X_1 \vee \neg X_2 \vee X_3$ corresponds to the constraint

$$C(X_1, X_2, X_3) = \\ (\{true, false\} \times \{true, false\} \times \{true, false\}) \setminus \{(false, true, false)\}.$$

- Since 3SAT is an NP-complete problem we cannot expect to solve finite-domain CSPs in less than exponential time (unless P = NP).
- However, in most *practical* applications, CSP algorithms can solve problems orders of magnitude larger than those solvable via general search algorithms.

Varieties of CSPs (ctd.)

- Discrete variables can also have *infinite domains*, e.g., the set of integers or the set of strings.
  - E.g., for construction job scheduling, variables are the start dates and the possible values are integer numbers of days from the current date.

- Note:
  - With infinite domains it is no longer possible to describe constraints by enumerating all allowed combinations of values.
  - Rather, a constraint language must be used.
    - E.g., if Job_1, which takes 5 days, must precede Job_3, then we need a language of algebraic inequalities like $StartJob_1 + 5 \leq StartJob_3$.

Varieties of CSPs (ctd.)

- It is also no longer possible to solve constraints with infinite domains by enumerating all possible assignments
  - ➥ there are infinitely many of them!
- Special solution algorithms exist for *linear constraints* on integer values
  - linear constraint = variables appear only in *linear* form
  - e.g., $StartJob_1 + 5 \leq StartJob_3$ is linear.
- Non-linear constraints are *undecidable*—no algorithm exists for solving such constraints!

Varieties of CSPs (ctd.)

- Finally, there are CSPs with *continuous domains*
  - very common in real-world applications and widely studied in operations research
  - e.g., scheduling the start/end times for the Hubble Space Telescope
    - require a very precise timing of observations,
    - taking a variety of real-valued astronomical, precedence, and power constraints into account.

- Linear constraints can be solved with *linear programming* methods in polynomial time.

Some real-world CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Floor planning

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

CSPs as standard search problems

➤ It is straightforward to give an *incremental formulation* of a CSP as a standard search problem.

- States are defined by the values assigned so far.
- Initial state: the empty assignment, $\emptyset$.
- Successor function: assign a value to an unassigned variable providing it does not conflict with the current assignment.
- Goal test: the current assignment is complete.

➤ This is the same for all CSPs!

➥ Any standard search algorithm can be used to solve CSPs.

CSPs as standard search problems (ctd.)

Caveat: Suppose we use breadth-first search.

- If there are n variables and d values, the branching factor at the top level is nd.
- At the next level, the branching factor is $(n-1)d$, and so on for n levels.
- ➥ We generate a tree with $n!d^n$ leaves although there are only d^n possible complete assignments!

Backtracking search

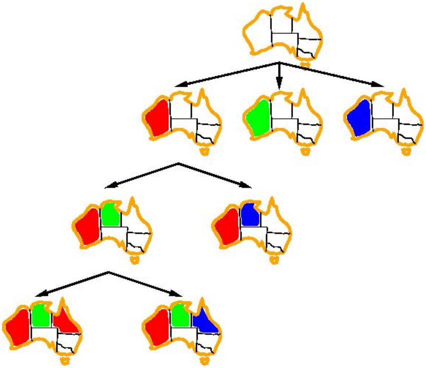
- The naive formulation ignored one crucial property of CSPs:
  - Variable assignments are *commutative*, i.e., the order of an assignment of variables does not matter and one reaches the same partial assignment regardless of order.
  - Therefore, CSP search algorithms need only to consider a single variable at each node of the search tree!
    - E.g., in the map-colouring problem, initially we may have a choice between $SA = red$, $SA = green$, and $SA = blue$,
    - but we would not choose between $SA = red$ and $WA = blue$.

  ➥ With this restriction, we generate only d^n leaves as expected.

- Depth-first search for CSPs with single-variable assignments is called *backtracking* search.
  - Backtracking search is the basic uninformed algorithm for CSPs.

Backtracking search (ctd.)

Below gives part of the search tree for the Australia problem, where the variables are assigned in the order WA, NT, Q, ...

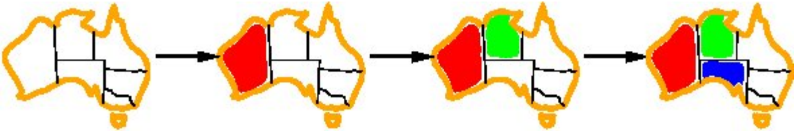


Backtracking search (ctd.)

- Since plain backtracking search is an uninformed algorithm, we do not expect it to be very effective for large problems.
- Different *general-purpose methods* help improving the performance, addressing the following issues:
  - Which variable should be assigned next, and in what order should its values be tried?
  - What are the implications of the current variable assignments for the other unassigned variables?
  - When a path fails, can the search avoid repeating this failure in subsequent paths?

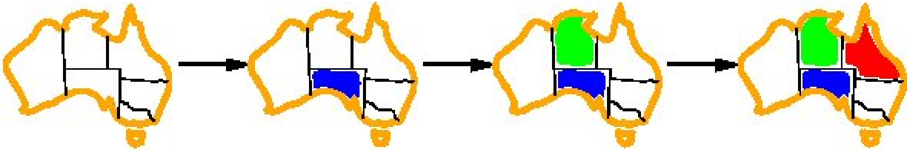
Minimum-remaining-values heuristic

- The minimum-remaining-values (MRV) heuristic:
  - choose the variable with the fewest legal values.
- If there is a variable X with 0 legal values remaining, the MRV heuristic will select X and failure will be detected immediately
  - avoiding pointless searches through further unassigned variables.
- E.g., in the Australia example, after the assignments for $WA = red$ and $NT = green$, there is only one possible value for SA.
  - It makes sense to assign $SA = blue$ next rather than assigning Q.
  - Actually, after SA is assigned, the choices for Q, NSW, and V are all forced.



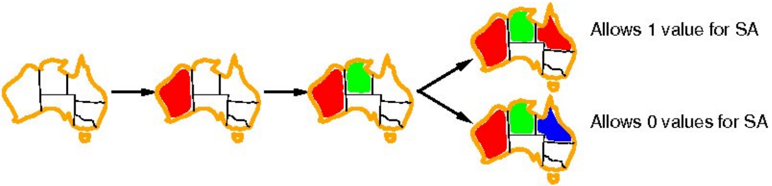
Degree heuristic

- The MRV heuristic does not help at all in choosing the *first* region to colour.
- In this case, the degree heuristic comes in:
  - it selects the variable that is involved in the *largest number of constraints* on other unassigned variables.
- In the Australia example, *SA* is the variable with highest degree, 5.
  - The others have degree 0, 2, or 3.
  - Actually, once *SA* is chosen, we can can assign the mainland regions clockwise or counterclockwise with a colour different from *SA* and the previous region.



Least-constraining-value heuristic

- Once a variable has been selected, to decide on the order in which to examine its values, the least-constraining-value heuristic can be effective:
  - it prefers a value that rules out the *fewest* choices for the neighbouring variables in the constraint graph.
- In the Australia example, suppose we have the partial assignment $WA = red$ and $NT = green$, and our next choice is for Q.
  - Blue would be a bad choice, because it eliminates the last legal value for Q's neighbour SA.
  - ➥ The least-constraining-value heuristic thus prefers red to blue.

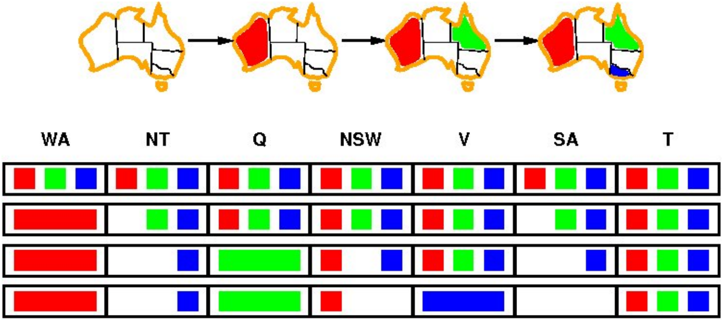


Forward checking

- The methods discussed so far consider the constraints on a variable *only at the time that the variable is chosen.*
- By looking at some of the constraints earlier in the search, or even before the search, the search space can be drastically reduced.
- One such method is forward checking:
  - whenever a variable X is assigned, it looks at each unassigned variable Y that is connected to X by a constraint
  - and deletes from the domain of Y any value that is inconsistent with the value chosen for X.

Forward checking (ctd.)

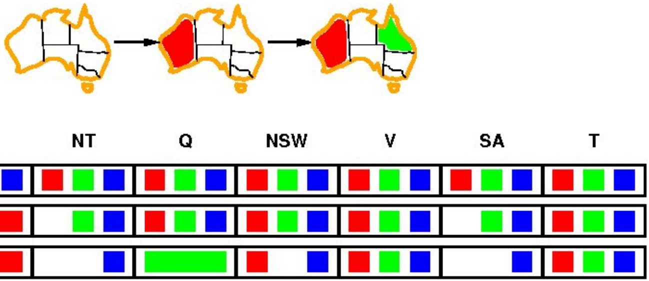
- Consider colouring Australia using forward checking:



- Note:
  - After assigning $WA = red$ and $Q = green$, the domains of NT and SA are reduced to a single value.
    - ➥ The MRV heuristic would select SA and NT next.
  - After assigning $V = blue$, the domain of SA is empty, so we get failure and the algorithm backtracks.

Forward checking (ctd.)

- Forward checking does not provide early detection for all failures:

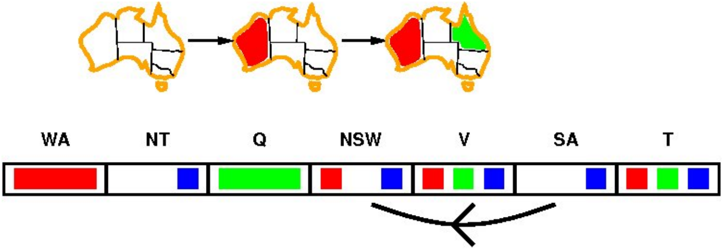


- *NT* and *SA* cannot both be blue!

☞ Constraint propagation is the general term for propagating the implications of a constraint on one variable onto other variables.

Arc consistency

- The simplest form of constraint propagation is arc consistency:
  - "arc" refers to a *directed* arc in the constraint graph;
  - $X \to Y$ is *consistent* iff for *every* value x of X there is *some* allowed value y of Y.
- For $SA = blue$ in the Australia colouring, there is a consistent assignment for NSW, namely red $\implies$ the arc from SA to NSW is consistent
  - the reverse arc is *not* consistent, but can be made so by deleting blue from the domain of NSW.



Further techniques

- Intelligent backtracking:
  - do not backtrack to the preceding variable if a failure occurs, but go back to one in the set of variables that *caused the failure*
    - this set is the conflict set
    - e.g., backjumping goes to the most recent variable in this conflict set.
- Local search algorithms are very effective for solving CSPs
  - the *million*-queens problem can be solved in an average of 50 steps.
- The structure of the constraint graph can be taken into account.
  - E.g., colouring Tasmania is an independent subproblem of colouring Australia.
  - Tree-structured problems can be solved in linear time.

Knowledge Representation

Knowledge-based Agents

What is knowledge representation?

- The *representation of knowledge* and *reasoning from knowledge* are central for AI
  . . . after all, humans know things and do reasoning.
- Knowledge and reasoning play a crucial role in dealing with *partially observable* environments.
  - A knowledge-based agent can combine general knowledge with current percepts to infer hidden aspects of the current state prior to selecting actions.
    - E.g., a physician diagnoses a patient prior to choosing a treatment.
    - ➥ For diagnosing, the physician uses knowledge from education and experience, as well as association patterns the physician cannot consciously describe.

What is knowledge representation? (ctd.)

- Understanding natural language also involves inferring hidden states—viz., the *intention of the speaker*.
  - E.g., when we hear

    "John threw the stone against the mirror and broke it",

    we know that "it" refers to "mirror" and not to "stone".

- In general, the goal of knowledge representation is the following:
  - representing implicit knowledge about a certain area in such a way that it can be processed by computers
  - original knowledge is encoded in suitable data structures and algorithms.

What is knowledge representation? (Ctd.)

- Knowledge representation is a multidisciplinary field involving methods and techniques from:
  - *logic:*
    - provides the formal structures and rules for performing deductions;
  - *ontology:*
    - defines the kinds of objects in the considered application area;
  - *computer science:*
    - supports the applications which distinguishes knowledge representation from pure philosophy.
- In short:
  - *knowledge representation* = application of logic and ontology for providing computational models.

Declarative vs. procedural approaches

- *Declarative knowledge representation techniques:*
  - knowledge is expressed as sentences in some suitable formal language which are accessed by the procedures using this knowledge
    - ➥ separation between the *explicit representation of knowledge* and the *processing* for answering queries.
  - Advantages:
    - increased versatility for performing complex tasks;
    - changes can be easily incorporated (modularity).
- *Procedural techniques:*
  - knowledge is *implicitly stored* in a sequence of operations, manifested in the actual execution of the operations (i.e., directly as program code).
  - Advantages: minimising the role of explicit representation and reasoning can yield more efficient systems.

Declarative vs. procedural approaches (ctd.)

- In the 1970s and 1980s there were heated debates between advocates of the two approaches.
- Now it is understood that successful agents often combine both declarative and procedural elements in their designs.

Knowledge-based agents

- Central components of a knowledge-based agent:
  - a *knowledge base*
    - a set of *sentences* in a formal language;
  - methods to *add new sentences* and methods to *query what is known*.
    - We use TELL and ASK as generic names for these tasks.
    - Both tasks may involve *inference*—i.e., deriving new sentences from old.

      ☞ In *logical agents*, answers to the ASK procedure is by means of logic!
- Schematic architecture:

$\boxed{\text{knowledge processing}}$	$\longleftarrow$	domain-independent algorithms
$\updownarrow$		
$\boxed{\text{knowledge base}}$	$\longleftarrow$	domain-specific content

A simple knowledge-based agent

- The agent must be able to:
  - represent states, actions, etc.;
  - incorporate new percepts;
  - update internal representations of the world;
  - deduce hidden properties of the world;
  - deduce appropriate actions.

- Each time the agent program is called, it does three things:
  1. It TELLS the knowledge base what it perceives;
  2. it ASKS the knowledge base what action it should perform;
  3. it records its choice with TELL and executes the action.

A simple knowledge-based agent (ctd.)

```
function KB-AGENT( percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE( percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

- MAKE-PERCEPT-SENTENCE constructs a sentence asserting that the agent perceived the given percept at the given time.
- MAKE-ACTION-QUERY constructs a sentence that asks what action should be done at the current time.
- MAKE-ACTION-SENTENCE constructs a sentence asserting that the chosen action was executed.
  - ☞ Details of the inference mechanisms are hidden inside TELL and ASK!

Elements of Propositional and First-Order Logic

Logic in general

- *Logics* are formal languages for representing information such that conclusions can be drawn.
- *Syntax* defines the sentences in the language.
- *Semantics* defines the "meaning" of sentences; i.e., defines *truth* of a sentence in a world.
- For example, consider the language of arithmetic:
  - $x + 2 \geq y$ is a sentence;
  - $x2 + y >$ is not a sentence;
  - $x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y;
  - $x + 2 \geq y$ is true in a world where $x = 7,\ y = 1$;
  - $x + 2 \geq y$ is false in a world where $x = 0,\ y = 6$.

Entailment

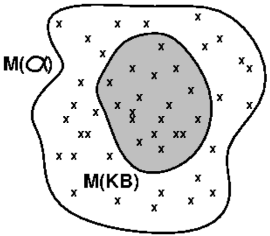
- *Entailment* means that one thing follows from another:
  - A knowledge base *KB* entails a sentence α, symbolically $KB \models \alpha$, iff α is true in all worlds where *KB* is true.
  - Here, *KB* is the *premiss* and α is the *conclusion* of the entailment.
  - Recall that knowledge bases are sets of sentences and they are also referred to as *theories*.

- Examples:
  - A knowledge base *KB* containing *"Batman laughs"* and *"Commodore Schmidlapp laughs"* entails *"Either Batman laughs or Commodore Schmidlapp laughs"*.
  - In the language of arithmetic, $x + y = 4$ entails $4 = x + y$.

☞ Entailment is a relationship between sentences (i.e., *syntax*) that is based on *semantics*.

Models

- Semantics is defined in terms of *interpretations*, which are formally structured worlds with respect to which truth can be evaluated.
- We say that interpretation m is a *model* of a sentence α if α is true in m, and m is a model of a knowledge base if it is a model of all its elements.
  - We denote by $M(\alpha)$ the set of all models of α.
- Then, $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$.
  - E.g., $KB \models \alpha$ holds for $KB =$ Batman laughs and Commodore Schmidlapp laughs and $\alpha =$ Commodore Schmidlapp laughs.



Important semantical notions

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

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

- A sentence is *valid* if *all* interpretations are models of it.
- A sentence is *satisfiable* if it has *some* model.
- A sentence is *unsatisfiable* if it has *no* model.
- Writing $\neg\alpha$ for the *negation* of α (with the meaning that $\neg\alpha$ is true precisely when α is not true), we can state:
  - α is valid if and only if $\neg\alpha$ is unsatisfiable;
  - $KB \models \alpha$ if and only if $KB \cup \{\neg\alpha\}$ is unsatisfiable, i.e., to prove α from KB by *reductio ad absurdum*.

Inference

- $KB \vdash_i \alpha :\Longleftrightarrow$ sentence α can be derived from KB in *proof system i*.
  - A proof system (also called *calculus* or *axiom system*), consists of *axioms* and *inference rules* (however, some proof systems do not require axioms).
  - A *derivation from KB* is a sequence of formulas s.t.
    (i) each formula is either an axiom,
    (ii) an element of KB, or
    (iii) results from inference rule applications using earlier elements in the sequence.
  - A derivation is also said to be a derivation *of* its last element.
- Intuitively:
  - Consequences of KB are a haystack; α is a needle.
    $\Longrightarrow$ Entailment = needle in haystack; inference = finding it

Inference (ctd.)

- Important properties:
  - *Soundness*:
    - i is sound if $KB \vdash_i \alpha$ implies $KB \models \alpha$.
  - *Completeness*:
    - i is complete if $KB \models \alpha$ implies $KB \vdash_i \alpha$.
- Many different sound and complete proof systems for various logics have been defined in the literature, like
  - Hilbert-type systems,
  - sequent-type calculi,
  - tableau calculi,
  - resolution calculi,
  - natural deduction systems, etc.

☞ Important in computer science are sequent-type calculi, tableau calculi, and resolution calculi.

Two fundamental logics

- Among the many different logics existing, designed for different purposes, two logics are pre-eminent:
  - *propositional logic*; and
  - *first-order logic* (FOL) (also called *predicate logic*).
- Propositional logic is simple, assuming that the world consists of facts which can be composed from atomic formulas using connectives:
  - $\neg S$ (*negation*), $S_1 \wedge S_2$ (*conjunction*), $S_1 \vee S_2$ (*disjunction*), $S_1 \Rightarrow S_2$ (*implication*), $S_1 \Leftrightarrow S_2$ (*biconditional*).
- E.g., $\neg A \Rightarrow (B \vee C)$ states that if A is not the case, then one of B or C holds.
  - This formula may represent, e.g., the following sentence:

    If the car is not proceeding, then it is broken or out of gas.

Truth tables for connectives

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*

Some logical equivalences in propositional logic

$$
\begin{aligned}
(\alpha \land \beta) &\equiv (\beta \land \alpha) && \text{commutativity of } \land \\
(\alpha \lor \beta) &\equiv (\beta \lor \alpha) && \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) &\equiv (\alpha \land (\beta \land \gamma)) && \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) &\equiv (\alpha \lor (\beta \lor \gamma)) && \text{associativity of } \lor \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \lor \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \land \beta) &\equiv (\neg\alpha \lor \neg\beta) && \text{De Morgan} \\
\neg(\alpha \lor \beta) &\equiv (\neg\alpha \land \neg\beta) && \text{De Morgan} \\
(\alpha \land (\beta \lor \gamma)) &\equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) && \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) &\equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) && \text{distributivity of } \lor \text{ over } \land
\end{aligned}
$$

Restricted Expressibility

- Unlike natural language, propositional logic has, however, only very limited expressive power.
  - E.g., the following argument (valid in natural language) cannot be adequately dealt with in propositional logic:

    All superheroes are brave.
    Superman is a superhero.
    Therefore: Superman is brave.

  - In propositional logic, the three sentences would be formalised using atomic sentences A, B, C—but $A, B \models C$ does *not* hold!

➥ This is where FOL comes in!

First-order logic (FOL)

FOL assumes that the world contains

- *Objects*: people, houses, numbers, theories, Superman, Commodore Schmidlapp, colours, centuries, ...
- *Relations*: red, round, bogus, prime, multistoried ...,
  brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
- *Functions*: father of, best friend, addition, one more than, end of ...

Syntax of FOL: Basic elements

Constants:	$\mathit{Superman}, \mathit{KingJohn}, 2, \ldots$;
Predicates:	$\mathit{Friend}, >, \ldots$;
Functions:	$\mathit{Sqrt}, \mathit{LeftLegOf}, \ldots$;
Variables:	$x, y, a, b, \ldots$;
Connectives:	$\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$;
Equality:	$=$;
Quantifiers:	$\forall$ (*universal quantifier*), $\exists$ (*existential quantifier*)

Atomic sentences

$$\text{Atomic sentence} := \mathit{predicate}(\mathit{term}_1, \ldots, \mathit{term}_n)$$
$$\text{or } \mathit{term}_1 = \mathit{term}_2$$

$$\text{Term} := \mathit{function}(\mathit{term}_1, \ldots, \mathit{term}_n)$$
$$\text{or } \mathit{constant} \text{ or } \mathit{variable}$$

Examples:

1. $\mathit{Friend}(\mathit{Superman}, \mathit{Batman})$;
2. $> (\mathit{Length}(\mathit{LeftLegOf}(\mathit{Superman})), \mathit{Length}(\mathit{LeftLegOf}(\mathit{Batman})))$

Complex sentences

- Complex sentences are made from atomic sentences using connectives and the quantifiers
  - $\forall x S$ (*universal quantifier*, *"for all x, S"*),
  - $\exists x S$ (*existential quantifier*, *"for some x, S"*).
- Examples:
  1. $\forall x(Archfiend(x, Superman) \Rightarrow Fights(x, Superman))$;
  2. $>(1,2) \vee \leq(1,2)$;
  3. $>(1,2) \wedge \neg>(1,2)$;
  4. $\forall x(Country(x) \Rightarrow \exists y\, Capitol(y, x))$.

Truth in first-order logic

- Sentences are true with respect to a *domain* and an *interpretation*.
  - The domain contains ≥ 1 objects (*domain elements*) for specifying relations among them.
  - The interpretation specifies referents over the domain for
    - constant symbols $\rightarrow$ objects;
    - predicate symbols $\rightarrow$ relations;
    - function symbols $\rightarrow$ functional relations.
- An atomic sentence $predicate(term_1, \ldots, term_n)$ is *true* iff the objects referred to by $term_1, \ldots, term_n$ are in the relation referred to by *predicate*.

Truth example

- Consider the formula $Brother(Richard, John)$ and the following interpretation:
  - $Richard \rightarrow$ Richard the Lionheart;
  - $John \rightarrow$ the evil King John;
  - $Brother \rightarrow$ the brotherhood relation.
- Under this interpretation, $Brother(Richard, John)$ is true just in case Richard the Lionheart and the evil King John are in the brotherhood relation.

Common mistakes to avoid

- Typically, $\Rightarrow$ is the main connective with $\forall$ as in:
  - all S are P: $\forall x(S(x) \Rightarrow P(x))$.
  - Common mistake: using $\wedge$ as the main connective with $\forall$:
    $$\forall x(At(x, Berkeley) \wedge Smart(x))$$
    means "everyone is at Berkeley and everyone is smart".

- Typically, $\wedge$ is the main connective with $\exists$ as in:
  - some S are P: $\exists x(S(x) \wedge P(x))$.
  - Common mistake: using $\Rightarrow$ as the main connective with $\exists$:
    $$\exists x(At(x, Stanford) \Rightarrow Smart(x))$$
    is true if there is anyone who is not at Stanford!

Some ambiguities

- In natural language, "all S are P" would normally not be asserted if it is already known that S does not hold.
- Indeed, people would not consider "all S are P" true if S is false.
  $\Longrightarrow$ "all S are P" would in this sense be translated as

  $$\exists x S(x) \wedge \forall x (S(x) \rightarrow P(x))$$

  rather than as $\forall x (S(x) \rightarrow P(x))$.

Some ambiguities (ctd.)

- Sometimes "all S are not-P" is understood as "not all S are P". Example:
    - "All that glisters is not gold" (Shakespeare, Merchant of Venice).
    - ➥ Translation would be of the form $\neg\forall x(S(x) \rightarrow P(x))$ but *not* of the form $\forall x(A(x) \rightarrow \neg P(x))$.

- The indefinite article "a" or "an" has sometimes different meaning:
    - "A child needs affection." $\Longrightarrow \forall x(C(x) \rightarrow A(x))$.
    - "A man climbed the Mount Everest." $\Longrightarrow \exists x(M(x) \wedge E(x))$.

Some ambiguities (ctd.)

- Also, the meaning of the expression "any" depends on the context:
  - When an any-expression stands by itself, it has the same force as "all".
  - But when an any-expression D is put into contexts $\neg D$ or $D \rightarrow E$, the meaning of "any" normally changes from "all" to "some".
- Examples:
  - "I would do that for anyone." $\Longrightarrow \forall x A(x)$.
  - "I wouldn't do that for anyone." $\Longrightarrow \neg\exists x A(x)$.
  - "Anyone who is godfearing is just." $\Longrightarrow \forall x(G(x) \rightarrow J(x))$.
  - "If any man is just, Aristides is just." $\Longrightarrow (\exists x J(x)) \rightarrow J(a)$.
  - "If Superman is a villain, then any man is a villain." $\Longrightarrow V(s) \rightarrow \forall x V(x)$.