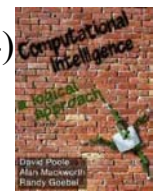
Chapter 5: Representing Knowledge

- Introduction (5.1)
- Defining a Solution (5.2)
- Choosing a Representation Language (5.3)
- Mapping from Problem to Representation (5.4)



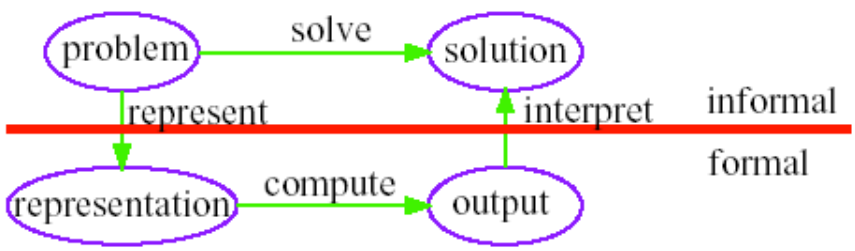
D. Poole, A. Mackworth, and R. Goebel, *Computational Intelligence: A Logical Approach*, Oxford University Press, January 1998

Introduction (5.1)

- Representing Knowledge
  - Given a problem to solve, how do you solve it?
    - What is a solution to the problem?
    - What do you need in the language to represent the problem?
    - How can you map from the informal problem description to a representation of the problem?
    - What distinctions in the world are important to solve the problem?
    - What knowledge is required?

Introduction (5.1) (cont.)

- What level of detail is required?
- What reasoning strategies are appropriate?
- Is worst-case performance or average-case performance the critical time to minimize?
- Is it important for a human to understand how the answer was derived?
- How can you acquire the knowledge from experts or from experience?
- How can the knowledge be debugged, maintained, and improved?



Knowledge Representation Framework

Defining a Solution (5.2)

- Given an informal description of a problem, you need to determine what would constitute a solution
  Delivery Robot: Do not take everything to the garbage can when asked to take all of the trash to the garbage can
- Typically much is left unspecified, but the unspecified parts can't be filled in arbitrarily
- Much work in AI is motivated by common-sense reasoning. You want the computer to be able to make common-sense conclusions about the unstated assumptions (ways to specify defaults!)

Defining a Solution (5.2) (cont.)

- Quality of Solutions
  - Does it matter if the answer is wrong or answers are missing or extra answers?
  - Classes of solution
    - Optimal solution: the best solution according to some measure of solution quality (to miss some trash than to waste too much time: utility)
    - Satisficing solution: one that is good enough, according to some description of which solutions are adequate
    - Approximately optimal solution: one whose measure of quality is close to the best theoretically possible (robot minimal distance approximated!)
    - Probable solution: one that is likely to be a solution (be 80% sure the robot has taken 3 items of trash after the robot has dropped the trash or failed to pick it up)

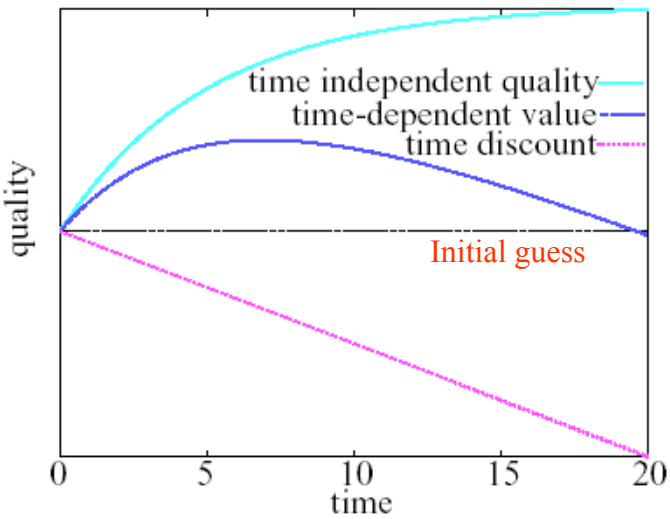
Defining a Solution (5.2) (cont.)

- Decisions & Outcomes
  - Good decisions can have bad outcomes (bad decisions can have good outcomes)
    (Robot travel fast near the top of the stairs! It might fall down…)
  - Information can be valuable because it leads to better decisions: value of information
  - A good decision is one that generally leads to a good outcome

Defining a Solution (5.2) (cont.)

- Info Availability & Solution Quality
  - Costs are associated with activities of an agent
    $\Rightarrow$ solution quality
  - You have to trade off computation time and solution quality: an anytime algorithm can provide a solution at any time; given more time it can produce better solutions (e.g., hill-climbing algorithms)
  - You don't only need to be concerned about finding the right answer, but about acquiring the appropriate information, and computing it in a timely manner
  - When defining a solution, you have to be concerned about what information is available, how information can be obtained & the cost & benefits associated with actions used to obtain the information

Anytime algorithm



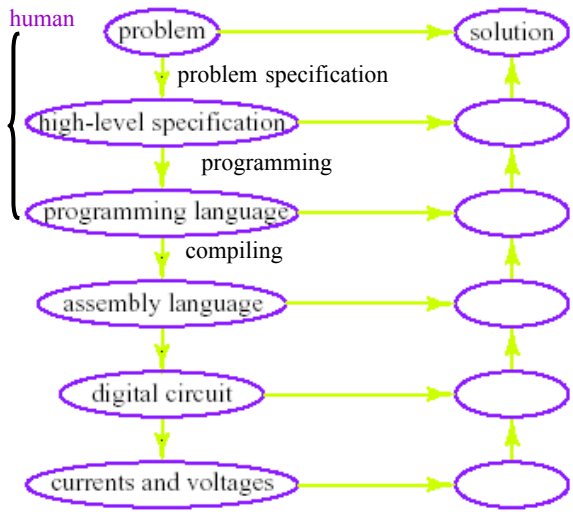
Solution Quality & Computation Time

Choosing a Representation Language (5.3)

- You need to represent a problem to solve it on a computer

problem
→ specification of problem
→ appropriate computation

- Example representations: CILog/Prolog, C++, English
- A logic is a language + specification of what follows from input in that language (syntax + semantics + reasoning system)



Hierarchy of Representations from a Soft. Eng. View

Mapping from Problem to Representation (5.4)

- What level of abstraction of the problem do you want to have to represent?
- What objects and relations in the world do you want to represent?
- How can you represent the knowledge to ensure that the representation is natural, modular, and maintainable?

Mapping from Problem to Representation (5.4) (cont.)

- Levels of Representations
  - Two levels of abstraction seem to be common among biological (brain) and computational (computer) entities
    - Knowledge level in terms of what an agent knows and what an agent's goals are, not knowing how the reasoning is done
    - Symbol level in terms of what symbols the agent is manipulating
  - The knowledge level is about the external world to the agent
  - The symbol level is about what symbols an agent uses to implement the knowledge level (inside view of an agent in order to reason about the external world)

Mapping from Problem to Representation (5.4) (cont.)

- Choosing a Level of Abstraction
  - A high-level description is easier for a human to specify and understand
  - A low-level description can be more accurate and more predictive (high-level descriptions abstract away details that may be important for actually solving the problem
  - The lower the level, the more difficult it is to reason with
  - You may not know the information needed for a low-level description
  - It is sometime possible to use multiple levels of abstraction

Mapping from Problem to Representation (5.4) (cont.)

- Choosing Objects & Relations
  - Goal: Given a logic & a world, you want to choose what in the world you want to refer to: ontology of the domain
  - How to represent: "Pen #7 is red."
    - *Red(pen_7).* It's easy to ask "What's red?"
      Can't ask "what is the color of pen_7?"
    - *Color(pen_7, red).* It's easy to ask "What's red?"
      It's easy to ask "What is the color of pen_7?" (through a variable X)
      Can't ask "What property of pen_7 has value *red*?"
    - *prop(pen_7, color, red).* It's easy to ask all these questions
  - *Prop(Object, Attribute, Value)* is the only relation needed: object-attribute-value representation

Mapping from Problem to Representation (5.4) (cont.)

- Universality of *prop*

To represent "a is a parcel"
prop(a, is_a, parcel), where *is_a* is a special relation (attribute is_a)
prop(a, parcel, true), where *parcel* is a Boolean attribute (property of individuals)

prop(object, attribute, value)

To represent *scheduled(cs*422, 2, 1030, *cc*208). "section 2 of course *cs*422 is scheduled at 10:30 in room cc208."
Let *b*123 name the booking:

prop(b123, course, cs422).
prop(b123, section, 2).
prop(b123, time, 1030).
prop(b123, room, cc208).

Easy to add prop(b123, instructor, craig)

Mapping from Problem to Representation (5.4) (cont.)

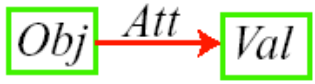
- Semantics Network

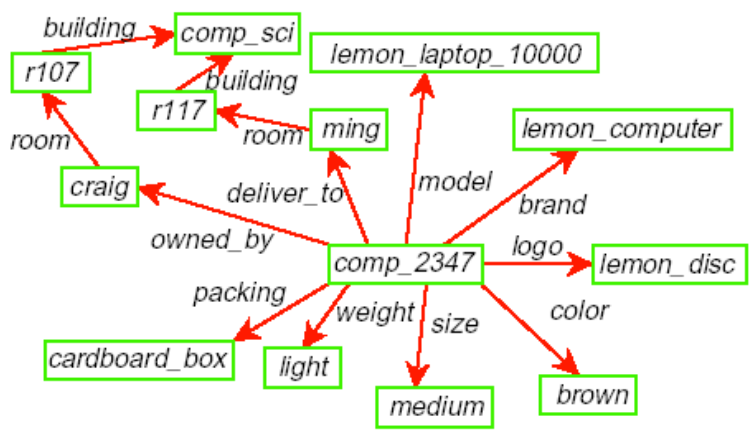
When you only have one relation, prop, it can be omitted without loss of information

Write

Prop(Obj, Att, Value)

As





An Example of Semantic Network

Mapping from Problem to Representation (5.4) (cont.)

- Equivalent Logic Program

info to a robot:

prop(comp_2347, owned_by, craig).
prop(comp_2347, deliver_to, ming).
prop(comp_2347, model, lemon_laptop_10000).
prop(comp_2347, brand, lemon_computer).
prop(comp_2347, logo, lemon_disc).
prop(comp_2347, color, brown).
prop(craig, room, r107).
prop(r107, building, comp_sci).
…

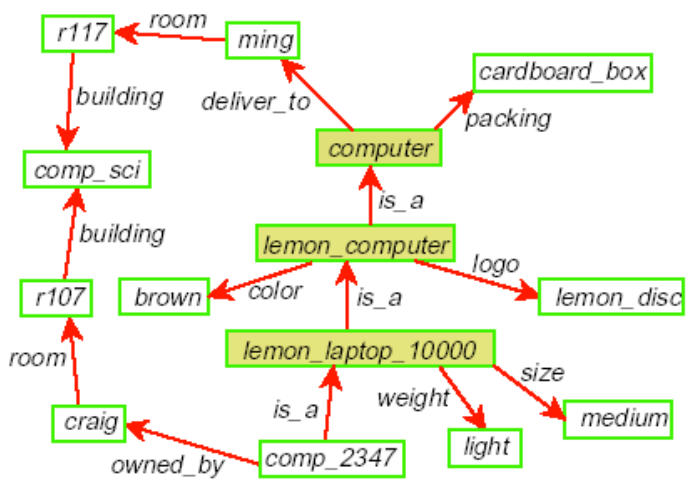
Mapping from Problem to Representation (5.4) (cont.)

- Frames
  - The properties and values for a single object can be grouped together into a frame
  - We can write this as a list of *attribute* (or slots) and values (or fillers)

Comp_2347
[owned_by = craig,
deliver_to = ming,
model = lemon_laptop_10000,
brand = lemon_computer,
logo = lemon_disc,
color = brown,
…]

Mapping from Problem to Representation (5.4) (cont.)

- Primitive versus Derived Relations
  - Primitive knowledge is that which is defined explicitly by facts
  - Derived knowledge is knowledge defined by rules (example: All lemon laptops may have *size* = *medium*
  - Associate this property with the class, not the individual
  - Allow a special attribute *is_a* between an individual and a class or between two classes that allows for property inheritance



A Structured Semantic Network

Mapping from Problem to Representation (5.4) (cont.)

- Logic of Property Inheritance
  - An arc $\overset{p}{\rightarrow}\mathbf{n}$ from a class c means every individual in the class has value n of attribute p.

    prop(Obj, p, n) ←
    prop(Obj, is_a, c).

  - Example:

    prop(X, weight, light) ←
    prop(X, is_a, lemon_laptop_10000).
    prop(X, is_a, lemon_computer) ←
    prop(X, is_a, lemon_laptop_10000).