Problem-Solving Methods

BMI 210A / CS 270A

Mark A. Musen

Stanford Medical Informatics
Stanford University

The battles of the 1970s

- Are frame systems better than more general semantic networks?
- Are frames useful if there is no associated description logic?
- Are rule-based systems better than frame-based systems?
- Isn't logic programming better than any of these less complete approaches?
- Who wants a representation system that may not allow for decidable inference?

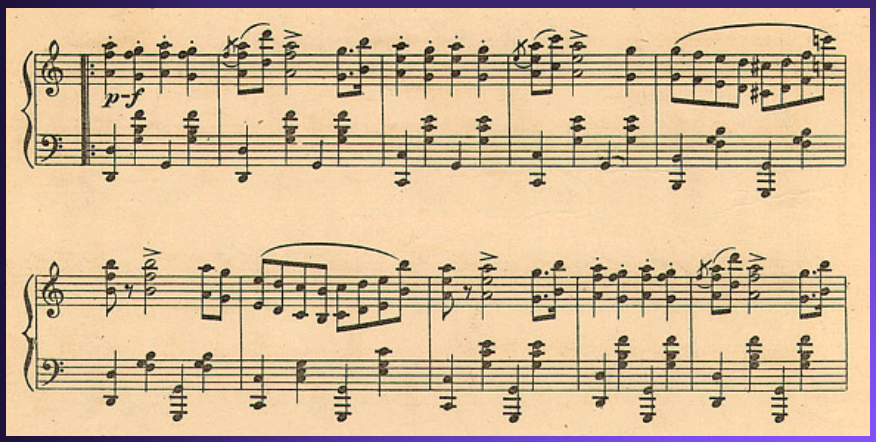
Allen Newell's AAAI Presidential Address (1980)

- We should stop bickering about representation
- What really matters is the *knowledge* that a system has, not how that knowledge is represented
- Knowledge is what an observer attributes to an agent to allow the observer to call that agent intelligent

The important distinction

- Knowledge representations are symbols such that, when some process is applied, an observer attributes intelligence to the emergent behavior
  - Knowledge representations acquire meaning only when there is some process that is applied to them; representations are symbols that must be interpreted
- Knowledge is a competence for intelligent behavior
  - Knowledge is inferred by observing an agent's behaviors; knowledge ultimately is something that is experienced and attributed

An analogy:
The notes are not the music



Knowledge has some degree of structure

- The goals that an agent has
- The actions of which an agent is capable
- How the agent selects actions to help it achieve its goals

The "knowledge level" (Newell, 1982)

- Computer systems can be viewed at discrete, hierarchical levels, where each level consists of
  - A medium that is processed
  - Components that provide primitive processing
  - Laws of composition
  - Laws of behavior
- Each level can be defined either
  - Autonomously
  - In terms of the components of the level below it

A hierarchy of computer-system levels

Intelligent behavior

Symbol level — Knowledge representations

Hardware level — Machine-level instructions

The Symbol Level

- Systems: Computer programs
- Medium: Symbols, expressions
- Components: Memory stores, operations
- Composition laws: Designation, association
- Behavior laws: Sequential interpretation

The Knowledge Level

- Systems: Agents
- Medium: Knowledge
- Components: Goals, actions, bodies of knowledge
- Composition laws: None; an agent has just the three components
- Behavior laws: The principle of rationality

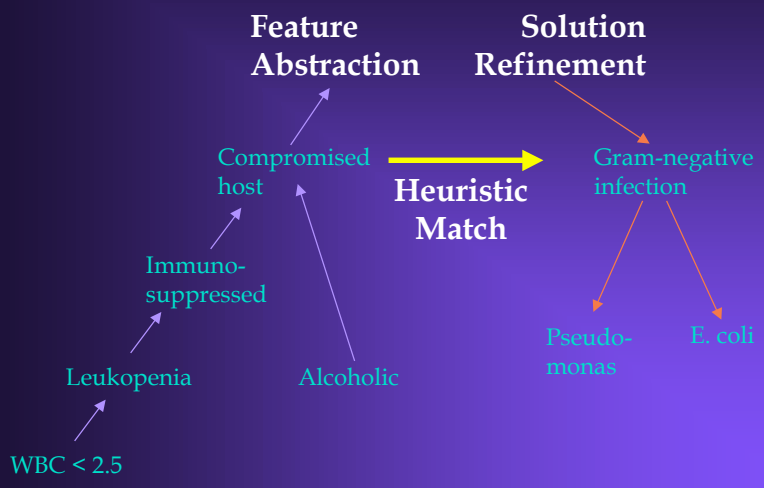
Newell's thoughts on the knowledge level

- Knowledge and rationality are intimately tied together
- Splitting what was once a single level into two allows each one to be addressed technically
- Knowledge is not representable by a structure at the symbol level; it requires both structures and processes.
- Knowledge is an abstraction that can never be had *in hand.*

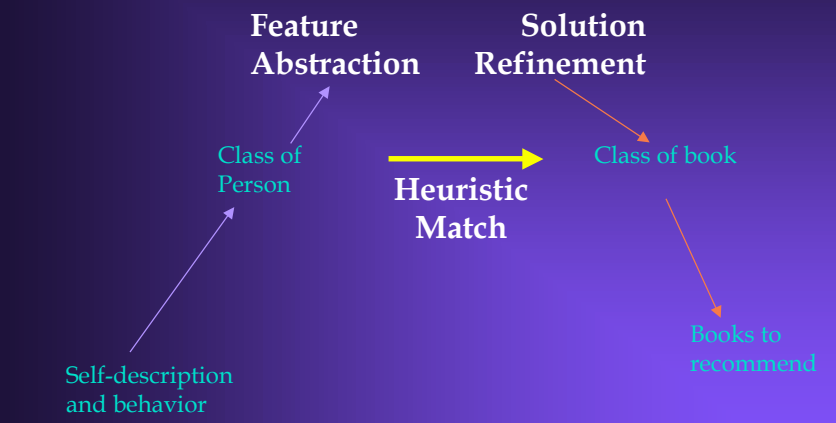
Knowledge-level analysis

- Ability to understand intelligent behavior in terms of
  - Goals
  - Actions
  - Bodies of knowledge
- Makes underlying knowledge representation irrelevant

Heuristic classification in MYCIN (after Clancey)



Heuristic classification in GRUNDY (after Clancey)



Problem-Solving Methods (PSMs)

- Provide abstract procedures for solving stereotypical *tasks*
- Offer a set of terms (an ontology) for talking about the problem-solving behavior
- Make precise the *roles* in which domain knowledge is used in problem solving

Some tasks for which developers have made PSMs:

- Classification
- Fault diagnosis
- Constraint satisfaction
- Planning
- Design
- Scheduling

A key distinction:

- Inference Engines
  (e.g., backward chaining) are procedures that operate on *data structures*
  (e.g., rules, frames)
- Problem-Solving Methods
  (e.g., heuristic classification) are procedures that operate on *ontologies*

Families of Problem-Solving Methods

- Classification
  Solutions are *selected* from a pre-enumerated set
- Construction
  Solutions are *created* during problem solving

Matching a task to candidate problem-solving methods

- Given a problem, there may be more than one appropriate PSM
- For example, we can multiply two numbers by
  - Using multiplication tables
  - Performing repeated addition
  - Adding logarithms
- Selecting a PSM is highly tied to
  - How a developer conceptualizes a problem
  - What knowledge may be available at knowledge-acquisition time
  - What data may be available at run time

A case study: The Propose and Revise method

- VT (Vertical Transportation) was a knowledge-based system developed by Marcus and McDermott (CMU) to configure elevators in new buildings
- VT used the Propose-and-Revise problem-solving method
  - As a generic, underlying reasoning strategy
  - To ensure that, as designs are extended, constraints are not violated:
    - Available parts must work together
    - Architectural requirements must be satisfied
    - Building codes may not be violated

Propose and Revise

1. Select a **procedure** to extend a configuration and identify **constraints** on the extension
2. Identify **constraint** violations; if none, go to Step 1.
3. Suggest potential **fixes** for the constraint violation.
4. Select the least costly **fix** not yet attempted.
5. Modify the configuration; identify **constraints** on the **fix.**
6. Identify **constraint** violations due to the **fix**; if any, go to Step 4.
7. Remove extensions incompatible with the revision.
8. If the configuration is incomplete, go to Step 1.

Propose-and-revise has a simple method ontology:

- Procedure
- Constraint
- Fix

Building a tool for knowledge entry into VT: SALT

- **SALT** (kNowledge ACquisition Language) was developed to allow reuse of the "propose and revise" problem-solving method
- SALT framed all knowledge in VT in terms of the *knowledge roles* entailed by propose-and-revise (the propose-and-revise method ontology)
- Developers conceptualized application tasks in terms of "propose and revise" method ontology
- SALT would output knowledge bases as OPS5 rules

SALT Dialog

1. PROCEDURE Enter a procedure for a value
2. CONSTRAINT Enter constraints for a value
3. FIX Enter remedies for a constraint violation
4. EXIT Exit interviewer

Enter your command [EXIT]: **1**

1. Name: **HOIST-CABLE-QUANTITY**
2. Precondition **NONE**
3. Procedure: **DATABASE-LOOKUP**
4. Table name: **HOIST-CABLE**
5. Column with value: **QUANTITY**
6. Parameter test: **MAX-LOAD > CAR-WEIGHT**
7. Parameter test: **DONE**
8. Ordering column: **QUANTITY**
9. Optimal: **SMALLEST**
10. Justification: **THIS ESTIMATE IS THE SMALLEST HOIST CABLE QUANTITY THAT CAN BE USED ON ANY JOB**

SALT allows knowledge-level analysis

- User conceptualizes content knowledge using terms of Propose-and-Revise method ontology (a knowledge-level description)
- SALT prompts user for "procedures", "constraints" and "fixes"
- SALT generates OPS5 symbols that can carry out the knowledge-level description

Reuse of Propose-and-Revise

- SALT has been used to build knowledge bases for systems for
  - Elevator design (Westinghouse)
  - Scheduling of flight simulator (Boeing)
- Propose-and-revise has been useful for constraint-satisfaction tasks for which backtracking to avoid unsatisfied constraints is not a frequent problem
- Propose-and-revise is a *terrible* method when backtracking takes place often

"Method-to-Task" approach embodied by SALT

- Propose-and-revise method ontology provides language for expressing problem-solving behavior (e.g., *constraint*, *fix*)
- Knowledge-acquisition tool (e.g., SALT) allows developer to instantiate method ontology to enter domain knowledge
- Benefit: All knowledge defined in terms of knowledge roles (e.g., *constraint*, *fix*), allowing modeling of knowledge at "the knowledge level"
- Problem: There is no explicit domain ontology

Another example: MOLE (Eshelman, 1986)

- MOLE, like SALT
  - Is a knowledge-acquisition system
  - Assumes a particular PSM (in this case, one called Cover-and-Differentiate)
- Generates systems that, when given a device model and a set of abnormal symptoms, perform fault diagnosis by
  - Identifying candidate faults that could explain symptoms
  - Differentiating among the candidates to select the best explanation

The PSM: Cover-and-Differentiate

- For each symptom, propose a set of covering explanations
- Seek runtime data that will differentiate among these alternatives
- Select the most parsimonious explanation

Acquiring initial symptoms

- List possible complaints or symptoms that might need to be diagnosed
- >> High-fly-ash-flow
- >> High-bottom-ash-flow
- >> Dark-ash
- >> Loss-in-gas
- >>

Acquiring covering knowledge

- List possible explanations for LOSS-IN-GAS:
- >> low-heat-transfer
- >> Excess-air high
- >>
- List possible explanations for LOW-HEAT-TRANSFER:
- >> misbalance-of-convection
- >> low radiation
- >>

Acquiring differentiating knowledge

LOW-HEAT-TRANSFER is explained by the following possible explanations:

MISBALANCE-OF-CONVECTION

LOW-RADIATION

Which of the following would be relevant evidence for preferring one of the explanations over the others:

1. LARGE-PARTICLES favoring LOW RADIATION
2. FOULING favoring MISBALANCE-OF-CONVECTION
3. EXCESS-AIR LOW favoring MISBALANCE-OF-CONVECTION

List the relevant response:

>> 2 3

As a result of this dialog:

- MOLE elicits
  - covering knowledge to build a network of symptoms, their potential explanations, explanations of those explanations, etc.
  - differentiating knowledge to tease apart situations when a symptom or intermediary explanation has more than one root explanation
- MOLE can transform the knowledge-level model of symptoms and explanations into a symbol-level representation in OPS5

Role-limiting PSMs

- Define an enumerable set of *knowledge roles* by which domain knowledge guides problem solving
- Help to ensure the adequacy of elicited knowledge: If a role is not satisfied by domain knowledge, then there the knowledge base must be incomplete
- Aid knowledge-base maintenance: The purpose of every entry in the knowledge base is clarified by its link to some knowledge role

During the 1980s

- Investigators began to identify dozens of abstract problem-solving methods (PSMs), hoping that each might be reusable for a variety of tasks
- In general, PSMs were identified by building a conventional knowledge-based system and then abstracting the control knowledge that the system used
- Often, PSMs formed the basis of knowledge-acquisition tools like SALT and MOLE
- The approach worked as long as a new task could be solved by a single PSM

But think about MYCIN …

- **One PSM**, heuristic classification, solves the subtask of identifying the organisms for which therapy is needed
- **A second PSM**, "the therapy-planning algorithm", *constructs* a set of antibiotics to administer, based on
  - likely sensitivities of the organisms
  - treatment heuristics (e.g., the tetracycline rule)
  - parsimony
- The "goal rule" invokes a set of backward-chaining rules in its premise to initiate the heuristic classification component; it calls a LISP function in its conclusion to invoke the therapy-planning algorithm

MYCIN's Goal Rule

IF:

1) Information has been gathered about organisms isolated from this patient, organisms noted on smears taken from this patient, negative cultures of this patient, suspected infections without microbiological evidence, current drugs of this patient, and prior drugs of this patient,
2) An attempt has been made to deduce the organisms which require therapy, and
3) You have given consideration to organisms (other than those noted in cultures and smears) that might be present

THEN:

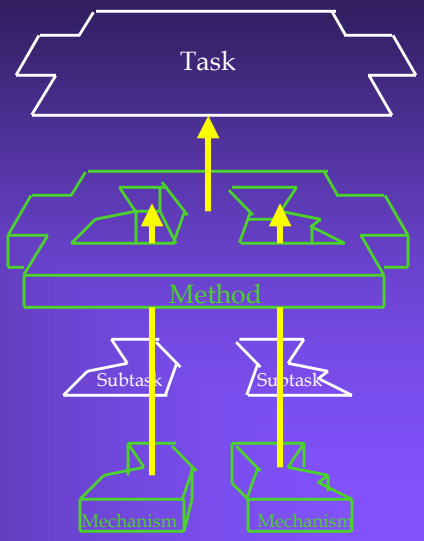
Determine the best therapy recommendation from among the drugs likely to be effective against the significant organisms, or indicate that no therapy is required at this time

Application tasks seldom can be solved by a single PSM

- Most real-world tasks are heterogeneous
- Need PSMs to decompose tasks into more homogeneous subtasks
- Need appropriate control structures to allow appropriate problem solving

PSMs and task decomposition

- Tasks are solved by PSMs
- PSMs may entail subtasks, which themselves are solved by PSMs, which themselves may entail subtasks, which …
- *Primitive* PSMs sometimes are called mechanisms



Problem-solving methods

- Assume that required problem solving can be construed as compositions of well-characterized, generic algorithms
- Provide terms and relationships for talking about
  - problem-solving behavior
  - data on which that behavior operates
- Encode such terms and relationships as a *method* ontology

The goal: Libraries of problem-solving methods that

- Serve as repositories of reusable algorithms
- Aid rapid development of new problem-solving systems
- Codify the results of careful analysis of biomedical problem-solving tasks

The next steps in this course:

- Learning how problem solving methods can be linked to domain ontologies to build real systems
- Understanding some specific problem-solving methods that are particularly important in biomedicine