Classification Problem Solving.
William J. Clancey,
Jul 1984

Heuristic Programming Project
Memo (HPP 84-7)

Revised (July 1984)

CLASSIFICATION PROBLEM SOLVING

William J. Clancey
Heuristic Programming Project
Computer Science Department
Stanford University
Stanford, CA 94305

Reprinted with permission from the AAAI 1984 Proceedings, published by the American Association for Artificial Intelligence.

Abstract

A broad range of heuristic programs—embracing forms of diagnosis, catalog selection, and skeletal planning—accomplish a kind of well-structured problem solving called classification. These programs have a characteristic inference structure that systematically relates data to a pre-enumerated set of solutions by abstraction, heuristic association, and refinement. This level of description specifies the knowledge needed to solve a problem, independent of its representation in a particular computer language. The classification problem-solving model provides a useful framework for recognizing and representing similar problems, for designing representation tools, and for understanding the problem-solving methods used by non-classification programs.

1. Introduction

Over the past decade a variety of heuristic programs have been written to solve problems in diverse areas of science, engineering, business, and medicine. Yet, presented with a given "knowledge engineering tool," such as EMYCIN [39], we are still hard-pressed to say what kinds of problems it can be used to solve well. Various studies have demonstrated advantages of using one representation language instead of another—for ease in specifying knowledge relationships, control of reasoning, and perspicuity for maintenance and explanation [9, 38, 1, 2, 10]. Other studies have characterized in low-level terms why a given problem might be inappropriate for a given language, for example, because data are time-varying or subproblems interact [21]. But attempts to describe *kinds of problems* in terms of shared features have not been entirely satisfactory: Applications-oriented descriptions like "diagnosis" are too general (e.g., the program might not use a device model), and technological terms like "rule-based" don't describe what problem is being solved [18, 19]. Logic has been suggested as a tool for a "knowledge level" analysis that would specify what a heuristic program does, independent of its implementation in a programming language [30, 29]. However, we have lacked a set of terms and relations for doing this.

In an attempt to specify in some canonical terms what many heuristic programs known as "expert systems" do, an analysis was made of ten rule-based systems (including MYCIN, SACON, and The Drilling Advisor), a frame-based system (GRUNDY) and a program coded directly in LISP (SOPHIE III). There is a striking pattern: These programs proceed through easily identifiable phases of data abstraction, heuristic mapping onto a hierarchy of pre-enumerated solutions, and refinement within this hierarchy. In short, these programs do what is commonly called *classification*.

Focusing on content rather than representational technology, this paper proposes a set of terms and relations for describing the knowledge used to solve a problem by classification. Subsequent

sections describe and illustrate the classification model in the analysis of MYCIN, SACON, GRUNDY, and SOPHIE III. Significantly, a knowledge level description of these programs corresponds very well to psychological models of expert problem solving. This suggests that the classification problem solving model captures general principles of how experiential knowledge is organized and used, and thus generalizes some cognitive science results. There are several strong implications for the practice of building expert systems and continued research in this field.

2. Classification problem solving defined

We develop the idea of classification problem solving by starting with the common sense notion and relating it to the reasoning that occurs in heuristic programs.

2.1. Simple classification

As the name suggests, the simplest kind of classification problem is to identify some unknown object or phenomenon as a member of a known class of objects or phenomena. Typically, these classes are stereotypes that are hierarchically organized, and the process of identification is one of matching observations of an unknown entity against features of known classes. A paradigmatic example is identification of a plant or animal, using a guidebook of features, such as coloration, structure, and size.

Some terminology we will find helpful: The *problem* is the object or phenomenon to be identified; *data* are observations describing this problem; possible *solutions* are patterns (variously called schema, frames or units); each solution has a set of *features* (slots or facets) that in some sense describe the concept either categorically or probabilistically; solutions are grouped into *a specialization hierarchy* based on their features (in general, not a single hierarchy, but multiple, directed acyclic graphs); a *hypothesis* is a solution that is under consideration; *evidence* is data that partially matches some hypothesis; the *output* is some solution.

The essential characteristic of a classification problem is that the problem solver selects from a set of pre-enumerated solutions. This does not mean, of course, that the "right answer" is necessarily one of these solutions, just that the problem solver will only attempt to match the data against the known solutions, rather than construct a new one. Evidence can be uncertain and matches partial, so the output might be a ranked list of hypotheses.

Besides matching, there are several *rules of inference* for making assertions about solutions. For example, evidence for a class is indirect evidence that one of its subtypes is present. Conversely,

given a closed world assumption, evidence against all of the subtypes is evidence against a class. *Search operators* for finding a solution also capitalize on the hierarchical structure of the solution space. These operators include: *refining* a hypothesis to a more specific classification; *categorizing* the problem by considering superclasses of partially matched hypotheses; and *discriminating* among hypotheses by contrasting their superclasses [31, 32, 12]. For simplicity, we will refer to the entire process of applying these rules of inference and operators as *refinement*. The specification of this process—a control strategy—is an orthogonal issue which we will consider later.

2.2. Data abstraction

In the simplest problems, data are solution features, so the matching and refining process is direct. For example, an unknown organism in MYCIN can be classified directly given the supplied data of gram stain and morphology.

For many problems, solution features are not supplied as data, but are inferred by *data abstraction*. There are three basic relations for abstracting data in heuristic programs:

- *qualitative abstraction* of quantitative data ("if the patient is an adult and white blood count is less than 2500, then the white blood count is low");
- *definitional abstraction* ("if the structure is one-dimensional of network construction, then its shape is a beam"); and
- *generalization* in a subtype hierarchy ("if the client is a judge, then he is an educated person").

These interpretations are usually made by the program with certainty; thresholds and qualifying contexts are chosen so the conclusion is categorical. It is common to refer to this knowledge as "descriptive," "factual," or "definitional."

2.3. Heuristic classification

In simple classification, the data may directly match the solution features or may match after being abstracted. In heuristic classification, solution features may also be matched heuristically. For example, MYCIN does more than identify an unknown organism in terms of features of a laboratory culture: It heuristically relates an abstract characterization of the patient to a classification of diseases. We show this *inference structure* schematically, followed by an example (Figure 2-1).

Basic observations about the patient are abstracted to patient categories, which are heuristically linked to diseases and disease categories. While only a subtype link with E.coli infection is shown

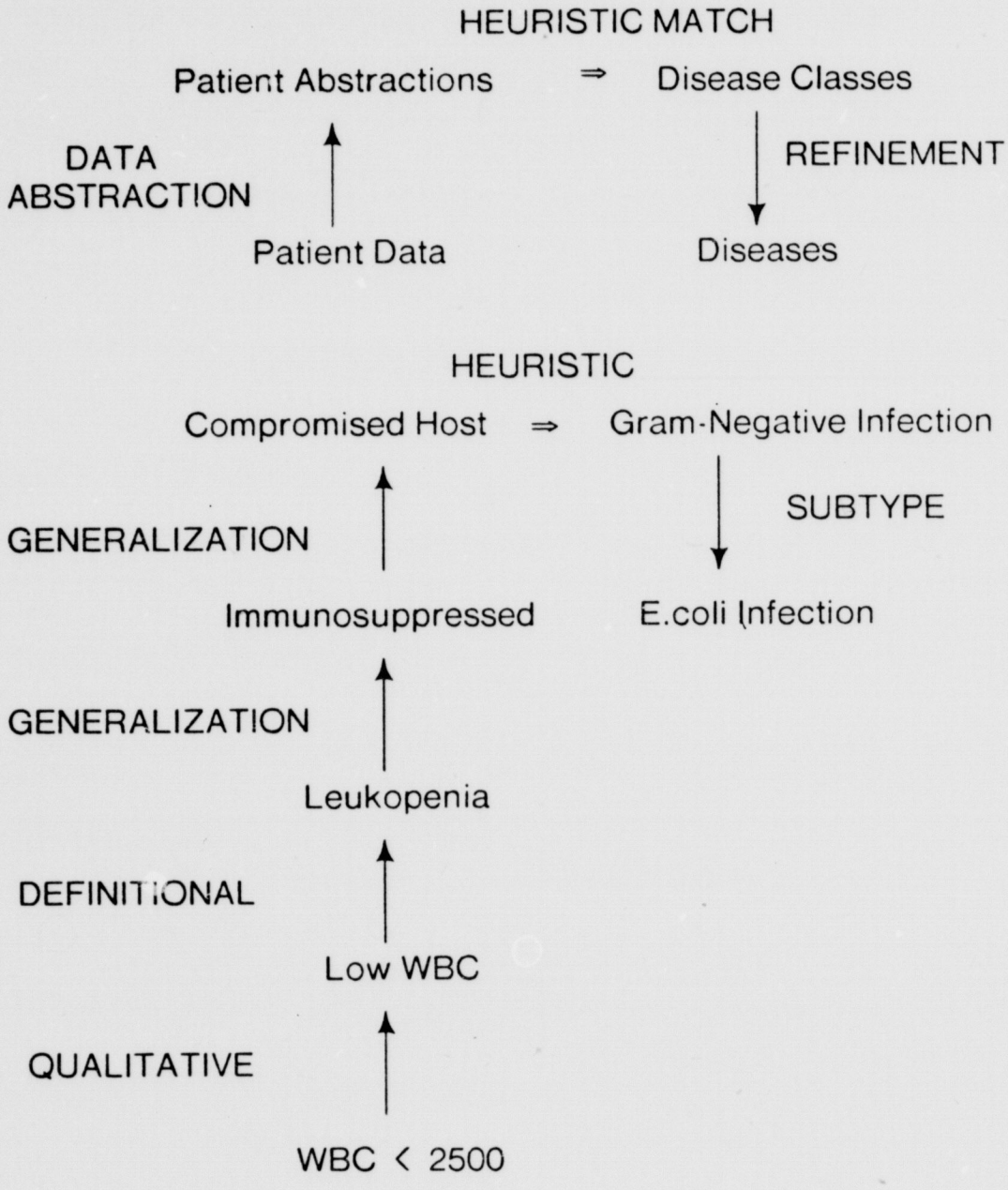

Figure 2-1: Inference structure of MYCIN

here, evidence may actually derive from a combination of inferences. Some data might directly match E.coli by identification. Discrimination with competing subtypes of gram-negative infection might also provide evidence. As stated earlier, the order in which these inferences are made is a matter of control strategy.

The important link we have added is a heuristic association between a characterization of the patient ("compromised host") and categories of diseases ("gram-negative infection"). Unlike the factual and hierarchical evidence propagation we have considered to this point, this inference makes a great leap. A *heuristic relation* is based on some implicit, possibly incomplete, model of the world. This relation is often empirical, based just on experience; it corresponds most closely to the "rules of thumb" often associated with heuristic programs [14].

Heuristics of this type reduce search by skipping over intermediate relations (this is why we don't call abstraction relations "heuristics"). These associations are usually uncertain because the intermediate relations may not hold in the specific case. Intermediate relations may be omitted because they are unobservable or poorly understood. In a medical diagnosis program, heuristics typically skip over the causal relations between symptoms and diseases.

To repeat, classification problem solving involves heuristic association of an abstracted problem statement onto features that characterize a solution. This can be shown schematically in simple terms (Figure 2-2).

This diagram summarizes how a distinguished set of terms (data, data abstractions, solution abstractions, and solutions) are related systematically by *different* kinds of relations and rules of inference. This is the *structure of inference* in classification problem solving.

In a study of physics problem solving, Chi [8] calls data abstractions "transformed" or "second order problem features." In an important and apparently common variant of the simple model, data abstractions are themselves patterns that are heuristically matched. In essence, there is a sequence of classification problems. GRUNDY, analyzed below, illustrates this.

2.4. Search in classification problem solving

The issue of search is orthogonal to the kinds of inference we have been considering. "Search" refers to how a network made up of abstraction, heuristic, and refinement relations is interpreted, how the flow of inference actually might proceed in solving a problem. Following Hayes [18], we call this the *process structure*. There are three basic process structures in classification problem solving:

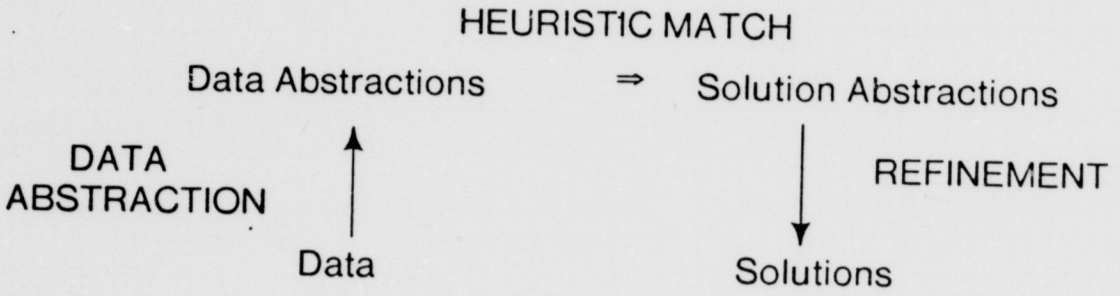

Figure 2-2: Classification problem solving inference structure

1. *Data-directed search*: The program works forwards from data to abstractions, matching solutions until all possible (or non-redundant) inferences have been made.

2. *Solution- or Hypothesis-directed search*: The program works backwards from solutions, collecting evidence to support them, working backwards through the heuristic relations to the data abstractions and required data to solve the problem. If solutions are hierarchically organized, then categories are considered before direct features of more specific solutions.

3. *Opportunistic search*: The program combines data and hypothesis-directed reasoning [20]. Data abstraction rules tend to be applied immediately as data become available. Heuristic rules "trigger" hypotheses, followed by a focused, hypothesis-directed search. New data may cause refocusing. By reasoning about solution classes, search need not be exhaustive.

Data- and hypothesis-directed search are not to be confused with the implementation terms "forward" or "backward chaining." R1 provides a superb example of how different implementation and knowledge level descriptions can be. Its rules are *interpreted* by forward-chaining, but it does a

form of hypothesis-directed search, systematically setting up subproblems by a fixed procedure that focuses reasoning on spatial subcomponents of a solution [28].

The degree to which search is *focused* depends on the level of indexing in the implementation and how it is exploited. For example, MYCIN's "goals" are solution classes (e.g., types of bacterial meningitis), but selection of rules for specific solutions (e.g., E.coli meningitis) is unordered. Thus, MYCIN's search within each class is unfocused [11].

The choice of process structure depends on the number of solutions, whether they can be categorically constrained, usefulness of data (the density of rows in a data/solution matrix), and the cost for acquiring data.

3. Examples of classification problem solving

Here we schematically describe the architectures of SACON, GRUNDY, and SOPHIE III in terms of classification problem solving. These are necessarily very brief descriptions, but reveal the value of this kind of analysis by helping us to understand what the programs do. After a statement of the problem, the general inference structure and an example inference path are given, followed by a brief discussion.

3.1. SACON

Problem: SACON [3] selects classes of behavior that should be further investigated by a structural analysis simulation program (Figure 3-1).

Discussion: SACON solves two problems by classification—analyzing a structure and then selecting a program. It begins by heuristically selecting a simple numeric model for analyzing a structure (such as an airplane wing). The model produces stress and deflection estimates, which the program then abstracts in terms of features that hierarchically describe different configurations of the MARC simulation program. There is no refinement because the solutions to the first problem are just a simple set of possible models, and the second problem is only solved to the point of specifying program classes. (In another software configuration system we analyzed, specific program input parameters are inferred in a refinement step.)

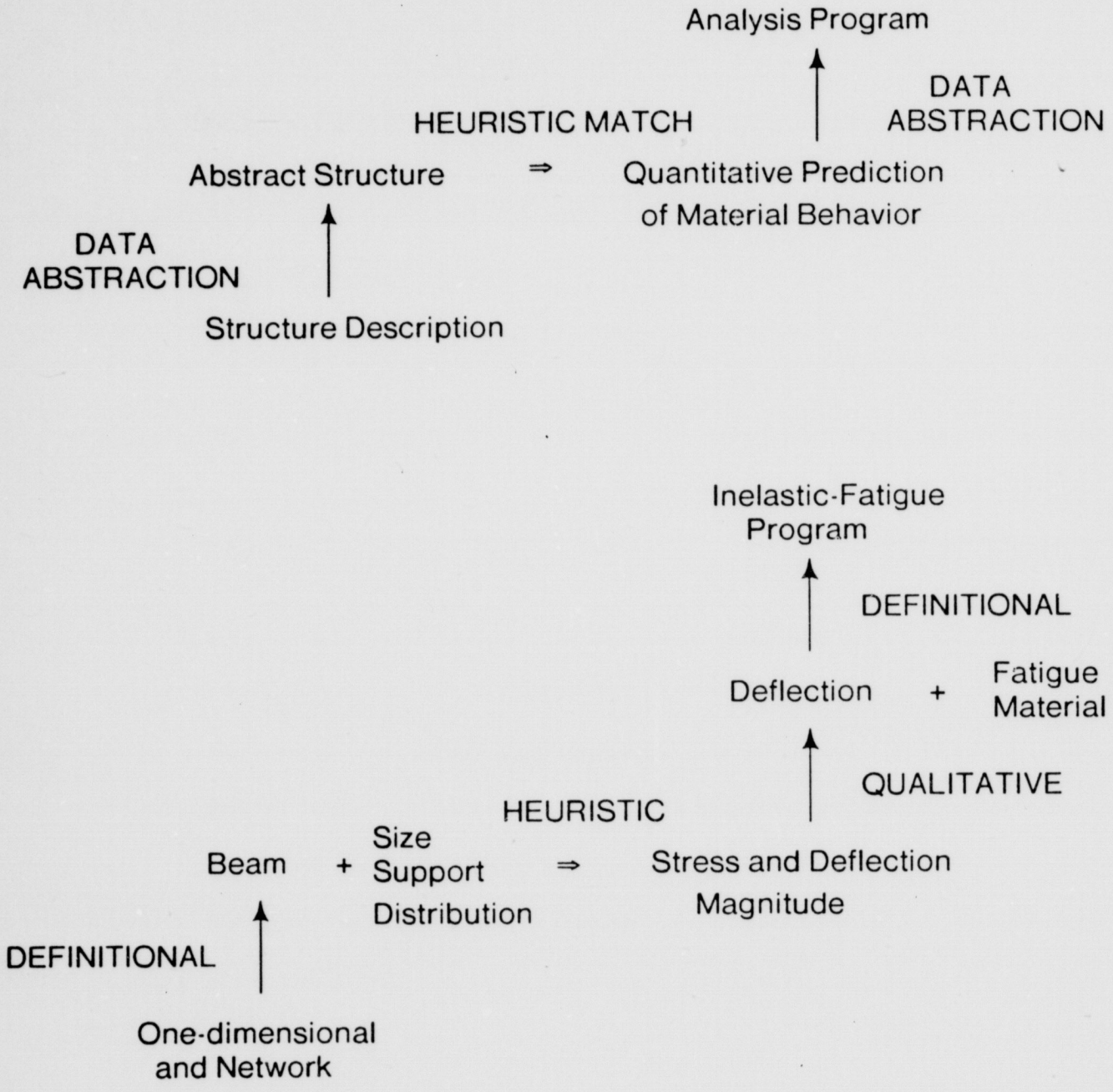

Figure 3-1: Inference structure of SACON

3.2. GRUNDY

Problem: GRUNDY [33] heuristically classifies a reader's personality and selects books he might like to read (Figure 3-2).

Discussion: GRUNDY solves two classification problems heuristically. Illustrating the power of a knowledge level analysis, we discover that the people and book classifications are not distinct in the implementation. For example, "fast plots" is a book characteristic, but in the implementation "likes fast plots" is associated with a person stereotype. The relation between a person stereotype and "fast plots" is heuristic and should be distinguished from abstractions of people and books. One objective of the program is to learn better people stereotypes (user models). The classification description of the user modeling problem shows that GRUNDY should also be learning better ways to characterize books, as well as improving its heuristics. If these are not treated separately, learning may be hindered. This example illustrates why a knowledge level analysis should precede representation.

It is interesting to note that GRUNDY does not attempt to perfect the user model before recommending a book. Rather, refinement of the person stereotype occurs when the reader rejects book suggestions. Analysis of other programs indicates that this multiple-pass process structure is common. For example, the Drilling Advisor makes two passes on the causes of sticking, considering general, inexpensive data first, just as medical programs commonly consider the "history and physical" before laboratory data.

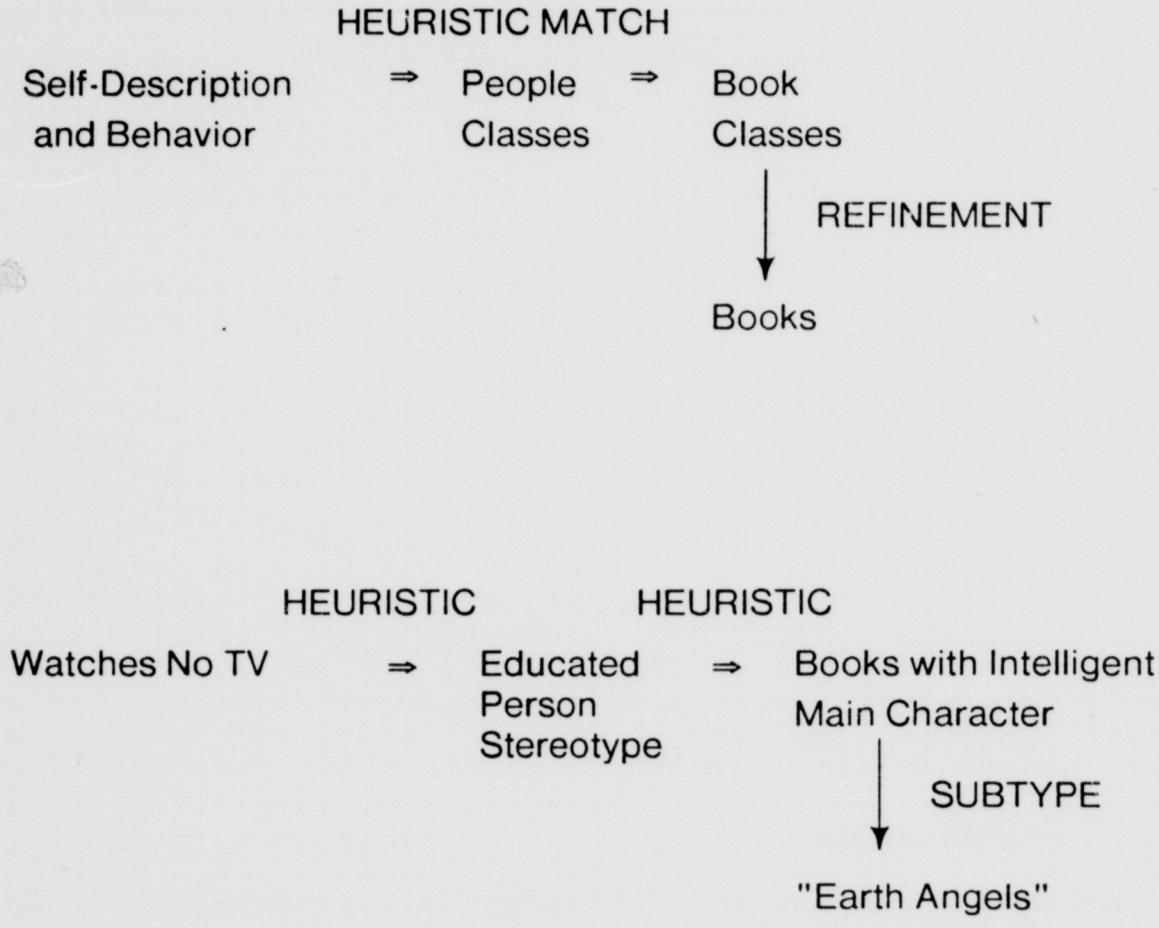

Figure 3-2: Inference structure of GRUNDY